IBM DB2 Information Integrator

Data Source Configuration Guide

Version 8.2

IBM DB2 Information Integrator

Data Source Configuration Guide

Version 8.2

Before using this information and the product it supports, be sure to read the general information under “Notices” on page 651.

Contents

About this book

This book describes how to configure a federated system to access data sources. This book contains:

- An introduction to federated system concepts, components, and capabilities.
- Recommendations for planning the configuration of the data sources.
- Instructions for registering the objects required for the federated server and database to access data sources.
- Extensive reference information specific for each data source including: server setup information, environment variable requirements, data type mappings, and options you can set to customize and tune the data source configuration.

Locating technical changes
: Technical changes to the text are indicated by a vertical line to the left of the change.

Who should read this book

This book is intended for system administrators, database administrators, security administrators, and system operators who need to set up and configure a federated system. Use this book to manage a federated system to access data from relational and nonrelational data sources. This book can also be used by programmers and other users who require an understanding of the configuration and use of a federated system. This book assumes that you are familiar with DB2. You should be familiar with standard database terminology, and have experience with database design and database administration. This book assumes that you are familiar with your own applications and the data sources that you want access.

Conventions and terminology used in this book

Federated terminology:

The glossary in this book defines the terms that are used when discussing federated systems.

DB2 Commands:

This book assumes that DB2 commands are entered in the DB2 Command Line Processor (CLP) or the DB2 Command Center GUI, unless otherwise specified. When a DB2 command is mentioned in the text, only the command and its parameters are listed. The command will not be proceeded by DB2.

DB2 Control Center:

The documentation indicates when tasks can be performed by using the DB2 Control Center and includes the steps to perform these tasks in this documentation.

Highlighting Conventions:

This book uses these highlighting conventions:

Boldface type
: Indicates commands and graphical user interface controls (such as names of fields, names of push buttons, and menu choices). Boldface type is used to designate notes, restrictions, prerequisites, and recommendations.

`Monospace type`
: Indicates text that you type, file names, and code examples. Monospace type is also used for SQL statement or DB2 command parameter names.

Italic type
: Indicates SQL statement or DB2 command parameter values that you replace with an appropriate value. SQL statement or DB2 command examples use italic type for sample parameter values. Italic type is used to emphasize words, to identify new terms, and to indicate document titles.

UPPERCASE TYPE
: Indicates the names of DB2 commands and SQL statements, and their keywords. Uppercase is also used for data type names, options, and acronyms.

How to read the syntax diagrams

Throughout this book, syntax is described using the structure defined as follows:

Read the syntax diagrams from left to right and top to bottom, following the path of the line.

The ►►── symbol indicates the beginning of a statement.

The ──► symbol indicates that the statement syntax is continued on the next line.

The ►── symbol indicates that a statement is continued from the previous line.

The ──►◄ symbol indicates the end of a statement.

Required items appear on the horizontal line (the main path).

```
►►──STATEMENT──required item──────────────────────────────────────────►◄
```

Optional items appear below the main path.

```
►►──STATEMENT──┬──────────────┬───────────────────────────────────────►◄
               └─optional item─┘
```

If an optional item appears above the main path, that item has no effect on the execution of the statement and is used only for readability.

```
               ┌─optional item─┐
►►──STATEMENT──┴──────────────┴───────────────────────────────────────►◄
```

If you can choose from two or more items, they appear in a stack.

If you *must* choose one of the items, one item of the stack appears on the main path.

```
►►──STATEMENT──┬─required choice1─┬──────────────────────────────────────────►◄
               └─required choice2─┘
```

If choosing none of the items is an option, the entire stack appears below the main path.

```
►►──STATEMENT──┬──────────────────┬──────────────────────────────────────────►◄
               ├─optional choice1─┤
               └─optional choice2─┘
```

If one of the items is the default, it will appear above the main path and the remaining choices will be shown below.

```
               ┌─default choice──┐
►►──STATEMENT──┼─────────────────┼───────────────────────────────────────────►◄
               ├─optional choice─┤
               └─optional choice─┘
```

An arrow returning to the left, above the main line, indicates an item that can be repeated. In this case, repeated items must be separated by one or more blanks.

```
               ┌─────────────────┐
               ▼                 │
►►──STATEMENT────repeatable item─┴───────────────────────────────────────────►◄
```

If the repeat arrow contains a comma, you must separate repeated items with a comma.

```
               ┌─,───────────────┐
               ▼                 │
►►──STATEMENT────repeatable item─┴───────────────────────────────────────────►◄
```

A repeat arrow above a stack indicates that you can make more than one choice from the stacked items or repeat a single choice.

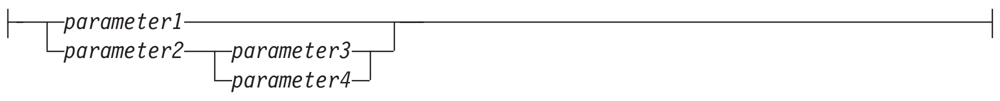
Keywords appear in uppercase (for example, `FROM`). They must be spelled exactly as shown. Variables appear in lowercase (for example, `column-name`). They represent user-supplied names or values in the syntax.

If punctuation marks, parentheses, arithmetic operators, or other such symbols are shown, you must enter them as part of the syntax.

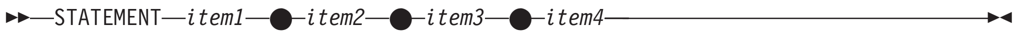
Sometimes a single variable represents a set of several parameters. For example, in the following diagram, the variable `parameter-block` can be replaced by any of the interpretations of the diagram that is headed **parameter-block**:

```
►►──STATEMENT──┤ parameter-block ├───────────────────────────────────────────►◄
```

parameter-block:



Adjacent segments occurring between “large bullets” (●) may be specified in any sequence.



The above diagram shows that item2 and item3 may be specified in either order. Both of the following are valid:

```
STATEMENT item1 item2 item3 item4
STATEMENT item1 item3 item2 item4
```

Part 1. Concepts and Planning

Chapter 1. Overview of a federated system

The following sections provide an overview of a federated system.

Federated systems

A DB2® *federated system* is a special type of distributed database management system (DBMS). A federated system consists of a DB2 instance that operates as a federated server, a database that acts as the federated database, one or more data sources, and clients (users and applications) that access the database and data sources. With a federated system, you can send distributed requests to multiple data sources within a single SQL statement. For example, you can join data that is located in a DB2 Universal Database™ table, an Oracle table, and an XML tagged file in a single SQL statement. The following figure shows the components of a federated system and a sample of the data sources you can access.

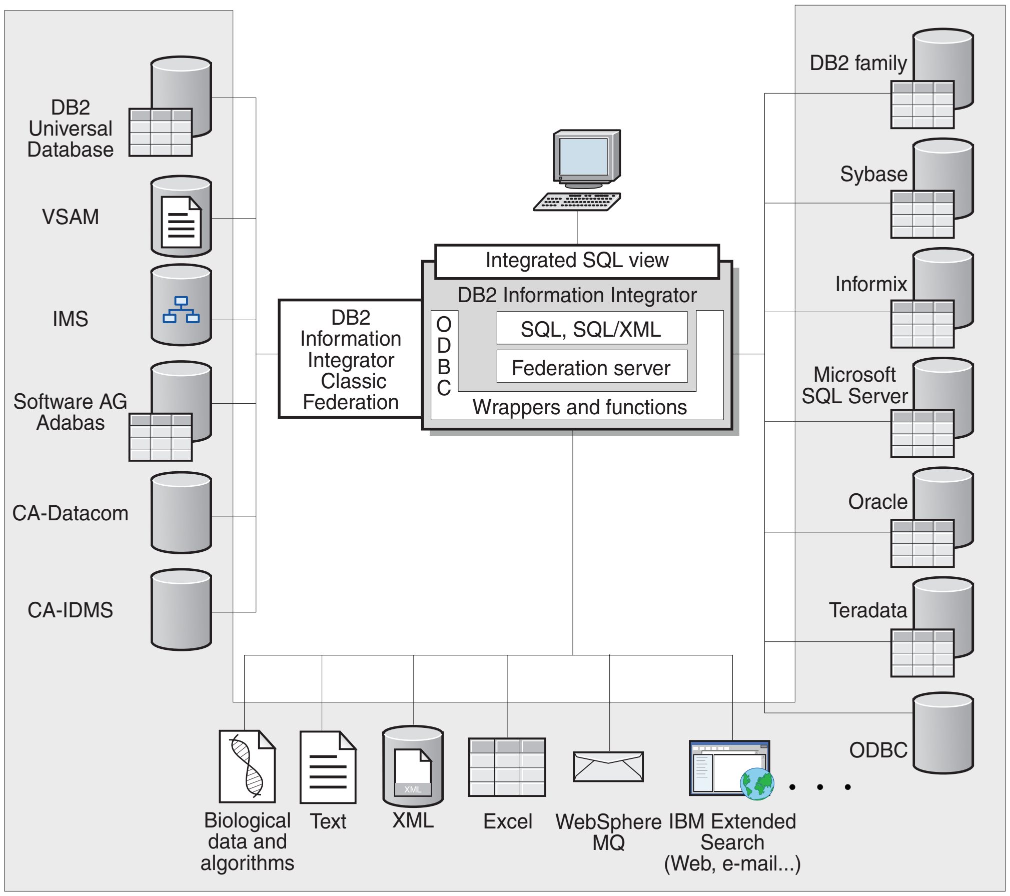


Figure 1. The components of a federated system

The power of a DB2 federated system is in its ability to:

- Join data from local tables and remote data sources, as if all the data is stored locally in the federated database
- Update data in relational data sources, as if the data is stored in the federated database

- Replicate data to and from relational data sources
- Take advantage of the data source processing strengths, by sending requests to the data sources for processing
- Compensate for SQL limitations at the data source by processing parts of a distributed request at the federated server

The federated server

The DB2® server in a federated system is referred to as the *federated server*. Any number of DB2 instances can be configured to function as federated servers. You can use existing DB2 instances as your federated servers, or you can create new ones specifically for the federated system.

The DB2 instance that manages the federated system is called a *server* because it responds to requests from end users and client applications. The federated server often sends parts of the requests it receives to the data sources for processing. A *pushdown* operation is an operation that is processed remotely. The DB2 instance that manages the federated system is referred to as the *federated server*, even though it acts as a client when it pushes down requests to the data sources.

Like any other application server, the federated server is a database manager instance. Application processes connect and submit requests to the database within the federated server. However, two main features distinguish it from other application servers:

- A federated server is configured to receive requests that might be partially or entirely intended for data sources. The federated server distributes these requests to the data sources.
- Like other application servers, a federated server uses DRDA® communication protocols (over TCP/IP) to communicate with DB2 family instances. However, unlike other application servers, a federated server uses the native client of the data source to access the data source. For example, a federated server uses the Sybase Open Client to access Sybase data sources and an Microsoft® SQL Server ODBC Driver to access Microsoft SQL Server data sources.

Related concepts:

- "What is a data source?" on page 4

What is a data source?

In a federated system, a *data source* can be a relational DBMS instance (such as Oracle or Sybase) or a nonrelational data source (such as BLAST search algorithm or an XML tagged file). Through some data sources you can access other data sources. For example, through the Extended Search data source you can access data sources such as Lotus® Notes databases, Microsoft® Access, Microsoft Index Server, Web search engines, and Lightweight Directory Access Protocol (LDAP) directories.

The method, or protocol, used to access a data source depends on the type of data source. For example, DRDA® is used to access DB2® for z/OS™ and OS/390® data sources and the Documentum Client API/Library is used to access Documentum data sources.

Data sources are semi-autonomous. For example, the federated server can send queries to Oracle data sources at the same time that Oracle applications can access these data sources. A DB2 federated system does not monopolize or restrict access to the other data sources, beyond integrity and locking constraints.

Related concepts:

- "The federated database" on page 7

Related reference:

- "Supported data sources" on page 5

Supported data sources

There are many data sources that you can access using a federated system. The following table lists the supported data sources:

Table 1. Supported data source versions and access methods.

Data source	Supported versions	Access method
DB2 Universal Database™ for Linux, UNIX, and Windows®	7.2, 8.1, 8.2	DRDA®
DB2 Universal Database for z/OS™ and OS/390®	6.1, 7.1 with the following APARs applied: • PQ62695 • PQ55393 • PQ56616 • PQ54605 • PQ46183 • PQ62139 8.1	DRDA
DB2 Universal Database for iSeries™	5.1 • with the following APARs applied: – SE06003 – SE06872 – II13348 • with the following PTFs applied: – SI05990 SI05991 5.2 with PTF SI0735 applied.	DRDA
DB2 Server for VM and VSE	7.1 (or later) with fixes for APARs for schema functions applied.	DRDA
Informix™	7.31, 8.32, 8.4, 9.3, 9.4	Informix Client SDK V2.7 (or later)

Table 1. Supported data source versions and access methods. (continued)

Data source	Supported versions	Access method
ODBC	3.x	ODBC driver for the data source, such as Redbrick ODBC Driver to access Redbrick.
OLE DB	2.7, 2.8	OLE DB 2.0 (or later)
Oracle	8.0.6, 8.1.6, 8.1.7, 9.0, 9.1, 9.2, 9i, 10g	Oracle net client or NET8 client software
Microsoft SQL Server	7.0, 2000 SP3 and later service packs on that release	On Windows, the Microsoft SQL Server Client ODBC 3.0 (or later) driver. On UNIX, the DataDirect Technologies (formerly MERANT) Connect ODBC 3.7 (or later) driver.
Sybase	11.9.2, 12.x	Sybase Open Client ctlib interface
Teradata	V2R3, V2R4, V2R5	Teradata Call-Level Interface, Version 2 (CLIv2) Release 04.06 (or later)
BLAST	2.2.3 and later 2.2 fixpacks supported	BLAST daemon (supplied with the wrapper)
BioRS	v5.0.14	None
Documentum	3.x, 4.x	Documentum Client library/APL3.1.7a (or later)
Entrez (PubMed and GenBank data sources)	1.0	None
HMMER	2.2g, 2.3	HMMER daemon (supplied with the wrapper)
IBM Lotus Extended Search	4.0.1, 4.0.2	Extended Search Client Library (supplied with the wrapper)
Microsoft Excel	97, 2000, 2002, 2003	Excel 97, 2000, 2002, or 2003 installed on the federated server
PeopleSoft	8.x	IBM WebSphere Business Integration Adapter for PeopleSoft v2.3.1, 2.4
SAP	3.x, 4.x	IBM WebSphere Business Integration Adapter for mySAP.com v2.3.1, 2.4
Siebel	7, 7.5, 2000	IBM WebSphere Business Integration Adapter for Siebel eBusiness Applications v2.3.1, 2.4
Table-structured files		None
User-defined functions for KEGG	Supported	
User-defined functions for Life Sciences	Supported	

Table 1. Supported data source versions and access methods. (continued)

Data source	Supported versions	Access method
Web services	SOAP 1.0., 1.1, WSDL 1.0, 1.1 specifications	HTTP
XML	1.0 specification	None

Related concepts:
- "What is a data source?" on page 4

The federated database

To end users and client applications, data sources appear as a single collective database in DB2®. Users and applications interface with the *federated database* managed by the federated server. The federated database contains a system catalog. The federated database system catalog contains entries that identify data sources and their characteristics. The federated server consults the information stored in the federated database system catalog and the data source wrapper to determine the best plan for processing SQL statements.

The federated system processes SQL statements as if the data sources were ordinary relational tables or views within the federated database. As a result:
- The federated system can join relational data with data in nonrelational formats. This is true even when the data sources use different SQL dialects, or do not support SQL at all.
- The characteristics of the federated database take precedence when there are differences between the characteristics of the federated database and the characteristics of the data sources:
  - Suppose the code page used by the federated server is different than the code page used by the data source. Character data from the data source is converted based on the code page used by the federated database, when that data is returned to a federated user.
  - Suppose the collating sequence used by the federated server is different than the collating sequence used by the data source. Any sort operations on character data are performed at the federated server instead of at the data source.

Related concepts:
- "The SQL Compiler" on page 8
- "The federated database system catalog" on page 7

The federated database system catalog

The federated database system catalog contains information about the objects in the federated database and information about objects at the data sources. The catalog in a federated database is called the *global catalog* because it contains information about the entire federated system. DB2® query optimizer uses the information in the global catalog and the data source wrapper to plan the best way to process SQL statements. The information stored in the global catalog includes remote and local information, such as column names, column data types, column default values, and index information.

Remote catalog information is the information or name used by the data source. *Local* catalog information is the information or name used by the federated database. For example, suppose a remote table includes a column with the name of *EMPNO*. The global catalog would store the remote column name as *EMPNO*. Unless you designate a different name, the local column name will be stored as *EMPNO*. You can change the local column name to *Employee_Number*. Users submitting queries which include this column will use *Employee_Number* in their queries instead of *EMPNO*. You use the ALTER NICKNAME statement to change the local name of the data source columns.

For relational data sources, the information stored in the global catalog includes both remote and local information.

For nonrelational data sources, the information stored in the global catalog varies from data source to data source.

To see the data source table information that is stored in the global catalog, query the SYSCAT.TABLES, SYSCAT.TABOPTIONS, SYSCAT.INDEXES, SYSCAT.COLUMNS, and SYSCAT.COLOPTIONS catalog views in the federated database.

The global catalog also includes other information about the data sources. For example, the global catalog includes information that the federated server uses to connect to the data source and map the federated user authorizations to the data source user authorizations. The global catalog contains attributes about the data source that you explicitly set, such as server options.

Related concepts:
- "The SQL Compiler" on page 8

Related reference:
- Appendix A, "Views in the global catalog table containing federated information," on page 563

The SQL Compiler

To obtain data from data sources, users and applications submit queries in DB2® SQL to the federated database. When a query is submitted, the DB2 SQL Compiler consults information in the global catalog and the data source wrapper to help it process the query. This includes information about connecting to the data source, server attributes, mappings, index information, and processing statistics.

Related concepts:
- "Wrappers and wrapper modules" on page 11
- "The query optimizer" on page 8

The query optimizer

As part of the SQL Compiler process, the *query optimizer* analyzes a query. The Compiler develops alternative strategies, called *access plans*, for processing the query. Access plans might call for the query to be:
- Processed by the data sources
- Processed by the federated server

- Processed partly by the data sources and partly by the federated server

DB2® UDB evaluates the access plans primarily on the basis of information about the data source capabilities and the data. The wrapper and the global catalog contain this information. DB2 UDB decomposes the query into segments that are called *query fragments*. Typically it is more efficient to *pushdown* a query fragment to a data source, if the data source can process the fragment. However, the query optimizer takes into account other factors such as:

- The amount of data that needs to be processed
- The processing speed of the data source
- The amount of data that the fragment will return
- The communication bandwidth
- Whether there is a usable materialized query table on the federated server that represents the same query result

The query optimizer generates local and remote access plans for processing a query fragment, based on resource cost. DB2 UDB then chooses the plan it believes will process the query with the least resource cost.

If any of the fragments are to be processed by data sources, DB2 UDB submits these fragments to the data sources. After the data sources process the fragments, the results are retrieved and returned to DB2 UDB. If DB2 UDB performed any part of the processing, it combines its results with the results retrieved from the data source. DB2 UDB then returns all results to the client.

Related concepts:

- "The SQL Compiler" on page 8
- "Compensation" on page 9
- "Tuning query processing" in the *Federated Systems Guide*

Compensation

The DB2® federated server does not push down a query fragment if the data source cannot process it, or if the federated server can process it faster than the data source can process it. For example, suppose that the SQL dialect of a data source does not support a CUBE grouping in the GROUP BY clause. A query that contains a CUBE grouping and references a table in that data source is submitted to the federated server. DB2 Information Integrator does not pushdown the CUBE grouping to the data source, but processes the CUBE itself. The ability by DB2 Information Integrator to process SQL that is not supported by a data source is called *compensation*.

The federated server compensates for lack of functionality at the data source in two ways:

- It can ask the data source to use one or more operations that are equivalent to the DB2 function stated in the query. Suppose a data source does not support the cotangent (COT(x)) function, but supports the tangent (TAN(x)) function. DB2 Information Integrator can ask the data source to perform the calculation (1/TAN(x)), which is equivalent to the cotangent (COT(x)) function.
- It can return the set of data to the federated server, and perform the function locally.

For relational data sources, each type of RDBMS supports a subset of the international SQL standard. In addition, some types of RDBMSs support SQL constructs that exceed this standard. An *SQL dialect*, is the totality of SQL that a type of RDBMS supports. If an SQL construct is found in the DB2 SQL dialect, but not in the relational data source dialect, the federated server can implement this construct on behalf of the data source.

DB2 Information Integrator can compensate for differences in SQL dialects. An example of this ability is the common-table-expression clause. DB2 SQL includes the clause common-table-expression. In this clause, a name can be specified by which all FROM clauses in a fullselect can reference a result set. The federated server will process a common-table-expression for a data source, even though the SQL dialect used by the data source does not include common-table-expression.

With compensation, the federated server can support the full DB2 SQL dialect for queries of data sources. Even data sources with weak SQL support or no SQL support will benefit from compensation. You must use the DB2 SQL dialect with a federated system, except in a pass-through session.

Related concepts:

- "Pass-through sessions" on page 10

Pass-through sessions

You can submit SQL statements directly to data sources by using a special mode called *pass-through*. You submit SQL statements in the SQL dialect used by the data source. Use a pass-through session when you want to perform an operation that is not possible with the DB2® SQL/API. For example, use a pass-through session to create a procedure, create an index, or perform queries in the native dialect of the data source.

Currently, the data sources that support pass-through, support pass-through using SQL. In the future, it is possible that data sources will support pass-though using a data source language other than SQL.

Similarly, you can use a pass-through session to perform actions that are not supported by SQL, such as certain administrative tasks. However, you cannot use a pass-through session to perform all administrative tasks. For example, you can create or drop tables at the data source, but you cannot start or stop the remote database.

You can use both static and dynamic SQL in a pass-through session.

The federated server provides the following SQL statements to manage pass-through sessions:

SET PASSTHRU
: Opens a pass-through session. When you issue another SET PASSTHRU statement to start a new pass-through session, the current pass-through session is terminated.

SET PASSTHRU RESET
: Terminates the current pass-through session.

GRANT (Server Privileges)
: Grants a user, group, list of authorization IDs, or PUBLIC the authority to initiate pass-through sessions to a specific data source.

REVOKE (Server Privileges)
Revokes the authority to initiate pass-through sessions.

The following restrictions apply to pass-through sessions:

- You must use the SQL dialect or language commands of the data source — you cannot use the DB2 SQL dialect. As a result, you do not query a nickname, but the data source objects directly.
- When performing UPDATE or DELETE operations in a pass-through session, you cannot use the WHERE CURRENT OF CURSOR condition.
- LOBs are not supported in pass-through sessions.

Related concepts:

- "Wrappers and wrapper modules" on page 11
- "Querying data sources directly with pass-through" in the *Federated Systems Guide*

Wrappers and wrapper modules

Wrappers are mechanisms by which the federated server interacts with data sources. The federated server uses routines stored in a library called a *wrapper module* to implement a wrapper. These routines allow the federated server to perform operations such as connecting to a data source and retrieving data from it iteratively. Typically, the DB2® federated instance owner uses the CREATE WRAPPER statement to register a wrapper in the federated database. You can register a wrapper as fenced or trusted using the DB2_FENCED wrapper option.

You create one wrapper for each type of data source that you want to access. For example, suppose that you want to access three DB2 for z/OS™ database tables, one DB2 for iSeries™ table, two Informix® tables, and one Informix view. You need to create one wrapper for the DB2 data source objects and one wrapper for the Informix data source objects. Once these wrappers are registered in the federated database, you can use these wrappers to access other objects from those data sources. For example, you can use the DRDA® wrapper with all DB2 family data source objects—DB2 for Linux, UNIX®, and Windows®, DB2 for z/OS and OS/390®, DB2 for iSeries, and DB2 Server for VM and VSE.

You use the server definitions and nicknames to identify the specifics (name, location, and so forth) of each data source object.

A wrapper performs many tasks. Some of these tasks are:

- It connects to the data source. The wrapper uses the standard connection API of the data source.
- It submits queries to the data source.
  - For data sources that support SQL, the query is submitted in SQL.
  - For data sources that do not support SQL, the query is translated into the native query language of the source or into a series of source API calls.
- It receives results sets from the data source. The wrapper uses the data source standard APIs for receiving results set.
- It responds to federated server queries about the default data type mappings for a data source. The wrapper contains the default type mappings that are used when nicknames are created for a data source object. For relational wrappers, data type mappings that you create override the default data type mappings. User-defined data type mappings are stored in the global catalog.

- It responds to federated server queries about the default function mappings for a data source. The wrapper contains information that the federated server needs to determine if DB2 functions are mapped to functions of the data source, and how the functions are mapped. This information is used by the SQL Compiler to determine if the data source is able to perform the query operations. For relational wrappers, function mappings that you create override the default function type mappings. User-defined function mappings are stored in the global catalog.

Wrapper options are used to configure the wrapper or to define how DB2 Information Integrator uses the wrapper.

Related tasks:

- "Trusted and fenced mode process environments" in the *IBM DB2 Information Integrator Wrapper Developer's Guide*
- "Registering wrappers for a data source" on page 61

Related reference:

- "CREATE WRAPPER statement" in the *SQL Reference, Volume 2*
- Appendix B, "Wrapper options for federated systems," on page 567

Default wrapper names

There are wrappers for each supported data source. Some wrappers have default wrapper names. When you use the default name to create the wrapper, the federated server automatically picks up the data source library associated with the wrapper.

Table 2. Default wrapper names for each data source.

Data source	Default wrapper names
DB2 Universal Database™ for Linux, UNIX and Windows®	DRDA
DB2 Universal Database for z/OS and OS/390®	DRDA
DB2 Universal Database for iSeries	DRDA
DB2 Server for VM and VSE	DRDA
Informix	INFORMIX
Microsoft® SQL Server	MSSQLODBC3
ODBC	ODBC
OLE DB	OLEDB
Oracle	NET8
Sybase	CTLIB
Teradata	TERADATA
BLAST	None
BioRS	None
Documentum	None
Entrez	None
Extended Search	None

Table 2. Default wrapper names for each data source. (continued)

Data source	Default wrapper names
HMMER	None
Microsoft Excel	None
Table-structured files	None
Web Services	None
WebSphere Business Integration	None
XML	None

Related concepts:

- "Wrappers and wrapper modules" on page 11

Server definitions and server options

After wrappers are created for the data sources, the federated instance owner defines the data sources to the federated database. The instance owner supplies a name to identify the data source, and other information that pertains to the data source. This information includes:

- The type and version of the data source
- The database name for the data source (RDBMS only)
- Metadata that is specific to the data source

For example, a DB2® family data source can have multiple databases. The definition must specify which database the federated server can connect to. In contrast, an Oracle data source has one database, and the federated server can connect to the database without knowing its name. The database name is not included in the federated server definition of an Oracle data source.

The name and other information that the instance owner supplies to the federated server are collectively called a *server definition*. Data sources answer requests for data and are servers in their own right.

The CREATE SERVER and ALTER SERVER statements are used to create and modify a server definition.

Some of the information within a server definition is stored as *server options*. When you create server definitions, it is important to understand the options that you can specify about the server. Some server options configure the wrapper and some affect the way DB2 Information Integrator uses the wrapper.

Server options can be set to persist over successive connections to the data source, or set for the duration of a single connection.

Related tasks:

- "Registering server definitions for a data source" on page 61

Related reference:

- "ALTER SERVER statement" in the *SQL Reference, Volume 2*
- "CREATE SERVER statement" in the *SQL Reference, Volume 2*
- Appendix D, "Server options for federated systems," on page 575

User mappings

When a federated server needs to pushdown a request to a data source, the server must first establish a connection to the data source.

For most data sources, the federated server does this by using a valid user ID and password to that data source. When a user ID and password is required to connect to a data source, you can define an association between the federated server authorization ID and the data source user ID and password. This association can be created for each user ID that will be using the federated system to send distributed requests. This association is called a *user mapping*.

In some cases, you do not need to create a user mapping if the user ID and password you use to connect to the federated database are the same as those you use to access the remote data source.

Related tasks:

- "Registering user mappings for a data source" on page 63

Related reference:

- "ALTER USER MAPPING statement" in the *SQL Reference, Volume 2*
- "CREATE USER MAPPING statement" in the *SQL Reference, Volume 2*
- Appendix E, "User mapping options for federated systems," on page 591

Nicknames and data source objects

After you create the server definitions and user mappings, the federated instance owner creates the nicknames. A *nickname* is an identifier that is used to reference the object located at the data sources that you want to access. The objects that nicknames identify are referred to as *data source objects*.

Nicknames are not alternative names for data source objects in the same way that aliases are alternative names. They are pointers by which the federated server references these objects. Nicknames are typically defined with the CREATE NICKNAME statement along with specific nickname column options and nickname options.

When an end user or a client application submits a distributed request to the federated server, the request does not need to specify the data sources. Instead, the request references the data source objects by their nicknames. The nicknames are mapped to specific objects at the data source. These mappings eliminate the need to qualify the nicknames by data source names. The location of the data source objects is transparent to the end user or the client application.

Suppose that you define the nickname *DEPT* to represent an Informix® database table called *NFX1.PERSON*. The statement SELECT * FROM *DEPT* is allowed from the federated server. However, the statement SELECT * FROM *NFX1.PERSON* is not allowed from the federated server (except in a pass-through session) unless there is a local table on the federated server named *NFX1.PERSON*.

When you create a nickname for a data source object, metadata about the object is added to the global catalog. The query optimizer uses this metadata, and the information in the wrapper, to facilitate access to the data source object. For example, if the nickname is for a table that has an index, the global catalog contains information about the index. The wrapper contains the mappings between the DB2® data types and the data source data types.

Currently, you cannot execute some DB2 UDB utility operations on nicknames.

You cannot use the Cross Loader utility to cross load into a nickname.

Related concepts:
- "Nickname column options" on page 16

Related tasks:
- "Registering nicknames for a data source" on page 63

Related reference:
- "ALTER NICKNAME statement" in the *SQL Reference, Volume 2*
- "CREATE NICKNAME statement" in the *SQL Reference, Volume 2*
- Appendix G, "Nickname column options for federated systems," on page 603
- Appendix F, "Nickname options for federated systems," on page 593
- "Valid data source objects" on page 15

Valid data source objects

Nicknames identify objects at the data source that you want to access. The following table lists the types of objects that you can create a nickname for in a federated system.

Table 3. Valid data source objects

Data source	Valid objects
DB2 for Linux, UNIX, and Windows	Nicknames, materialized query tables, tables, views
DB2 for z/OS and OS/390	Tables, views
DB2 for iSeries	Tables, views
DB2 for VM and VSE	Tables, views
Informix	Tables, views, synonyms
Microsoft SQL Server	Tables, views
ODBC	Tables, views
Oracle	Tables, views, synonyms
Sybase	Tables, views
Teradata	Tables, views
BLAST	FASTA files indexed for BLAST search algorithms
BioRS	BioRS databanks
Documentum	Objects and registered tables in a Documentum Docbase
Entrez	Entrez databases
Extended Search	Files from data sources such as Lotus Notes databases, Microsoft Access, Microsoft Index Server, Web search engines, and LDAP directories.

Table 3. Valid data source objects (continued)

Data source	Valid objects
HMMER	HMM database files (libraries of Hierarchical Markov Models, such as PFAM), that can be searched by HMMER’s hmmpfam or hmmsearch programs.
Microsoft Excel	.xls files (only the first sheet in the workbook is accessed)
Table-structured files	Text files that meet a specific format.
Websphere Business Integration adapters	Websphere Business Integration business objects that map to BAPIs in SAP, business components in Siebel, and component interfaces in PeopleSoft
Web Services	Operations in a Web services description language file
XML-tagged files	Sets of items in an XML document

Related concepts:
- “Nicknames and data source objects” on page 14
- “Nickname column options” on page 16

Nickname column options

You can supply the global catalog with additional metadata information about the nicknamed object. This metadata describes values in certain columns of the data source object. You assign this metadata to parameters that are called *nickname column options*. The nickname column options tell the wrapper to handle the data in a column differently than it normally would handle it. The SQL Complier and query optimizer use the metadata to develop better plans for accessing the data.

Nickname column options are used to provide other information to the wrapper as well. For example for XML data sources, a nickname column option is used to tell the wrapper the XPath expression to use when the wrapper parses the column out of the XML document.

With federation, the DB2® server treats the data source object that a nickname references as if it is a local DB2 table. As a result, you can set nickname column options for any data source object that you create a nickname for. Some nickname column options are designed for specific types of data sources and can be applied only to those data sources.

Suppose that a data source has a collating sequence that differs from the federated database collating sequence. The federated server typically would not sort any columns containing character data at the data source. It would return the data to the federated database and perform the sort locally. However, suppose that the column is a character data type (CHAR or VARCHAR) and contains only numeric characters (’0’,’1’,...,’9’). You can indicate this by assigning a value of ’Y’ to the NUMERIC_STRING nickname column option. This gives the DB2 query optimizer the option of performing the sort at the data source. If the sort is performed remotely, you can avoid the overhead of porting the data to the federated server and performing the sort locally.

You can define nickname column options for relational nicknames using the ALTER NICKNAME statement. You can define nickname column options for nonrelational nicknames using the CREATE NICKNAME and ALTER NICKNAME statements.

Related concepts:
- "Data type mappings" on page 17

Related tasks:
- "Working with nicknames" in the *Federated Systems Guide*

Related reference:
- Appendix G, "Nickname column options for federated systems," on page 603

Data type mappings

The data types at the data source must map to corresponding DB2® data types so that the federated server can retrieve data from data sources. Some examples of default data type mappings are:
- The Oracle type FLOAT maps to the DB2 type DOUBLE
- The Oracle type DATE maps to the DB2 type TIMESTAMP
- The DB2 for z/OS™ type DATE maps to the DB2 type DATE

For most data sources, the default type mappings are in the wrappers. The default type mappings for DB2 data sources are in the DRDA® wrapper. The default type mappings for Informix® are in the INFORMIX wrapper, and so forth.

For some nonrelational data sources, you must specify data type information in the CREATE NICKNAME statement. The corresponding DB2 for Linux, UNIX®, and Windows® data types must be specified for each column of the data source object when the nickname is created. Each column must be mapped to a particular field or column in the data source object.

For relational data sources, you can override the default data type mappings. For example, by default the Informix INTEGER data type maps to the DB2 INTEGER data type. You could override the default mappings and map Informix's INTEGER data type to DB2 DECIMAL(10,0) data type.

You should create new type mappings or modify the default type mappings before you create nicknames. Otherwise nicknames created before the type mapping changes will not reflect the new mappings.

Related concepts:
- "Data type mappings in a federated system" in the *Federated Systems Guide*

Function mappings

For the federated server to recognize a data source function, the function must be mapped to an existing counterpart function in DB2® for Linux, UNIX® and Windows®. DB2 Information Integrator supplies default mappings between existing built-in data source functions and built-in DB2 counterpart functions. For most data sources, the default function mappings are in the wrappers. The default function mappings to DB2 for z/OS™ and OS/390® functions are in the DRDA® wrapper. The default function mappings to Sybase functions are in the CTLIB wrapper, and so forth.

For relational data sources, you can create a function mapping when you want to use a data source function that the federated server does not recognize. The mapping that you create is between the data source function and a DB2 counterpart function at the federated database. Function mappings are typically used when a new built-in function or a new user-defined function become available at the data source. Function mappings are also used when a DB2 counterpart function does not exist. In this case, you must also create a function template.

Related concepts:

- "Function mappings in a federated system" in the *Federated Systems Guide*
- "Index specifications" on page 18

Index specifications

When you create a nickname for a data source table, information about any indexes that the data source table has is added to the global catalog. The query optimizer uses this information to expedite the processing of distributed requests. The catalog information about a data source index is a set of metadata, and is called an *index specification*. A federated server does not create an index specification when you create a nickname for:

- A table that has no indexes
- A view, which typically does not have any index information stored in the remote catalog
- A data source object that does not have a remote catalog from which the federated server can obtain the index information

Suppose that a table acquires a new index, in addition to the ones it had when the nickname was created. Because index information is supplied to the global catalog at the time the nickname is created, the federated server is unaware of the new index. Similarly, when a nickname is created for a view, the federated server is unaware of the underlying table (and its indexes) from which the view was generated. In these circumstances, you can supply the necessary index information to the global catalog. You can create an index specification for tables that have no indexes. The index specification tells the query optimizer which column or columns in the table to search on to find data quickly.

Related concepts:

- "Index specifications in a federated system" in the *Federated Systems Guide*

How you interact with a federated system

Because the federated database is a DB2® Universal Database, you can interact with a federated system using any one of these methods:

- The DB2 command line processor (CLP)
- The DB2 Command Center GUI
- The DB2 Control Center GUI
- Application programs

- DB2 family tools
- Web services providers

The steps in the federated documentation provide the commands and SQL statements that can be entered in the DB2 command line processor or the DB2 Command Center GUI. The documentation indicates when tasks can be performed through the DB2 Control Center GUI. Since the DB2 Control Center GUI is intuitive, the steps to perform these tasks through the DB2 Control Center are not included in this documentation.

DB2 command line processor (CLP)

You can perform most of the tasks necessary to setup, configure, tune, and maintain the federated system through the DB2 command line processor. In some cases you must use either the DB2 command line processor or the DB2 Command Center. For example:

- Create, alter, or drop user-defined data type mappings
- Create, alter, or drop user-defined function mappings

DB2 Command Center

Through the DB2 Command Center, you can create and run distributed requests without having to manually type out lengthy SQL statements. Use the DB2 Command Center when you are tuning the performance of the federated system. The DB2 Command Center is a convenient way to use the DB2 Explain functionality to look at the access plans for distributed requests. The DB2 Command Center can also be used to work with the SQL Assistant tool.

DB2 Control Center

The DB2 Control Center GUI allows you to perform most of the tasks necessary to setup, configure, and modify the federated system. The DB2 Control Center uses panels—dialog boxes and wizards—to guide you through a task. These panels contain interactive help when your mouse hovers over a control such as a list box or command button. Additionally, each panel has a help button that provides information about the panel task, and links to related concepts and reference information.

You can either use a wizard to create the federated objects, or you can create each object individually.

Use the DB2 Control Center to configure access to Web services, WebSphere® Business Integration, and XML data sources. The features built into the DB2 Control Center simplify the steps that are required for you to configure the federated server to access these data sources.

The DB2 Control Center GUI is the easiest way to perform the essential data source configuration tasks:

- Create the wrappers and set the wrapper options
- Specify the environment variables for your data source
- Create the server definitions and set the server options
- Create the user mappings and set the user options
- Create the nicknames and set the nickname options or column options

After you configure the federated server to access your data sources, you can use the DB2 Control Center to:

- Modify the data source configuration
- Monitor the status of the nicknames and servers
- Maintain current statistics for your nicknames
- Create and modify cache tables
- Specify informational constraints on nicknames
- Create remote tables through DB2 Information Integrator using transparent DDL

Application programs

Applications do not require any special coding to work with federated data. Applications access the system just like any other DB2 client application. Applications interface with the federated database that is within the federated server. To obtain data from data sources, applications submit queries in DB2 SQL to the federated database. DB2 Information Integrator then distributes the queries to the appropriate data sources, collects the requested data, and returns this data to the applications. However, since DB2 Information Integrator interacts with the data sources through nicknames, you need to be aware of:

- The SQL restrictions you have when working with nicknames
- How to perform operations on nicknamed objects

DB2 family tools

You can also interact with a federated database using host and midrange tools such as:

- DB2 SPUFI on DB2 for z/OS™ and OS/390®
- Interactive SQL (STRSQL) on DB2 for iSeries™

Web services providers

You can also interact with a federated database through web services providers using the Web Services wrapper.

Related concepts:

- "The Web services wrapper and the Web services description language document" on page 381

Related tasks:

- "Adding Web services data sources to a federated server" on page 387

Chapter 2. Planning for federated data source configuration

The following sections provide information you can use to help you plan your federated system.

Federated object naming rules

As with other DB2 objects, there are rules for naming federated database objects.

Federated database objects include:
- Function mappings
- Index specifications
- Nicknames
- Servers
- Type mappings
- User mappings
- Wrappers

Federated object names must begin with one of the following:
- A letter, including a valid accented letter (such as Ö)
- A multibyte character, except a multibyte space (for multibyte environments)

Federated object names cannot begin with a number or with the underscore character.

Federated object names can also include the following characters:
- A through Z
- 0 through 9
- @, #, $, and _ (underscore)

Federated object names cannot exceed 128 bytes.

Options (such as server options and nickname options) and option settings are limited to 255 bytes.

Names without quotation marks are converted to uppercase.

Related concepts:
- "Naming rules in an NLS environment" in the *Administration Guide: Implementation*
- "Naming rules in a Unicode environment" in the *Administration Guide: Implementation*

Related reference:
- "Checklist for planning your federated system configuration" on page 31
- "Preserving case-sensitive values in a federated system" on page 22

Preserving case-sensitive values in a federated system

In a federated system you occasionally need to specify values that are case-sensitive at the data source, such as user IDs and passwords. To ensure that the case is correct when these values are passed to the data source, follow these guidelines:

- Specify the values in the required case and enclose them in the proper quotation marks. Double quotation marks are optional for object names, such as the name of a wrapper or nickname. Single quotation marks are required for option values, such as the REMOTE_AUTHID and REMOTE_PASSWORD user mapping options.
- For user IDs and passwords, you can set the FOLD_ID and FOLD_PW server options to automatically convert the values to the proper case. With this option, you do not have to remember the required case for each data source. You can type the values in any case and they will be converted automatically.

From a UNIX operating system command prompt:

If you enclose a case-sensitive value in quotation marks at the federated server operating system command prompt, you must ensure that the quotation marks are parsed correctly:

- If the SQL statement contains double quotation marks, but does not contain single quotation marks, you enclose the entire statement in single quotation marks. For example, if you want to issue this SQL statement:

  ```
  CREATE NICKNAME my_nick FOR my_server."owner"."my_table"
  ```

  You enter the following text at the UNIX command prompt

  ```
  DB2 'CREATE NICKNAME my_nick FOR my_server."owner"."my_table"'
  ```

- If the SQL statement contains single quotation marks, but does not contain double quotation marks, you enclose the entire statement in double quotation marks. For example, if you want to issue this SQL statement:

  ```
  CREATE USER MAPPING FOR USER SERVER my_server
      OPTIONS(REMOTE_AUTHID 'my_id', REMOTE_PASSWORD 'my_password')
  ```

  You enter the following text at the UNIX command prompt

  ```
  DB2 "CREATE USER MAPPING FOR USER SERVER my_server
      OPTIONS(REMOTE_AUTHID 'my_id', REMOTE_PASSWORD 'my_password')"
  ```

- If the SQL statement contains both single and double quotation marks:
  - Enclose the entire statement in double quotation marks
  - Precede the values that require double quotation marks with a backslash

  For example, to issue this SQL statement:

  ```
  CREATE USER MAPPING FOR "local_id" SERVER my_server
      OPTIONS(REMOTE_AUTHID 'my_id', REMOTE_PASSWORD  'my_password')
  ```

  You enter the following text at the UNIX command prompt:

  ```
  DB2 "CREATE USER MAPPING FOR \"local_id\" SERVER my_server
      OPTIONS(REMOTE_AUTHID 'my_id',  REMOTE_PASSWORD 'my_password')"
  ```

The above examples assume you are entering SQL statements from the UNIX command prompt and are passing the statement to the DB2 command, without the -f option. If you enter the SQL statements from a file using the DB2 command with the -f option, then you should enter the statements as show in the first occurrence of each example.

From a Windows operating system command prompt:

To preserve case-sensitive values on Windows, precede each double quotation mark with a backward slash. For example, you want to create the nickname NICK1 for the Microsoft SQL Server table weekly_salary. The table resides in the NORBASE database. The local schema is my_schema.

At the Windows command prompt on your federated server, you type:

```
DB2 CREATE NICKNAME nick1
    FOR norbase.\"my_schema\".\"weekly_salary\"
```

From the DB2 CLP or from an application program:

When you specify a value from the DB2 command line prompt (CLP) or in an application program, you can preserve case-sensitive values by enclosing the values in the proper quotation marks.

For example, you want to create a user mapping for the user ID local_id. The remote user ID my_id and the remote password is my_password. You want all three of these values to be preserved in lowercase. At the DB2 command prompt you type:

```
CREATE USER MAPPING FOR "local_id" SERVER my_server
    OPTIONS(REMOTE_AUTHID 'my_id', REMOTE_PASSWORD 'my_password')
```

Related reference:

- Appendix D, "Server options for federated systems," on page 575
- Appendix E, "User mapping options for federated systems," on page 591
- "Checklist for planning your federated system configuration" on page 31

Update data source statistics

If you plan to access a relational data source, you should update the statistics at the remote data source before you configure the federated server to access the data source. By ensuring that the remote data source has current statistics, you can improve query performance.

The federated server relies on the data source statistics that are stored in the federated database to optimize query processing. These statistics are gathered when you create a nickname for a data source object. The federated database verifies the presence of the object at the data source, and then attempts to gather existing statistical data for the data source. Information useful to the query optimizer is read from the data source catalogs and added to the system catalog in the federated database. Because some or all of the catalog information from the data source might be used by the query optimizer, it is recommended that you update statistics at the data source before you create a nickname. Use the command at the data source that is equivalent to the DB2 RUNSTATS command to update the data source statistics.

The federated database retrieves that statistical information for a data source object when you create a nickname for the object. If the data source updates its catalog statistics for an object after your create the nickname, the changes in the statistical information are not propagated to the system catalog in the federated database. To make sure that the system catalog in the federated database reflects the current statistics for the remote data source object, you must request that the federated server update the statistics.

Action: Identify the data source objects that you want to access. These are objects that you will create nicknames for. Determine which of the data sources that these objects are part of allow you to update statistics. List those data sources in the data source statistics table in the planning checklist.

Related concepts:

- "Nickname statistics update facility - overview" in the *Federated Systems Guide*

Related reference:

- "Checklist for planning your federated system configuration" on page 31

Choose the correct wrapper

For most data sources there is only one wrapper that you can use to access the data source. However, for some data sources you have a choice as to which wrapper you use to access the data in the data source.

You can access data sources that support the ODBC API either by using the wrappers that are designed for those data sources or by using the ODBC wrapper. Examples of these data sources include Oracle, Microsoft Excel, Microsoft SQL Server. Typically, query performance is better when you use the wrappers that are specifically designed for these data sources.

Use the ODBC wrapper to access any data source that has an ODBC driver but is not supported by specific data source wrappers that are included with DB2 Information Integrator. For example, use the ODBC wrapper to access RedBrick data sources.

DB2 for Linux, UNIX, and Windows data sources
Do not use the ODBC wrapper to access DB2 Universal Database for Linux, UNIX, and Windows data sources. Using the ODBC wrapper to access DB2 Universal Database for Linux, UNIX, and Windows data sources is not supported. Use the DRDA wrapper to access DB2 Universal Database for Linux, UNIX, and Windows data sources.

Excel data sources
Depending on your needs, you can use the ODBC wrapper to access Excel data instead of using the Excel wrapper.

Informix data sources
Do not use the ODBC wrapper to access Informix data sources. Using the ODBC wrapper to access Informix data sources is not supported. To access Informix data sources, use the Informix wrapper.

Action: Identify the wrappers that you will create for your federated system in the wrapper table in the planning checklist.

Related concepts:

- "Methods of accessing Excel data" on page 25

Related tasks:

- "Adding Excel data sources to a federated server" on page 218
- "Adding ODBC data sources to a federated server" on page 305

Related reference:

- “Checklist for planning your federated system configuration” on page 31

Methods of accessing Excel data

You can access data in Microsoft® Excel worksheets by using either the Excel wrapper or the ODBC wrapper.

To query Excel data, both wrappers require a DB2® federated server that can open and read the worksheets in the Excel workbook. Therefore, the Excel workbook must be on the same computer as the federated server or on a network accessible drive.

If you use the Excel wrapper, the Excel application must be installed on the federated server.

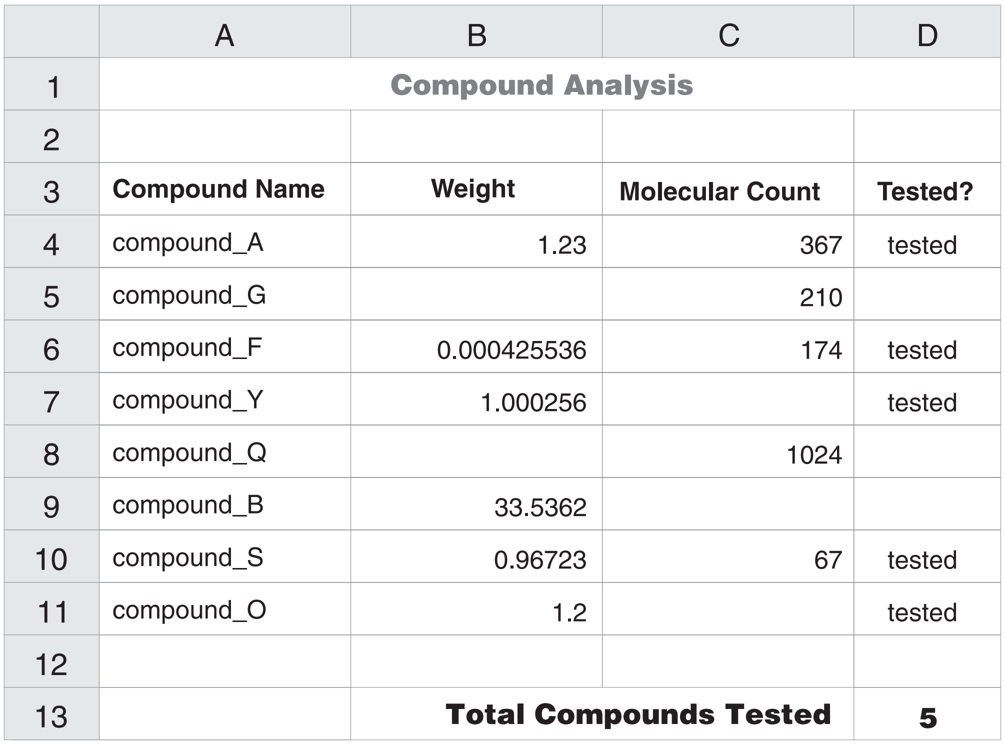
If you use the ODBC wrapper, the Excel ODBC driver must be on the federated server. This driver is installed automatically with Microsoft Windows®. The Excel application does not need to be installed on the federated server.

Each wrapper imposes some requirements on the location and layout of the data in the Excel workbooks. With the Excel wrapper, only the data in the first worksheet in the workbook can be accessed. With the ODBC wrapper, you can access data from any worksheet in the workbook.

The following examples show the worksheet layout requirements for these two wrappers.

Example of a worksheet that contains rows of labels and a formula:

This example shows a worksheet that contains several rows of labels at the top of the worksheet, blank rows, and a formula in row 13. To access the data in the worksheet, you must identify the range of cells that you want to access.



	A	B	C	D
1	**Compound Analysis**			
2				
3	**Compound Name**	**Weight**	**Molecular Count**	**Tested?**
4	compound_A	1.23	367	tested
5	compound_G		210	
6	compound_F	0.000425536	174	tested
7	compound_Y	1.000256		tested
8	compound_Q		1024	
9	compound_B	33.5362		
10	compound_S	0.96723	67	tested
11	compound_O	1.2		tested
12				
13			**Total Compounds Tested**	**5**

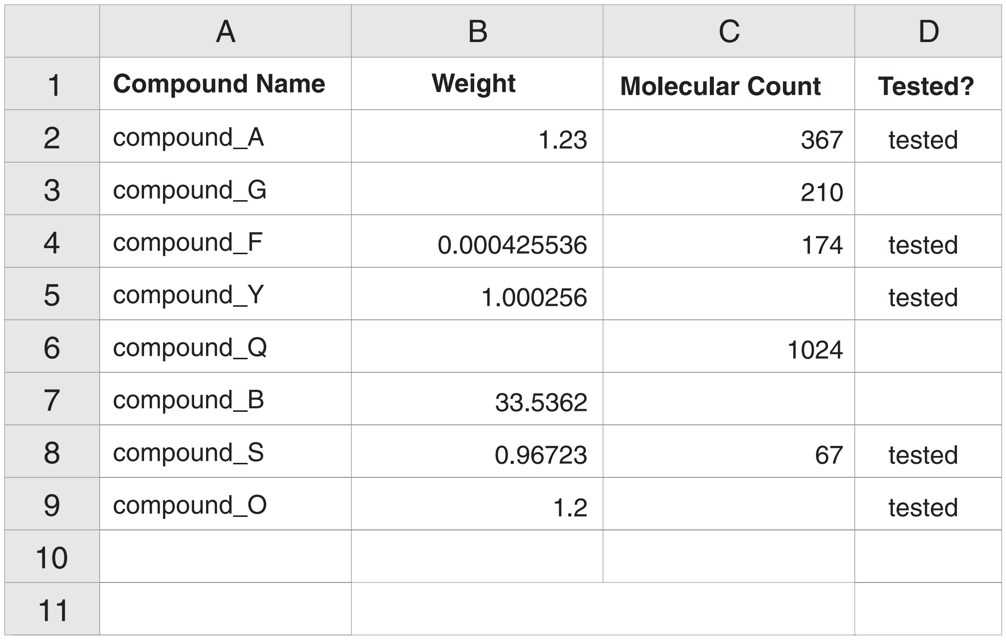
Figure 2. A worksheet that contains several rows of labels and a formula

If you use the Excel wrapper
You specify the range of cells in the CREATE NICKNAME statement by using the RANGE option. Include only the data in the range that you specify. Do not include any column labels in the range. Cells that contain formulas, such as SUM, return the result of the formula and not the formula. Unless you want the formula results returned, do not include the cells that contain formulas in the range. In this example, the range of cells that you include in the RANGE option is A4:D11.

If you use the ODBC wrapper
You must create a name for the range of cells to explicitly designate the location of the data within the worksheet. Excel refers to this range of cells as a *named range*. The Excel ODBC driver recognizes only one row of labels, the first row in the range. No blank rows are allowed between the labels and the data. The named range must include only one row of column labels. You specify the named range in the CREATE NICKNAME statement. You must include one row of column labels in the range that you name. If you do not include one row of column labels in the named range, the first row of data is treated as column labels. Cells that contain formulas, such as SUM, return the result of the formula and not the formula. Unless you want the formula results returned, do not include the cells that contain formulas in the range. In this example, the range of cells that you name is A3:D11.

Example of a worksheet that contains one row of labels:

This example shows a worksheet that contains only one row of column labels at the top of the worksheet. The layout does not include extra rows with labels, blank rows, or cells with formulas.



	A	B	C	D
1	**Compound Name**	**Weight**	**Molecular Count**	**Tested?**
2	compound_A	1.23	367	tested
3	compound_G		210	
4	compound_F	0.000425536	174	tested
5	compound_Y	1.000256		tested
6	compound_Q		1024	
7	compound_B	33.5362		
8	compound_S	0.96723	67	tested
9	compound_O	1.2		tested
10				
11				

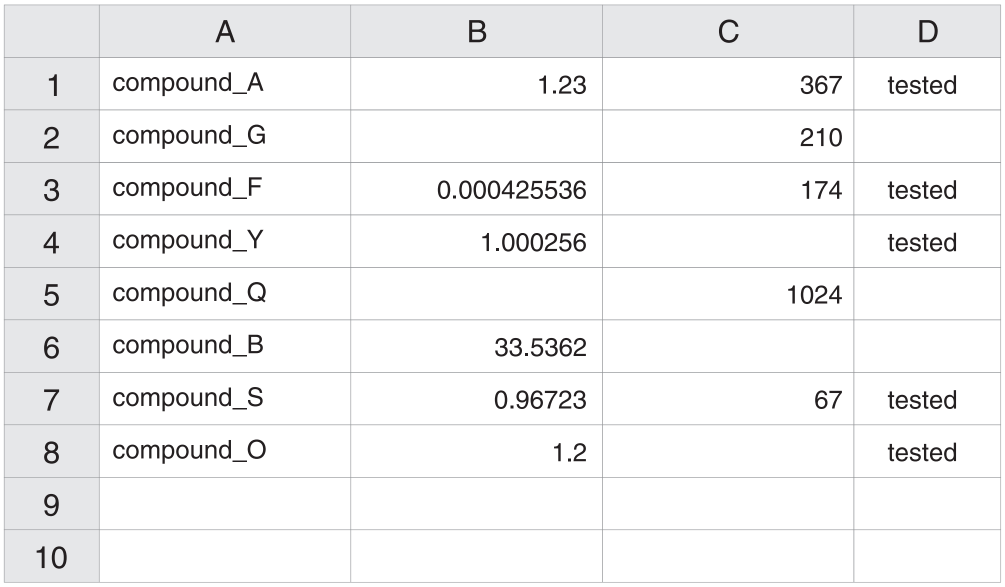
Figure 3. A worksheet that contains one row of column labels in row 1

If you use the Excel wrapper
: You must specify the range of cells in the CREATE NICKNAME statement by using the RANGE option. The range cannot include the column labels in row 1. The range of cells that you would specify is A2:D9.

If you use the ODBC wrapper
: You can access this data without creating a named range. You specify the worksheet name in the CREATE NICKNAME statement. The wrapper reads the first nonblank row as labels and uses the information as column names for the nickname. Subsequent rows are read as data.

Example of a worksheet that contains only data:

This example shows a worksheet that contains only data. There are no rows of column labels, no blank rows, and no cells with formulas.



	A	B	C	D
1	compound_A	1.23	367	tested
2	compound_G		210	
3	compound_F	0.000425536	174	tested
4	compound_Y	1.000256		tested
5	compound_Q		1024	
6	compound_B	33.5362		
7	compound_S	0.96723	67	tested
8	compound_O	1.2		tested
9				
10				

Figure 4. A worksheet that contains only data

If you use the Excel wrapper
If the data is in the first worksheet in the workbook, the wrapper will access the data without using the RANGE option. If the data is in another worksheet in the workbook, you must specify the RANGE option in the CREATE NICKNAME statement.

If you use the ODBC wrapper
When you use the ODBC wrapper to access Excel data, the wrapper is limited by what the Excel ODBC driver supports. The Excel ODBC driver requires a specific format for the worksheet. The driver assumes that the first nonblank row contains the column labels. If the first nonblank row contains data, the data in that row is treated as the column labels for the remaining data. If the worksheet does not contain a row of column labels, the first row is used as the labels and not as data. In effect, you lose the first row of data. You can overcome this requirement by modifying your worksheet. Insert a new row before the data and add labels for each column of data, so that it looks like the example that contains one row of labels.

Related tasks:
- "Adding Excel data sources to a federated server" on page 218
- "Accessing Excel data using the ODBC wrapper" on page 316

Plan the user mappings

When a federated server needs to pushdown a request to a data source, the server must first establish a connection to the data source. For some data sources, the federated server establishes a connection by using a valid user ID and password to that data source. For these data sources, you must define an association between the federated server user ID and password and the data source user ID and password. This association must be created for each user ID that will be using the federated system to send distributed requests. This association is called a *user mapping*.

You can use the DB2 Control Center to create a user mapping for a group of users that will access a data source with the same user ID and password.

Action: Identify the user IDs that require a user mapping between the federated server and the data source. List the federated server user IDs and corresponding data source user IDs in the user mapping table in the planning checklist.

Related reference:
- "Checklist for planning your federated system configuration" on page 31

Plan the data type mappings

Data source data types are referred to as *remote* data types, and federated database data types are referred to as *local* data types.

For some data sources, the wrappers contain the default mappings between the data source data types and federated database data types. When you create a nickname for a data source object, information about the columns is stored in the federated database system catalog. The data types for the columns comes from the default forward data type mappings.

For other data sources, you must specify the column information and the data type when you create the nickname. Some of the nonrelational wrappers create all of the columns required to access a data source. These are called *fixed columns*. With other nonrelational data sources you can specify some or all of the data types for the columns.

Your applications might require data type mappings that are different than the default mappings. For the wrappers that allow you to specify data type mappings, you can override the default mappings to:
- Change a type mapping for all data source objects located on a specific server
- Change a type mapping for a specific data source object
- Change a type mapping for a specific data source type
- Change a type mapping for a specific data source type and version

Use the CREATE TYPE MAPPING statement to define new data type mappings. Mappings you create are stored in the federated database global catalog SYSCAT.TYPEMAPPINGS view.

Change a data type mapping *before* you create nicknames for the data source objects. When you create a nickname for a data source object, the federated server populates the global catalog with information about the table. This information includes the nickname, the data source table name, the column names and the data types that are defined for each table column.

Only nicknames created after a mapping is changed reflect the new type mapping. Nicknames created before the mapping is changed will use the default data type mapping.

If you create the data type mappings after you create the nicknames, you will have to alter each nickname to reflect the new mapping or drop and create the nicknames again.

Note: If a data source table contain columns that are distinct or user-defined data types, you have two choices:

- You can create the type mapping in the federated database before you create a nickname for that data source table. By creating the type mappings before you create the nickname, the federated server will know what data type to map these columns to. If the mappings for these distinct or user-defined data types are not created before you issue the CREATE NICKNAME statement, you will receive an error.
- If the columns in the data source table meet either of the following conditions:
  - The columns are user-defined data types that are based on system or built-in data types
  - The columns have attributes that are not supported for data type mappings

  You can create a view at the data source in which the columns are associated with or *cast* to the underlying built-in data type. Then create a nickname for the view instead of for the table.

Action: Identify the data type mappings that you want to define new mappings for. List the data sources and the type mappings you want to create in the data type mappings table in the planning checklist.

Related concepts:

- "Data type mappings in a federated system" in the *Federated Systems Guide*

Related reference:

- "Checklist for planning your federated system configuration" on page 31
- "Data types supported for nonrelational data sources" on page 517

Plan the function mappings

DB2 for UNIX and Windows supplies default function mappings between existing built-in data source functions and built-in DB2 functions. For most data sources, the default function mappings are in the wrappers. For some nonrelational data sources, you cannot alter the default function mappings.

To use a data source function that the federated server does not recognize, you must create a function mapping. The mapping you create is between the data source function and a counterpart function at the federated database. Function mappings are typically used when a new built-in function or a new user-defined function becomes available at the data source.

Function mappings are also used when a DB2 counterpart function does not exist. In this situation, before you create the function mapping you will have to create a function template in the federated database.

Action: Determine if you need to create function mappings for your data sources. List the function mappings needed in the function mappings table in the planning checklist.

Related concepts:

- "Function mappings" on page 17

Checklist for planning your federated system configuration

You can make the federated system configuration easier by following this planning checklist. This checklist guides you in ways to optimize the federated system configuration.

Checklist: Federated object naming rules

Are you familiar with the naming rules for federated objects?

See the related links at the end of this topic to locate information about the federated object naming rules.

Checklist: Preserving case-sensitive values

Do you intend to set the FOLD_ID and FOLD_PW server options to preserve case for user ID and password values sent to the data sources? Use the following table to identify which server definitions you will apply these options to.

Table 4. Planning checklist: FOLD_ID and FOLD_PW server options to set for the federated system

Data source	What name will you specify for the server in the server definition for this data source?	What setting will you specify for the FOLD_ID server option?	What setting will you specify for the FOLD_PW server option?

Checklist: Data source statistics

| In the following table, list the data sources that will be part of your federated
| system. Indicate the data sources that you will update the statistics for before you
| configure the federated server to access the data source. DB2 UDB for Linux,
| UNIX, and Windows is listed in this table as an example.

Table 5. Planning checklist: Data sources statistics to update for the federated system

Data source	Does this data source maintain catalog information? (Y/N)	Will you update statistics for this data source? (Y/N)	Name of the utility that the data source uses to update statistics
DB2 for Linux, UNIX, and Windows	Y	Y	RUNSTATS

Checklist: Data type mappings

In the following table, identify the data source data types and the corresponding federated server data types that you need to create a mapping for.

Table 6. Planning checklist: Data type mappings to create for the federated system

Data source	**What name will you specify for the server in the server definition for this data source?**	**Data source data type**	**Federated server data type**

Checklist: User mappings

In the following table, identify the federated server user IDs and corresponding user IDs for *each* data source that will be part of the federated system.

Table 7. Planning checklist: User mappings to create for the federated system

		Data source ________	**Data source** ________	**Data source** ________
User name	DB2 for Linux, UNIX, and Windows user ID	User ID	User ID	User ID

Checklist: Wrappers

In the following table, identify the wrappers that you will create.

Table 8. Planning checklist: Wrappers to create for the federated system

Data source	**Default wrapper name**	**Name that you will give the wrapper**
BioRS	none	
BLAST	none	

Table 8. Planning checklist: Wrappers to create for the federated system (continued)

Data source	Default wrapper name	Name that you will give the wrapper
Business applications (WebSphere Business Integration wrapper)	none	
DB2 Universal Database™ for Linux, UNIX, and Windows® DB2 Universal Database for z/OS and OS/390® DB2 Universal Database for iSeries DB2 Server for VM and VSE	DRDA	
Documentum	none	
Entrez	none	
Excel	none	
Extended Search	none	
HMMER	none	
Informix	INFORMIX	
Microsoft® SQL Server	MSSQLODBC3	
Oracle	NET8	
ODBC	none	
OLE DB	OLEDB	
Sybase	CTLIB	
Table-structured files	none	
Teradata	TERADATA	
Web services	none	
XML	none	

Related concepts:
- "Fast track to configuring your data sources" on page 55
- "Methods of accessing Excel data" on page 25

Related tasks:
- "Checking the setup of the federated server" on page 37
- "Creating a federated database" on page 51

Related reference:
- "Federated database national language considerations" on page 45
- "Federated object naming rules" on page 21
- "Preserving case-sensitive values in a federated system" on page 22
- "Update data source statistics" on page 23
- "Plan the data type mappings" on page 29
- "Plan the function mappings" on page 30
- "Plan the user mappings" on page 28
- "Choose the correct wrapper" on page 24

Part 2. Federated server and database

Chapter 3. Checking the setup of the federated server

Checking the setup of the federated server

You can avoid potential configuration problems by checking key settings on the federated server.

Procedure:

To check the setup of the federated server:
- Confirm the link-edit of the wrapper library files to the data source client software (UNIX)
- Check that the FEDERATED parameter is set to YES

After you check the setup of the federated server, you must create a federated database.

Related tasks:
- "Confirming the link-edit of the wrapper library files (UNIX)" on page 37
- "Checking the FEDERATED parameter" on page 42
- "Creating a federated database" on page 51

Confirming the link-edit of the wrapper library files (UNIX)

Confirming the link-edit of the wrapper library files (UNIX)

Confirming the link-edit of the wrapper library files is part of the larger task of checking the setup of the federated server.

On UNIX federated servers, some wrappers must be link-edited with the data source client software for the data source. The link-edit step is attempted when you install DB2 Information Integrator. The link-edit step creates a wrapper library for each data source that the federated server will communicate with.

This task applies to only the following data sources:
- Informix
- Microsoft SQL Server
- Oracle
- Sybase
- Teradata

Before you configure the federated server and database to access data sources, you should confirm that the link-edit of the wrapper library files was successful.

Prerequisites:

A federated server that is properly setup to access your data sources. This includes the installation and configuration of any required software, such as the data source client software.

Procedure:

Determine the status of the link-edit of the wrapper library files:

- If the link-edit was successful, the wrapper library file appears in the directory where DB2 Information Integrator is installed.
- If the link-edit failed, check the error message file in the directory where DB2 Information Integrator is installed.
- If the link-edit was not performed, neither the library file nor the message file appears in the directory where DB2 Information Integrator is installed. You will have to manually run the link script.

Related tasks:

- "Checking the wrapper library files (UNIX)" on page 38
- "Checking the link-edit message files (UNIX)" on page 40
- "Manually linking the wrapper libraries to the data source client software" on page 41
- "Adding data sources to a federated server using the DB2 UDB Control Center" on page 56

Checking the wrapper library files (UNIX)

Checking the wrapper library files is part of the larger task of confirming the link-edit between the wrapper libraries and the data source client software.

The wrapper library files are required so that you can access the data sources. For some data sources, the library files are added to the federated server when you install DB2 Information Integrator. For other data sources a link-edit script must be run to create the library files.

This task applies to only the following data sources:

- Informix
- Microsoft SQL Server
- Oracle
- Sybase
- Teradata

Procedure:

To check if the wrapper library files are on your federated server:

1. Check for the library files in the directory path for the wrapper library. You must confirm that a library files exists on the federated server for each data source that you want to access.
2. If the library files are not in the directory, you must manually link the wrapper libraries to the data source client software.

Related tasks:

- "Checking the setup of the federated server" on page 37
- "Checking the link-edit message files (UNIX)" on page 40
- "Manually linking the wrapper libraries to the data source client software" on page 41

Related reference:

- "Wrapper library files" on page 39
- "Life sciences user-defined function library files" on page 443

Wrapper library files

The wrapper library files are required so that you can access the data sources. For some data sources, the library files are added to the federated server when you install DB2 Information Integrator. For other data sources a link-edit script must be run to create the library files.

The wrapper library files are required when you register the wrapper for the data source.

You should verify that the wrapper library files are on your federated server. There should be a set wrapper library files for each of the data sources that you want to access.

If the wrapper library files are not on the federated server, you must manually run the link-edit script to create the library files.

Related tasks:

- "Checking the wrapper library files (UNIX)" on page 38
- "Manually linking the wrapper libraries to the data source client software" on page 41

Related reference:

- "BLAST wrapper library files" on page 106
- "BioRS wrapper library files" on page 72
- "DB2 wrapper library files" on page 160
- "Documentum wrapper library files" on page 174
- "Entrez wrapper library files" on page 197
- "Excel wrapper library files" on page 220
- "Extended Search wrapper library files" on page 236
- "HMMER wrapper library files" on page 262
- "Informix wrapper library files" on page 281
- "Microsoft SQL Server wrapper library files" on page 296
- "ODBC wrapper library files" on page 308
- "OLE DB wrapper library files" on page 323
- "Oracle wrapper library files" on page 333
- "Sybase wrapper library files" on page 346
- "Teradata wrapper library files" on page 371
- "Table-structured files wrapper library files" on page 358
- "WebSphere Business Integration wrapper library files" on page 127
- "Web services wrapper library files" on page 388
- "XML wrapper library files" on page 419

Checking the link-edit message files (UNIX)

Checking the link-edit message files is part of the larger task of confirming the link-edit between the wrapper libraries and data source client software.

If the link-edit fails, errors are listed in the message file in the library directory. The existence of an message file does not mean that the link-edit failed. There is a message file in the library directory even if the link-edit is successful.

You must open the message file to determine if the link-edit failed.

This task is required for only the following data sources:
- Informix
- Microsoft SQL Server
- Oracle
- Sybase
- Teradata

Procedure:

To determine if the link-edit failed, open the link-edit message files. The link-edit message files are in the directory where DB2 is installed, in the `lib` or `lib64` subdirectory. The names of the link-edit message files are listed in the following table.

Table 9. Link-edit message file names by data source

Data source	Message file names
Informix	djxlinkInformix.out
Microsoft SQL Server	djxlinkMssql.out
Oracle	djxlinkOracle.out
Sybase	djxlinkSybase.out
Teradata	djxlinkTeradata.out

There are several reasons why the link might fail when you setup the federated server:
- If the data source client software is not installed before the link-edit is attempted, then the link-edit will fail. For example, if you do not install the Informix client software before you install the DB2 server software, the link-edit will fail. Likewise, if you do not install the Sybase Open Client software before you install DB2 Information Integrator, the link-edit will fail. In these situations, you will have to perform the link manually.
- Verify that the version of the data source client software is supported. If the version of the data source client software that you have installed is not supported, the link-edit will fail. You will have to install a client version that is supported and then perform the link manually.

Related tasks:
- "Confirming the link-edit of the wrapper library files (UNIX)" on page 37
- "Checking the wrapper library files (UNIX)" on page 38
- "Manually linking the wrapper libraries to the data source client software" on page 41

Manually linking the wrapper libraries to the data source client software

Manually linking the wrapper libraries to the data source client software is part of the larger task of checking the setup of the federated server.

If the library files are not in the directory path, you must manually link the wrapper libraries.

This task applies only to the following data sources:
- Informix
- Microsoft SQL Server
- Oracle
- Sybase
- Teradata

Prerequisites:

You need root authorization to run the link scripts.

For the djxlinkxxx scripts to issue their messages in your language, there must be at least one DB2 instance. If an instance does not exist, the scripts will still work. However, you will receive error messages. Each error message begins with `db2djxmsg: Error retrieving message number`. This error message is followed by a message in English. For example:

```
db2djxmsg: Error retrieving message number 9004
  (return code -2029059891 from sqlogmsg).
Begin processing for wrapper: INFORMIX
INFORMIXDIR = /wsdb/v82/bldsupp/AIX/informix2.81
db2djxmsg: Error retrieving message number 9015
  (return code -2029059891 from sqlogmsg).
Library libdb2informixF.a was built successfully.
db2djxmsg: Error retrieving message number 9006
  (return code -2029059891 from sqlogmsg).
End processing for wrapper: INFORMIX
```

Procedure:

To link the wrapper libraries to the data source client software quickly:
1. Install and configure the client software on the DB2 federated server (if necessary).
2. Use the product CDs to perform the link:
   - For Informix data sources, run the DB2 server installation again and specify the Typical installation option.
   - For Microsoft SQL Server, Oracle, Sybase, and Teradata data sources run the DB2 Information Integrator installation again. From the launchpad, click **Install Products** and follow the instructions in the wizard.
3. After the link is performed, check the permissions on the wrapper libraries. Make sure that the libraries can be read and executed by the DB2 instance owners.

Alternatively, you can run the link-edit scripts from the UNIX command prompt.
1. Open a UNIX command prompt.
2. Run the link-edit script for each data source that you want to access.

The following table lists the names of link-edit script for each data source.

Table 10. Link scripts by data source

Data source	Link script name
Informix	djxlinkInformix
Microsoft SQL Server	djxlinkMssql
Oracle	djxlinkOracle
Sybase	djxlinkSybase
Teradata	djxlinkTeradata

For example, if you are setting up the federated server to access Informix data sources, run the `djxlinkInformix` script from the UNIX command prompt:

```
djxlinkInformix
```

3. Issue the **db2iupdt** command on each DB2 instance to enable federated access to the data sources.
4. After the link is performed, check the permissions on the wrapper libraries. Make sure that the libraries can be read and executed by the DB2 instance owners.

Attention: There is another script, the `djxlink` script, that attempts to create a wrapper library for every data source that is supported by DB2 Information Integrator. If you run `djxlink` script and have the client software for only some of the data sources installed, you will receive an error message for each of the data sources that you do not have installed.

Related tasks:

- "Installing DB2 Information Integrator (Windows)" in the *IBM DB2 Information Integrator Installation Guide for Linux, UNIX, and Windows*
- "Installing DB2 Information Integrator (UNIX)" in the *IBM DB2 Information Integrator Installation Guide for Linux, UNIX, and Windows*
- "Checking the setup of the federated server" on page 37
- "Checking the FEDERATED parameter" on page 42
- "Checking the link-edit message files (UNIX)" on page 40

Checking the FEDERATED parameter

Checking the FEDERATED parameter is part of the larger task of checking the setup of the federated server.

Before you add data sources to the federated server and database, you should check the FEDERATED parameter setting.

The FEDERATED parameter must be set to YES to enable the federated server to access to the data sources.

Prerequisites:

- DB2 Information Integrator must be installed on a server that will act as the federated server

Procedure:

To check the FEDERATED parameter setting:

1. Issue the following DB2 command to display all of the parameters and their current settings:
```
GET DATABASE MANAGER CONFIGURATION
```
2. Check the CONCENTRATOR parameter setting. The CONCENTRATOR parameter and the FEDERATED parameter cannot be configured to YES at the same time. If the CONCENTRATOR parameter is set to YES, change the setting to NO. Issue the following DB2 command to change the setting:
```
UPDATE DATABASE MANAGER CONFIGURATION USING CONCENTRATOR NO
```
3. Check the FEDERATED parameter setting. If the FEDERATED parameter is set to NO, change the setting to YES. Issue the following DB2 command to change the setting:
```
UPDATE DATABASE MANAGER CONFIGURATION USING FEDERATED YES
```

Related concepts:

- "DB2 Information Integrator installation process - overview" in the *IBM DB2 Information Integrator Installation Guide for Linux, UNIX, and Windows*

Related tasks:

- "Checking the setup of the federated server" on page 37

Chapter 4. Creating a federated database

Before you can configure the federated server to access your data sources, you must create a database that is used as the federated database.

Federated database national language considerations

For many relational data sources, the wrapper performs the following tasks when the wrapper connects to the data source:

1. Determines the codepage and territory of the federated database.
2. Maps the codepage and territory to a data source client locale name.
3. Depending on the data source, sets an environment variable or calls a data source API to tell the data source what the client locale is.
4. The data source then converts character data between the codepage of the remote database and the codepage of the federated database.

   For example, if the federated database uses codepage 819, territory US. The equivalent Oracle client locale is American_America.WE8ISO8859P1. The wrapper will set the NLS_LANG variable to the Oracle client locale value. When data is sent from the Oracle database to the wrapper, the Oracle database converts the data from codeset American_America.WE8ISO8859P1 to codepage 819. When data is sent from the Oracle database to the wrapper, the Oracle server or client converts the data from the codepage of the Oracle database to codepage 819. When data is sent from the wrapper to the Oracle database, the Oracle server or client converts the data from codepage 819 to the codepage of the Oracle database.

For relational data sources that do not perform codepage conversion, some of the wrappers perform the conversion.

Defining the federated database to use the same codeset, territory, and collating sequence as your data source can improve performance. If you define the federated database to use the same codeset, territory, and collating sequence as your data source, then the codepage conversion is not necessary. Using the same national language settings can improve performance when you transfer large amounts of character data.

To specify the codeset and territory on the federated database, you use the USING CODESET and TERRITORY parameters on the CREATE DATABASE statement.

Related concepts:

- “Unicode support for federated systems” on page 48
- “Collating sequences in a federated system” on page 46

Related tasks:

- “Creating a federated database” on page 51
- “Setting the federated database collating sequence” on page 47

Collating sequences in a federated system

When the federated server receives a query, the DB2® SQL Compiler consults information in the global catalog and the data source wrapper to help it process the query. As part of the SQL Compiler process, the *query optimizer* analyzes a query. The Compiler develops alternative strategies, called *access plans*, for processing the query. The access plans might call for the query to be:

- Processed by the data sources
- Processed by the federated server
- Processed partly by the data sources and partly by the federated server

Performing character sorts and comparisons at the data source usually improves performance.

If the query requires character sorting or comparisons, the SQL Compiler uses collating sequence information to determine which access plan to use.

By default, the federated database collating sequence is case-sensitive. However some of the federated data sources use collating sequences that are case-insensitive. If the collating sequences of the federated database and the data source are different, the query results might differ. If the operation is a character sort, the same data is returned but the order of the results will be different. If the operation is a character comparison, the results returned might be different.

Where the sorting or comparison is processed depends on several factors:

- If the federated database collating sequence is the same as the data source collating sequence, the character sort or comparison can take place at the data source. The query optimizer can decide which is the most efficient way to complete the query, a local operation or a remote operation. It is assumed that all types of character comparisons and sorts by the data source would yield the same results as if those actions were performed by the federated database.
- If the federated database collating sequence is different than the data source collating sequence, but the data source collating sequence is case-sensitive, the character sort or comparison will take place at the federated database. It is assumed that the data source will yield the same results on character data for WHERE=, DISTINCT, and GROUP BY operations. But other operations, such as ORDER BY and WHERE with a greater than or less than predicate, will yield different results on character data.
- If the federated database collating sequence is different than the data source collating sequence, but the data source collating sequence is case-insensitive, the character sort or comparison will take place at the federated database. It is assumed that the data source will count uppercase and lowercase letters as equivalent and will include them both in a result, regardless of whether uppercase or lowercase was specified in the requested operation. WHERE=, WHERE with a greater than or less than predicate, ORDER BY, DISTINCT, and GROUP BY operations will not be pushed down to the data source.

For example, a case-insensitive data source assigns the same weights to the characters ″S″ and ″s″. A case-insensitive data source with an English code page considers the words **STEWART**, **SteWArT**, and **stewart** to be equal. However when a case-sensitive collating sequence is used, different weights are assigned to the characters. Depending on the sensitivity of the collating sequence, the result set of a character sort or comparison will be different.

When the collating sequences of the federated database and the data source differ, the federated server retrieves the data and the character sorts and comparisons are performed locally. The reason for performing these tasks locally is that DB2 users expect to see the query results ordered according to the collating sequence defined for the federated server. By ordering the data locally, DB2 users are guaranteed that the result sets will be consistent. Retrieving data for local sorts and comparisons usually decreases performance.

If you need to see the character data ordered in the data source collating sequence, you can submit your query in a pass-through session.

To determine if a data source and the federated database have the same collating sequence, consider the following factors:

Code page
: The code page scheme, such as ASCII and EBCDIC, that is used by the federated server and the data source impacts the results.

National language support (NLS)
: The collating sequence is related to the language supported on a server. Compare the DB2 NLS information for your operating system to the data source NLS information.

Data source characteristics
: Some data sources are created using case-insensitive collating sequences, which can yield different results from DB2 in order-dependent operations.

Customization
: Some data sources provide multiple options for collating sequences or allow the collating sequence to be customized.

There are several options that you have for setting the collating sequence, you can:
- Set the collating sequence when you create the federated database
- Set the COLLATING_SEQUENCE option when you create the server definition for a data source. This option is available only for relational data sources.

Related concepts:
- "Unicode support for federated systems" on page 48

Related tasks:
- "Creating a federated database" on page 51
- "Setting the federated database collating sequence" on page 47

Related reference:
- "Federated database national language considerations" on page 45

Setting the federated database collating sequence

Administrators can create federated databases with a particular collating sequence that matches a data source collating sequence.

You set the federated database collating sequence as part of the CREATE DATABASE API. Through this API, you can specify one of the following sequences:
- An identity sequence

- A *system* sequence (the sequence used by the operating system that supports the database)
- A *customized* sequence (a predefined sequence that DB2 UDB supplies or that you define yourself)

Procedure:

To specify the collating sequence of the federated database, you use the COLLATE USING parameter on the CREATE DATABASE statement.

For relational data sources, if the federated database and the data source use the same collating sequence, you should set the COLLATING_SEQUENCE server option to 'Y'. Setting the COLLATING_SEQUENCE server option to 'Y' tells the federated server that the collating sequences of the federated database and the data source match. You set the COLLATING_SEQUENCE server option when you create the server definitions for the relational data sources.

The relational data sources that support the COLLATING_SEQUENCE server option are:

- DB2 family
- Informix
- Microsoft SQL Server
- ODBC
- OLE DB
- Oracle
- Sybase
- Teradata

Related concepts:
- "Unicode support for federated systems" on page 48
- "Collating sequences in a federated system" on page 46

Related tasks:
- "Creating a federated database" on page 51

Related reference:
- "Federated database national language considerations" on page 45

Unicode support for federated systems

Relational and nonrelational wrappers and user-defined functions can run on a Unicode database (UTF-8 database). The Unicode database provides federated server environments that are platform independent. The Unicode database can manipulate data that is stored in various code pages on different data sources.

The wrappers and user-defined functions that support Unicode are:

- Relational wrappers
  - DRDA®
  - Informix®
  - MS SQL Server
  - ODBC
  - OLE DB

  - Oracle
  - Sybase
  - Teradata
- Nonrelational wrappers and user-defined functions
  - BioRS wrapper
  - BLASTwrapper
  - Documentum wrapper
  - Entrez wrapper
  - Excel wrapper
  - HMMER wrapper
  - IBM® Lotus® Extended Search wrapper
  - KEGG user-defined functions
  - MQ user-defined functions
  - Table-structured file wrapper
  - Web services user-defined functions
  - Web services wrapper
  - WebSphere® Business Integration wrapper
  - XML wrapper

In Figure 5 on page 50 a company has branch offices in different countries. Each branch office stores customer data with its own databases in their own code page. The Microsoft® SQL Server database stores data in code page A. The Oracle database stores data in code page B. Code page A and code page B are in different territories. To integrate the data from the different territories, the company can set the federated database's code page to Unicode. The company can then join the tables to see the total number of purchase orders, regardless of territory.

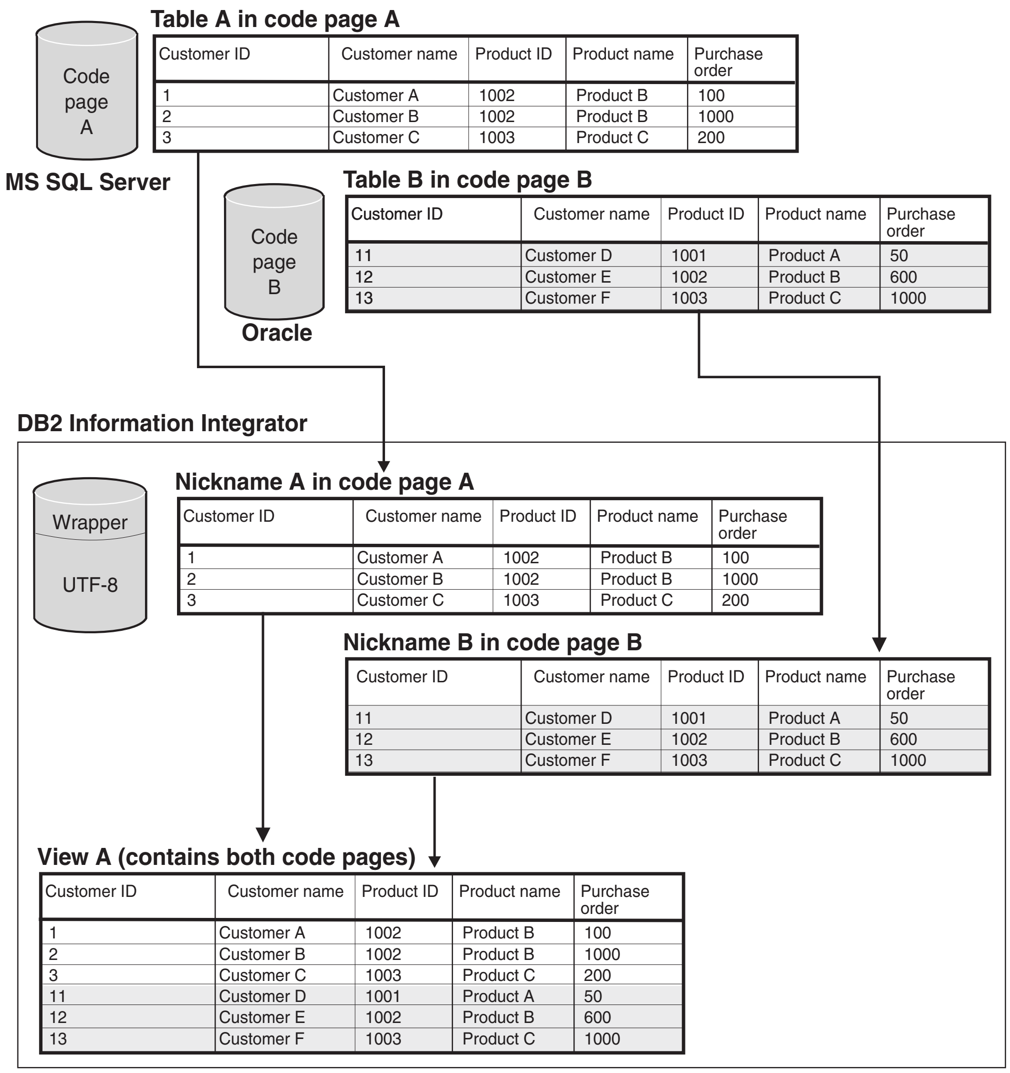


Table A in code page A

Customer ID	Customer name	Product ID	Product name	Purchase order
1	Customer A	1002	Product B	100
2	Customer B	1002	Product B	1000
3	Customer C	1003	Product C	200

Table B in code page B

Customer ID	Customer name	Product ID	Product name	Purchase order
11	Customer D	1001	Product A	50
12	Customer E	1002	Product B	600
13	Customer F	1003	Product C	1000

Nickname A in code page A

Customer ID	Customer name	Product ID	Product name	Purchase order
1	Customer A	1002	Product B	100
2	Customer B	1002	Product B	1000
3	Customer C	1003	Product C	200

Nickname B in code page B

Customer ID	Customer name	Product ID	Product name	Purchase order
11	Customer D	1001	Product A	50
12	Customer E	1002	Product B	600
13	Customer F	1003	Product C	1000

View A (contains both code pages)

Customer ID	Customer name	Product ID	Product name	Purchase order
1	Customer A	1002	Product B	100
2	Customer B	1002	Product B	1000
3	Customer C	1003	Product C	200
11	Customer D	1001	Product A	50
12	Customer E	1002	Product B	600
13	Customer F	1003	Product C	1000

Figure 5. Unicode example

Related tasks:

- "Specifying the client code page for Unicode support of Microsoft SQL Server and ODBC data sources" in the *Federated Systems Guide*
- "Specifying the file code page for Unicode support of table-structured file data sources" in the *Federated Systems Guide*

Related reference:

- "Unicode default forward data type mappings - NET8 wrapper" in the *Federated Systems Guide*
- "Supported Unicode code pages for the MSSQL and ODBC wrapper CODEPAGE option" in the *Federated Systems Guide*
- "Unicode default reverse data type mappings - NET8 wrapper" in the *Federated Systems Guide*
- "Unicode default forward data type mappings - Sybase wrapper" in the *Federated Systems Guide*
- "Unicode default reverse data type mappings - Sybase wrapper" in the *Federated Systems Guide*

- "Unicode default forward data type mappings - ODBC wrapper" in the *Federated Systems Guide*
- "Unicode default reverse data type mappings - ODBC wrapper" in the *Federated Systems Guide*
- "Unicode default forward data type mappings - Microsoft SQL Server wrapper" in the *Federated Systems Guide*
- "Unicode default reverse data type mappings - Microsoft SQL Server wrapper" in the *Federated Systems Guide*
- "Specifying the file code page for Unicode support of table-structured file data sources - example" in the *Federated Systems Guide*

Creating a federated database

After you set up the federated server, the DB2 instance owner must create a DB2 database on the federated server instance that will act as the federated database.

Recommendation: If the remote data sources that you need to connect to are using different or incompatible codepages, define the federated database as a Unicode database. To define the federated database as a Unicode database, specify USING CODESET UTF-8 on the CREATE DATABASE statement.

This step must be completed before you can configure the federated server to access your data sources.

Prerequisites:
- SYSADM or SYSCTRL authority to create a DB2 database.
- DB2 Information Integrator must be installed on a server that will act as the federated server
- Determine if you want to specify a collating sequence when you create the federated database

Procedure:

You can create the federated database from the DB2 Control Center or the DB2 command line.

To do this task from the DB2 Control Center:
1. Right-click on the **Databases** folder and click **Create –>Database Using Wizard**. The Create Database Wizard opens.
2. Complete the steps in the wizard.

To do this task from the DB2 command line, issue the CREATE DATABASE command. For example:

```
CREATE DATABASE federated
```

This command:
- Initializes a new database
- Creates the three initial table spaces
- Creates the system tables
- Allocates the recovery log

If your DB2 instance uses a multiple partition configuration, the CREATE DATABASE command affects all of the partitions that are listed in the `db2nodes.cfg` file. The database partition from which this command is issued becomes the catalog partition for the new database.

Related concepts:

- "DB2 Information Integrator installation process - overview" in the *IBM DB2 Information Integrator Installation Guide for Linux, UNIX, and Windows*
- "Fast track to configuring your data sources" on page 55

Related tasks:

- "Setting the federated database collating sequence" on page 47

Part 3. Data sources

Chapter 5. Overview of configuring access to data sources

The following sections provide a concise guide to configuring a federated server and database to access your data sources:

- They contain information about the basic steps needed to quickly perform the configuration steps.
- They outline several optional steps, if you need them, to fine-tune the data source configuration.

There are individual configuration chapters for each data source.

Fast track to configuring your data sources

You can accomplish most of the steps required to configure access to a data source through the DB2® Control Center. Use the DB2 Command Center for the steps that require a command line. Toggle between these graphical user interfaces to quickly configure access to a data source.

Before you configure access to a data source, make sure that the federated server has been set up properly.

The steps to configure the federated server to access a data source are similar, regardless of the data source. The basic steps and recommended interface are:

Table 11. The recommended interface and configuration steps

Configuration step	Recommended interface	Notes
1. Prepare the federated server for the data source.	Client Configuration Assistant	Required for only some data sources. This step might require you to install software, configure a file, or check a setting.
2. Set the required environment variables.	DB2 Control Center	Environment variables are required for: • Documentum • Informix® • Microsoft® SQL Server • Oracle • Sybase • Teradata
2. Register the wrappers.	The Federated Objects wizard in the DB2 Control Center.	A wrapper is required for each data source that you want to access.
3. Register the server definitions.	The Federated Objects wizard in the DB2 Control Center.	Server definitions are associated with a wrapper and used when you register nicknames.

Table 11. The recommended interface and configuration steps (continued)

Configuration step	Recommended interface	Notes
4. Create the user mappings.	The Federated Objects wizard in the DB2 Control Center.	Required for only some data sources. If you attempt to retrieve the remote password associated with a user mapping from the SYSCAT.USEROPTIONS catalog view, the remote password value is displayed encrypted.
5. Test the connection to the data source server.	DB2 Command Center	Required for only some data sources.
6. Create the nicknames.	The Federated Objects wizard in the DB2 Control Center.	A nickname is required for each data source object that you want to access.

Related concepts:

- "Optional configuration steps" on page 66

Related tasks:

- "Adding data sources to a federated server using the DB2 UDB Control Center" on page 56
- "Registering wrappers for a data source" on page 61
- "Registering server definitions for a data source" on page 61
- "Registering user mappings for a data source" on page 63
- "Registering nicknames for a data source" on page 63
- "Setting the data source environment variables" on page 58
- "Checking the setup of the federated server" on page 37

Adding data sources to a federated server using the DB2 UDB Control Center

To configure the federated server to access data sources, you must provide the federated server with information about the data sources and objects that you want to access.

Procedure:

The DB2 Control Center includes a wizard to guide you through the steps required to configure the federated server. To start the wizard, right-click the **Federated Database Objects** folder and click **Create Federated Objects**.

The steps that are required to configure the federated server are different for each data source.

You can configure multiple federated servers to access data sources by using the Action Output window.

Related tasks:

- “Configuring multiple federated servers to access data sources” on page 57
- “Adding table-structured file data sources to a federated server” on page 357
- “Adding Documentum data sources to a federated server” on page 171
- “Adding Excel data sources to a federated server” on page 218
- “Adding BLAST data sources to a federated server” on page 98
- “Adding Entrez data sources to a federated server” on page 194
- “Adding Teradata data sources to a federated server” on page 365
- “Adding BioRS data sources to a federated server” on page 68
- “Adding DB2 family data sources to a federated server” on page 157
- “Adding Extended Search data sources to a federated server” on page 235
- “Adding HMMER data sources to a federated server” on page 254
- “Adding Informix data sources to federated servers” on page 275
- “Adding Microsoft SQL Server data sources to a federated server” on page 291
- “Adding ODBC data sources to a federated server” on page 305
- “Adding OLE DB data sources to a federated server” on page 321
- “Adding Oracle data sources to a federated server” on page 327
- “Adding Sybase data sources to a federated server” on page 341
- “Adding business application data sources to a federated system” on page 125
- “Adding Web services data sources to a federated server” on page 387

Configuring multiple federated servers to access data sources

A federated system can consist of multiple federated servers. Instead of configuring each federated server separately, you can save time by using the DB2 Control Center to configure the federated servers. When you configure the first server, the **Action Output** window captures the DDL statements that are issued when you create the federated objects. You can reuse or modify these statements, and apply the statements to quickly configure additional federated servers.

The **Action Output** window remains active for the current session. If you close the **Action Output** window, the DDL statements for the current session continue to be stored in the **Action Output** window. However, if you close the DB2 Control Center all of the DLL statements from the current session are removed from the **Action Output** window.

Prerequisites:
- DB2 Information Integrator installed on a server that will act as the federated server
- A federated database must exist on the federated server

Procedure:

To configure multiple federated servers to access data sources:
1. Using the DB2 Control Center, configure the first federated server for the data sources that you want to access. This captures each DDL statement.
2. Display the **Action Output** page in the **Action Output** window.

   If you closed the **Action Output** window, right-click the **Federated Database Objects** folder and click **Show Actions** to open the **Action Output** window.

3. Delete any DDL statements that you do not want to use on the other federated servers. To delete a statement, right-click the statement and click **Remove**. For example, you might want to delete any statements that display Failed in the status column on the **Action Output** page.
4. Copy that statements that you want to use on the other federated servers to the **Command Editor** page:
   a. Select the statements that you want to copy. To select multiple statements, use the Ctrl key.
   b. Right-click on the selected statements and click **Copy to Command Editor**. The **Command Editor** page opens.
5. Change any DDL statements in the **Command Editor** page that you want to use on the other federated servers. For example, you might want to change any statement that specifies a local schema.

   You must change the user mappings to specify the passwords. When the DDL for the CREATE USER MAPPING statements is captured in the **Action Output** window, the passwords are masked by asterisks. You must replace the asterisks with the proper passwords.
6. Issue the DDL statements on the next federated server.

Related concepts:
- "DB2 Information Integrator installation process - overview" in the *IBM DB2 Information Integrator Installation Guide for Linux, UNIX, and Windows*
- "Fast track to configuring your data sources" on page 55

Related tasks:
- "Creating a federated database" on page 51

Setting the data source environment variables

Setting the data source environment variables

When you install DB2 Information Integrator, the installation process attempts to set the environment variables that are required by some of the data sources. The installation process might not be able to set the environment variables if, for example, you do not have the client software installed on the federated server before you install DB2 Information Integrator.

When you follow the steps to add a data source to a federated server, you can check the environment variables and set them (if necessary).
- If you use the DB2 Control Center to add data sources to the federated server, the requirements for the environment variables are automatically checked. You can set the environment variables when you create or alter a wrapper.
- If you use the DB2 command line to add data sources to the federated server, you must set the environment variables manually.

Setting the environment variables is required for the following data sources:
- Documentum
- Informix
- Microsoft SQL Server
- Oracle
- Sybase

- Teradata

Prerequisites:

This task should be performed by the system administrator.

Restrictions:

See the topic: Restrictions for the db2dj.ini file

Procedure:

The steps to check the environment variables are different for each data source.

Related concepts:
- "Fast track to configuring your data sources" on page 55

Related tasks:
- "Setting the Documentum environment variables" on page 172
- "Registering wrappers for a data source" on page 61
- "Setting the Informix environment variables" on page 277
- "Setting the Microsoft SQL Server environment variables" on page 293
- "Setting the Oracle environment variables" on page 328
- "Setting the Sybase environment variables" on page 342
- "Setting the Teradata environment variables" on page 368

Related reference:
- "Restrictions for the db2dj.ini file" on page 59

Restrictions for the db2dj.ini file

The following restrictions apply to the `db2dj.ini` file:
- Entries must use the format *evname=value*

  evname is the name of the environment variable and *value* is its value.
- The environment variable name has a maximum length of 255 bytes.
- The environment variable value has a maximum length of 765 bytes.
- The maximum length of any line in the file is 1021 bytes. Data beyond that length is ignored.

Related tasks:
- "Setting the Documentum environment variables" on page 172
- "Setting the data source environment variables" on page 58
- "Setting the Informix environment variables" on page 277
- "Setting the Microsoft SQL Server environment variables" on page 293
- "Setting the Oracle environment variables" on page 328
- "Setting the Sybase environment variables" on page 342
- "Setting the Teradata environment variables" on page 368

Applying environment variables in a multi-partition instance configuration

If your federated server instance has a multi-partition configuration, you must apply the data source environment variables to all partitions.

This step is necessary only if your federated server has a multiple-partition instance configuration.

The `db2dj.ini` file contains the data source environment variables. This file was added to the federated server when you installed DB2 Information Integrator.

You must add a copy of the same `db2dj.ini` file to all of the partitions in your multiple-partition instance configuration. The default name of the file is `db2dj.ini` file. On UNIX federated servers, the default path for the `db2dj.ini` file is `INSTHOME/sqllib/cfg`, where `INSTHOME` is the home directory of the instance owner. On Windows federated servers, the default path to the file is `x:\SQLLIB\cfg`, where `x:\SQLLIB` is the drive and directory specified in the DB2PATH registry variable or environment variable.

The default path and file name for the `db2dj.ini` file can be overridden by the DB2_DJ_INI registry variable.

You can apply the DB2_DJ_INI registry variable to all of the partitions or a subset of the partitions.

Procedure:

To apply the DB2_DJ_INI registry variable to the appropriate partitions on your federated server, use the **db2set** command. The **db2set** command displays, sets, or removes DB2 profile variables.

The syntax of the **db2set** command that you use dependents on your database system structure.

- To apply the DB2_DJ_INI registry variable to all database partitions within this instance, issue this command
  ```
  db2set -g DB2_DJ_INI=$HOME/sqllib/cfg/my_db2dj.ini
  ```
- To apply the DB2_DJ_INI registry variable to only the current partition, issue this command:
  ```
  db2set DB2_DJ_INI=$HOME/sqllib/cfg/my_db2dj.ini
  ```
- To apply the DB2_DJ_INI registry variable to a specific partition, issue this command:
  ```
  db2set -i INSTANCEX  3 DB2_DJ_INI=$HOME/sqllib/cfg/partition3.ini
  ```

  INSTANCEX
  : The name of the instance.

  3
  : The partition number as listed in the `db2nodes.cfg` file.

  partition3.ini
  : The modified and renamed version of the `db2dj.ini` file.

Attention: When you set the DB2_DJ_INI registry variable, you must set it to an absolute path. If the FEDERATED parameter is set to YES and the DB2_DJ_INI registry variable is set to a relative path, the DB2 Universal Database engine will not start.

Related tasks:

- "Setting the data source environment variables" on page 58

Registering wrappers for a data source

You register one wrapper for each type of data source that you want to access. To access three DB2 for z/OS database tables, one DB2 for iSeries table, and two BLAST search types, you need to create two wrappers. You need to register one wrapper for the DB2 databases and one wrapper for the BLAST search types.

After the wrappers are registered in the federated database, you can use these wrappers to access other objects from those data sources. For example, you can use the DRDA wrapper to access data sources from all of the DB2 family data source objects, including DB2 for Linux, UNIX, and Windows, DB2 for z/OS and OS/390, DB2 for iSeries, and DB2 Server for VM and VSE.

Procedure:

You can register a wrapper from the DB2 Control Center or the DB2 command line:

- To do this task from the DB2 Control Center, use the Federated Objects wizard or right-click the **Federated Objects** folder and click **Create Wrapper**.
- To do this task from the DB2 command line, use the CREATE WRAPPER statement.

Related tasks:

- "Registering the table-structured file wrapper" on page 358
- "Registering the Documentum wrapper" on page 174
- "Registering the Excel wrapper" on page 219
- "Registering the BLAST wrapper" on page 105
- "Registering the XML wrapper" on page 419
- "Registering the Entrez wrapper" on page 196
- "Registering the Teradata wrapper" on page 370
- "Registering the BioRS wrapper" on page 71
- "Registering the DB2 wrapper" on page 160
- "Registering the Extended Search wrapper" on page 236
- "Registering the HMMER wrapper" on page 262
- "Registering the Informix wrapper" on page 280
- "Registering the Microsoft SQL Server wrapper" on page 295
- "Registering the ODBC wrapper" on page 307
- "Registering the OLE DB wrapper" on page 322
- "Registering the Oracle wrapper" on page 332
- "Registering the Sybase wrapper" on page 345
- "Registering the WebSphere Business Integration wrapper" on page 126
- "Registering the Web services wrapper" on page 388

Registering server definitions for a data source

The purpose of a server definition varies from data source to data source.

A server definition for relational data sources usually represents a remote database, database partition, or node. For nonrelational data sources, some server definitions map to a search type and daemon, to a web site, or to a web server. For other nonrelational data sources, a server definition is created only because it is required by federation.

Every data source object that you create a nickname for must be associated with a specific server definition.

For some data sources, you must specify a node when you register a server definition. The concept of a node varies from data source to data source. For relational data sources, a node reflects a server instance of the data source.

Procedure:

You can register a server definition from the DB2 Control Center or the DB2 command line:

- To do this task from the DB2 Control Center, use the Federated Objects wizard or right-click the **Server Definitions** folder and click **Create**.
- To do this task from the DB2 command line, use the CREATE SERVER statement.

Related tasks:

- "Registering the server for table-structured files" on page 359
- "Registering the server for Documentum data sources" on page 175
- "Registering the server for an Excel data source" on page 220
- "Registering the server for a BLAST data source" on page 106
- "Registering the server for an XML data source" on page 420
- "Registering the server for an Entrez data source" on page 197
- "Registering the server definitions for a Teradata data source" on page 371
- "Registering the server definition for a BioRS data source" on page 72
- "Registering the server definitions for a DB2 data source" on page 161
- "Registering the server for Extended Search data sources" on page 237
- "Registering the server definition for a HMMER data source" on page 263
- "Registering the server definitions for an Informix data source" on page 282
- "Registering the server definitions for a Microsoft SQL Server data source" on page 297
- "Registering the server definitions for an ODBC data source" on page 309
- "Registering the server definitions for an OLE DB data source" on page 323
- "Registering the server definitions for an Oracle data source" on page 334
- "Registering the server definitions for a Sybase data source" on page 347
- "Registering the server definition for business application data sources" on page 127
- "Registering the server definition for Web services data sources" on page 389

Registering user mappings for a data source

For same data sources, you must define an association between the federated server authorization ID and the data source user ID and password. You create a user mapping for each user ID that uses the federated system to send distributed requests.

Procedure:

You can create a user mapping from the DB2 Control Center or the DB2 command line:

- To do this task from the DB2 Control Center, use the Federated Objects wizard or right-click the **User Mappings** folder and click **Create**.
- To do this task from the DB2 command line, use the CREATE USER MAPPING statement.

Related tasks:

- "Registering user mappings for Documentum data sources" on page 176
- "Creating the user mapping for a Teradata data source" on page 373
- "Registering user mappings for BioRS data sources" on page 73
- "Creating the user mapping for a DB2 data source" on page 163
- "Registering user mappings for Extended Search data sources" on page 238
- "Creating the user mapping for an Informix data source" on page 284
- "Creating a user mapping for a Microsoft SQL Server data source" on page 299
- "Creating a user mapping for an ODBC data source" on page 311
- "Creating a user mapping for an OLE DB data source" on page 324
- "Creating the user mappings for an Oracle data source" on page 335
- "Creating a user mapping for a Sybase data source" on page 349

Registering nicknames for a data source

Registering nicknames for a data source

The task of registering a nickname is typically the most involved of the configuration tasks. The steps and requirements for registering a nickname are different for each data source.

Recommendation: Use the DB2 Control Center to register nicknames. For most data sources, you can use the Discover tool in the DB2 Control Center. The Discover tool helps you to quickly identify data source objects that you might want to register nicknames for.

You must register a nickname for each data source object that you want to access.

Data source objects can be relational or nonrelational:

- Examples of relational data source objects are database tables, views, and synonyms (Informix and Oracle only)
- Examples of nonrelational data source objects are BLAST-able databases, objects and registered tables in a Documentum Docbase, Microsoft Excel files (.xls), table-structured files (.txt), and XML tagged files

Tables and views that reside in the federated database are *local objects*. You do not register nicknames for these objects. You use the actual object name in your queries.

Data source objects are:
- Tables and views in another DB2® database instance on the federated server.
- Tables and views in a DB2 instance on another server.
- Data source objects that reside in another data source, such as: Oracle, Sybase, Documentum, and ODBC.

You must register nicknames for these objects. Data source objects are sometimes referred to as *remote objects*.

When you submit a distributed request to the federated server, the request references a data source object by its nickname. Nicknames are mapped to specific object names at the data source. The mappings eliminate the need to qualify the nicknames by data source names. The location of the data source objects are transparent to the client application or end user. Nicknames are not alternative names for data source objects. They are pointers by which the federated server references these objects.

For example, if you define the nickname *DEPT* to represent an Informix database table called *NFX1.PERSON.DEPT*, the statement SELECT * FROM *DEPT* is allowed from the federated server. However, the statement, SELECT * FROM *NFX1.PERSON.DEPT* is not allowed.

When you register a nickname, metadata information about that nickname is stored in the federated database system catalog. For a relational data source object, catalog data from the remote server is retrieved and stored in the federated database system catalog. For nonrelational data sources, the way that the data source information is stored in the federated database system catalog varies from data source to data source. The information might be retrieved from the remote server, or you might have to include this information in the CREATE NICKNAME statement.

The SQL Compiler uses this metadata information to facilitate access to the data source object. For example, when a nickname is registered for a table with an index, the metadata information related to the index is stored in the federated database system catalog. The SQL Compiler uses the index metadata information, such as the name of each column in the index key, when you query the nickname.

Procedure:

You can register a nickname from the DB2 Control Center or the DB2 command line:
- To do this task from the DB2 Control Center, use the Federated Objects wizard or right-click the **Nicknames** folder and click **Create**. Use the Discover tool to identify the objects that you want to create nicknames for.
- To do this task from the DB2 command line, use the CREATE NICKNAME statement.

You can define more than one nickname for the same data source object.

Related tasks:

- “Registering nicknames for table-structured files” on page 359
- “Registering nicknames for Documentum data sources” on page 176
- “Registering nicknames for Excel data sources” on page 221
- “Registering nicknames for BLAST data sources” on page 107
- “Registering nicknames for XML data sources” on page 424
- “Registering nicknames for Entrez data sources” on page 199
- “Registering nicknames for Teradata tables and views” on page 377
- “Registering nicknames for BioRS data sources” on page 74
- “Registering nicknames for DB2 tables and views” on page 166
- “Registering nicknames for Extended Search data sources” on page 239
- “Registering nicknames for HMMER data sources” on page 265
- “Registering nicknames for Informix tables, views, and synonyms” on page 286
- “Registering nicknames for Microsoft SQL Server tables and views” on page 302
- “Registering nicknames for ODBC data source tables and views” on page 314
- “Registering nicknames for Oracle tables and views” on page 338
- “Registering nicknames for Sybase tables and views” on page 352
- “Registering nicknames for business application data sources” on page 129
- “Registering nicknames for Web services data sources” on page 390

Related reference:
- “Federated object naming rules” on page 21

Creating a nickname on a nickname

Occasionally, you might need to create a nickname on a nickname.

Procedure:

Suppose that you have a federated server using AIX® and a federated server using Windows. You want to access an Excel spreadsheet from both federated servers. However, the Excel wrapper is only supported on federated servers that use Windows. To access the Excel spreadsheet from the AIX federated server, use these steps:
1. On the Windows federated server, install DB2 Information Integrator.
2. Configure the Windows federated server to access Excel data sources.
3. On the Windows federated server, create a nickname for the Excel spreadsheet.
4. On the AIX federated server, install DB2 Information Integrator.
5. Configure the AIX federated server to access DB2 family data sources.
6. On the AIX federated server, create a nickname for the Excel nickname on the Windows federated server.

Specifying nickname columns for a nonrelational data source

For some nonrelational data sources, you must define a list of columns when you register a nickname. Each column that you specify is mapped to a particular field, column, or element in the data source object.

The wrapper for some nonrelational data sources requires a set of fixed input and output columns. The fixed columns are automatically defined when you register the nickname and are added to the federate database system catalog.

Procedure:

To define a list of columns when you register a nickname, you specify the column name and data type. You might also specify an option on the nickname column.

Related tasks:
- "Registering nicknames for a data source" on page 63

Optional configuration steps

There are several optional steps that you might need to take when you configure the federated server to access data sources.

Index specifications:

Define an index specification for objects that did not have an index. For example, you would create an index specification when a table acquires a new index or if the data source object (such as a view) typically does not have and index.

Data type mappings:

You can specify alternative data type mappings for only relational data sources.

Specify alternative data type mappings, instead of using the default data type mappings. You can specify a mapping that is used only for a specific data source object, such as a specific table within a database.

Function mappings:

You can specify function mappings for only relational data sources.

Define alternative function mappings, instead of using the default function mappings. This is especially useful when you want to force DB2® to use a user-defined function at the data source.

Related concepts:
- "Data type mappings in a federated system" in the *Federated Systems Guide*
- "Function mappings in a federated system" in the *Federated Systems Guide*
- "Index specifications in a federated system" in the *Federated Systems Guide*
- "Fast track to configuring your data sources" on page 55

Chapter 6. Configuring access to BioRS data sources

This chapter explains how to configure your federated server to access data that is stored in BioRS data sources. You can configure access to BioRS data sources by using the DB2 Control Center or by issuing SQL statements.

This chapter:
- Explains what BioRS is
- Lists the tasks that you need to perform
- Contains examples of the SQL statements that you need
- Lists the error messages associated with the BioRS wrapper

What is BioRS?

BioRS is a query and retrieval system that is developed by Biomax Informatics. You can use BioRS to retrieve information from multiple data sources, including flat files and relational databases. You usually download public data, such as SwissProt and GenBank, as flat files into your BioRS system. BioRS can integrate public data sources and proprietary data sources (for example, private databases that are maintained by your organization) into a common environment.

After a data source is integrated into the BioRS system, it is referred to as a *databank*. The elements that are contained in each databank entry are collectively referred to as a *schema*. Elements of a databank that are indexed can be used in the BIORS.CONTAINS, BIORS.CONTAINS_GE, and BIORS.CONTAINS_LE functions. The BioRS functions are specified in the WHERE clause of the SELECT statement. Elements that are not indexed can be referenced in the SELECT list and in other predicates in the WHERE clause. Elements that are not indexed are processed by the federated server.

You can establish relationships between entries in databanks, so that you can link databanks together in the BioRS system.

BioRS databanks can have a parent-child relationship (databanks can be nested). In such a relationship, the child databank contains a Reference data type element called PARENT. The PARENT element refers to the _ID_ element of the parent databank. Other than the presence of this predefined PARENT element, nested databanks contain the same data as unnested databanks.

BioRS provides a Web-based interface that enables users to run queries on the data in BioRS databanks. The BioRS wrapper uses the same application programming interfaces (APIs) as the BioRS Web-based interface to run queries.

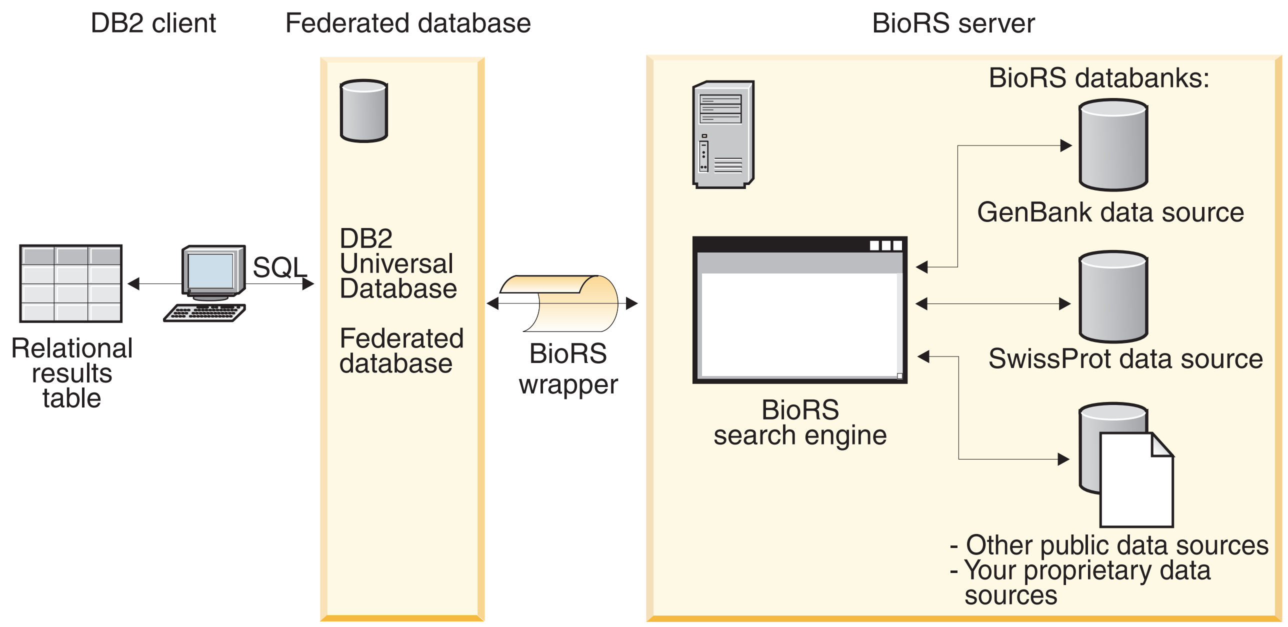


Figure 6. How the BioRS wrapper works

From the client, users or applications submit a query using SQL statements. Then, the query is sent to your federated system where the BioRS wrapper is installed. Depending on how the query is constructed, both the federated servers and your BioRS server might be used to process the query. The BioRS server can be on a different computer from the federated system. Authentication information must be provided by the federated system to the BioRS server for each query. This information can be either a user ID and password combination, or an unauthenticated indication (usually a guest account).

The BioRS wrapper works with BioRS Version 5.0.14.

For detailed information about the BioRS product, see the Biomax Web site at: http://www.biomax.com.

Related tasks:

- "Adding BioRS data sources to a federated server" on page 68

Related reference:

- "BioRS wrapper - Example queries" on page 81

Adding BioRS to a federated server

Adding BioRS data sources to a federated server

To configure the federated server to access BioRS data sources, you must provide the federated server with information about the data sources and objects that you want to access.

You can configure the federated server to access BioRS data sources by using the DB2 Control Center or the DB2 command line. The DB2 Control Center includes a wizard to guide you through the steps required to configure the federated server.

Prerequisites:

- DB2 Information Integrator must be installed on a server that will act as the federated server
- A federated database must exist on the federated server

Procedure:

To add BioRS data sources to a federated server:

1. Register custom functions for the BioRS wrapper.
2. Register the BioRS wrapper.
3. Register the BioRS server definition.
4. Optional: Create user mappings.
5. Register nicknames for BioRS databanks.

Related concepts:

- "DB2 Information Integrator installation process - overview" in the *IBM DB2 Information Integrator Installation Guide for Linux, UNIX, and Windows*
- "Fast track to configuring your data sources" on page 55

Related tasks:

- "Creating a federated database" on page 51
- "Registering the custom functions for the BioRS wrapper" on page 69
- "Registering the BioRS wrapper" on page 71
- "Registering the server definition for a BioRS data source" on page 72
- "Registering user mappings for BioRS data sources" on page 73
- "Registering nicknames for BioRS data sources" on page 74
- "Checking the setup of the federated server" on page 37

Related reference:

- "Supported operating systems for DB2 Information Integrator (32-bit)" in the *IBM DB2 Information Integrator Installation Guide for Linux, UNIX, and Windows*
- "Supported operating systems for DB2 Information Integrator (64-bit)" in the *IBM DB2 Information Integrator Installation Guide for Linux, UNIX, and Windows*

Registering the custom functions for the BioRS wrapper

Registering custom functions for the BioRS wrapper is part of the larger task of adding BioRS data sources to a federated server.

The custom functions for the BioRS wrapper are:

- biors.contains
- biors.contains_le
- biors.contains_ge
- biors.search_term

Prerequisites:

- All of the custom functions for the BioRS wrapper must be registered with the schema name `biors`.
- You must register each custom function once for each federated database where the BioRS wrapper is installed.

Procedure:

To register custom functions, issue the CREATE FUNCTION statement with the AS TEMPLATE DETERMINISTIC NO EXTERNAL ACTION keywords.

The fully qualified name of each function is `biors.function_name`.

The following example registers one version of the CONTAINS function:

```
CREATE FUNCTION biors.contains (varchar(), varchar())
   RETURNS INTEGER AS TEMPLATE
   DETERMINISTIC NO EXTERNAL ACTION;
```

To register the custom functions, use the sample file `create_function_mappings.ddl`. The sample file is in the `sqllib/samples/lifesci/biors` directory. The sample files contains definitions for each of the custom function. You can run this DDL file to register the custom functions on each federated database where the BioRS wrapper is installed.

The next task in this sequence of tasks is registering the BioRS wrapper.

Related tasks:
- "Registering the BioRS wrapper" on page 71

Related reference:
- "CREATE FUNCTION (Sourced or Template) statement" in the *SQL Reference, Volume 2*
- "Custom functions and BioRS queries" on page 77
- "BioRS wrapper - Example queries" on page 81
- "Custom function table - BioRS wrapper" on page 70

Custom function table - BioRS wrapper

You use the CREATE FUNCTION statement to register the BioRS custom functions.

The following table lists the four BioRS custom functions with examples of the data types that you can specify when you register the functions.

The first data type that is specified in the function is the indexed column. The second data type that is specified in the function is the search term.

Table 12. Custom functions for the BioRS wrapper

Function	Description
biors.contains (varchar(), varchar()) biors.contains (varchar(), char()) biors.contains (varchar(), date) biors.contains (varchar(), timestamp)	Searches an indexed column for values that are equal (according to the BioRS query semantics) to the value that you specify. The first argument must be a reference to the indexed column and the second argument is the value that you specify.

Table 12. Custom functions for the BioRS wrapper (continued)

Function	Description
biors.contains_LE (varchar(), varchar()) biors.contains_LE (varchar(), smallint) biors.contains_LE (varchar(), bigint) biors.contains_LE (varchar(), decimal) biors.contains_LE (varchar(), double) biors.contains_LE (varchar(), real)	Searches an indexed column for values that are less than or equal (according to the BioRS query semantics) to the value that you specify. The first argument must be a reference to the indexed column and the second argument is the value that you specify.
biors.contains_GE (char(), char()) biors.contains_GE (char(), date) biors.contains_GE (char(), timestamp) biors.contains_GE (char(), integer) biors.contains_GE (char(), smallint) biors.contains_GE (clob(), date)	Searches an indexed column for values that are greater than or equal (according to the BioRS query semantics) to the value that you specify. The first argument must be a reference to the indexed column and the second argument is the value that you specify.
biors.search_term (varchar(), varchar()) biors.search_term (varchar(), char()) biors.search_term (char(), varchar()) biors.search_term (char(), char())	Passes a BioRS search term to the BioRS search engine.

Related tasks:
- "Registering the custom functions for the BioRS wrapper" on page 69

Registering the BioRS wrapper

Registering the BioRS wrapper is part of the larger task of adding BioRS data sources to a federated server.

You must register a wrapper to access BioRS data sources. Wrappers are used by federated servers to communicate with and retrieve data from data sources. Wrappers are implemented as a set of library files.

Procedure:

To register a wrapper, issue the CREATE WRAPPER statement with the name of the wrapper and the name of the wrapper library file.

For example, to register a wrapper with the name `biors_wrapper` on the federated server that uses the AIX operating system, issue the following statement:

```
CREATE WRAPPER biors_wrapper LIBRARY 'libdb2lsbiors.a';
```

The name of the wrapper library file that you specify depends on the operating system of the federated server. See the list of BioRS wrapper library files for the correct name to specify in the CREATE WRAPPER statement.

The next task in this sequence of tasks is registering the server definitions for the BiorRS wrapper.

Related reference:
- "BioRS wrapper library files" on page 72
- "CREATE WRAPPER statement" in the *SQL Reference, Volume 2*

BioRS wrapper library files

The following table lists the directory paths and library file names for the BioRS wrapper.

When you install DB2 Information Integrator, 3 library files are added to the directory path listed in the table. For example, if the federated server is running on AIX, the wrapper library files added to the directory path are `libdb2lsbiors.a`, `libdb2lsbiorsF.a`, and `libdb2lsbiorsU.a`.

When you register a wrapper, specify only the library file name that is listed in the table.

Table 13. BioRS wrapper library locations and file names

Operating system	Directory path	Wrapper library file
AIX	/usr/opt/db2_08_01/lib/	libdb2lsbiors.a
Linux	/opt/IBM/db2/V8.1/lib	libdb2lsbiors.so
Solaris	/opt/IBM/db2/V8.1/lib	libdb2lsbiors.so
Windows	%DB2PATH%\bin	db2lsbiors.dll

%DB2PATH% is the environment variable that is used to specify the directory path where DB2 Information Integrator is installed on Windows. The default Windows directory path is `C:\Program Files\IBM\SQLLIB`.

Related reference:

- "Wrapper library files" on page 39

Registering the server definition for a BioRS data source

Registering the server definition for a BioRS data source is part of the larger task of adding BioRS to a federated system. After you register the wrapper, you must register a corresponding server definition.

Procedure:

To register the BioRS server definition in the federated database, issue the CREATE SERVER statement.

For example:

```
CREATE SERVER brs_server WRAPPER wrap_biors OPTIONS(NODE 'biors_server2.com');
```

The next task in this sequence of tasks is registering user mappings for BioRS data sources.

Related tasks:

- "Registering nicknames for BioRS data sources" on page 74

Related reference:

- "CREATE SERVER statement options - BioRS wrapper" on page 535

Registering user mappings for BioRS data sources

Registering user mappings is part of the larger task of adding BioRS to a federated system.

You might not need to create user mappings, depending on the account access method or methods that are used in your BioRS system.

- If your BioRS server is configured for guest access for all user accounts, you do not need to create user mappings in DB2 Information Integrator.
- If your BioRS server is configured to authenticate user accounts with IDs and passwords, you must create user mappings in your federated database for the accounts that must use the BioRS wrapper.
- If your BioRS server is configured to use a mixture of guest and authenticated user accounts, you must create user mappings for the authenticated user accounts in your federated database for the accounts that must use the BioRS wrapper.

User mappings provide a way to authenticate the access of users or applications that query a BioRS data source with the BioRS wrapper. If a user or application submits an SQL query to a registered BioRS nickname, and no user mappings are defined for that user or application, the BioRS wrapper uses a default user ID and password in an attempt to retrieve data from the remote BioRS server. If a databank that is being queried requires authentication, an error message might be returned.

To ensure that the correct user ID and password get passed to the BioRS server, create user mappings in your federated database for users who are authorized to search BioRS data sources. When you create a user mapping, the password is stored in an encrypted format in a federated database system catalog table.

Procedure:

To register BioRS user mappings, use the CREATE USER MAPPING statement.

For example, the following CREATE USER MAPPING statement maps user `Charlie` to user `Charlene` on the `Biors_Server1` server.

```
CREATE USER MAPPING FOR Charlie SERVER Biors_Server1
OPTIONS(REMOTE_AUTHID 'Charlene', REMOTE_PASSWORD 'Charlene_pw');
```

You can also define your own user mapping. In the following example, USER is a keyword that identifies the current user, not a username of USER.

```
CREATE USER MAPPING FOR USER SERVER Biors_Server1
OPTIONS(REMOTE_AUTHID 'Yudong', REMOTE_PASSWORD 'Yudong_pw')
```

The next task in this sequence of tasks is registering nicknames for the BioRS data sources.

Related tasks:

- "Registering nicknames for BioRS data sources" on page 74

Related reference:

- "CREATE USER MAPPING statement" in the *SQL Reference, Volume 2*
- "CREATE USER MAPPING statement options - BioRS wrapper" on page 536

Registering nicknames for BioRS data sources

Registering nicknames for BioRS data sources is part of the larger task of adding BioRS to a federated server.

After you register a server definition, you must register a nickname for each BioRS data source that you want to access. When you refer to a BioRS data source in a query, you use the nickname.

After a data source has been integrated into the BioRS system, it is referred to as a *databank* in BioRS. Databanks in BioRS equate to nicknames in a federated system.

Prerequisites:

- If a BioRS databank name does not conform to the syntax required by the CREATE NICKNAME statement, you must use the REMOTE_OBJECT nickname option when you register the nickname.
- If a BioRS element name does not conform to the syntax required by the CREATE NICKNAME statement, you must use the ELEMENT_NAME column option when you register the nickname.

Restrictions:

Do not use the BioRS AllText element as the first column for a nickname. You can use the BioRS AllText element in any other column position (for example, as the second column or as the third column).

Procedure:

To register a BioRS nickname, use the CREATE NICKNAME statement.

When you create a BioRS nickname, you define a list of nickname columns. The specified nickname columns must correspond to elements of a specific BioRS databank format. BioRS defines five possible data types for elements: Text, Number, Date, Author, and Reference. The BioRS data types can be mapped only to the DB2 data types CHAR, CLOB, or VARCHAR.

The simplest way to register a nickname for a BioRS databank is to give the nickname the same name as the BioRS databank. For example:

```
CREATE NICKNAME SwissProt
 (ID VARCHAR(32) OPTIONS (ELEMENT_NAME '_ID_'),
 ALLTEXT VARCHAR(128),
 ENTRYDATE VARCHAR (64))
FOR SERVER brs_server;
```

The BioRS databank `SwissProt` is the name of the nickname.

Using this simple CREATE NICKNAME syntax limits you to one family of nicknames for each DB2 schema. For example, you have two databanks that have a parent-child relationship. The databanks are SWISSPROT and SPFEAT. These databanks form a family. If you use the default syntax for the CREATE NICKNAME statement, you will have one nickname (SWISSPROT) for the SWISSPROT databank and one nickname (SPFEAT) for the SPFEAT databank. To have more than one nickname for SWISSPROT in the schema, you must use the REMOTE_OBJECT option.

The REMOTE_OBJECT nickname option specifies the name of the BioRS databank that is associated with the nickname. The name that you specify in the REMOTE_OBJECT option determines the schema and the BioRS databank for the nickname. The REMOTE_OBJECT option also specifies the relationship of the nickname to other nicknames.

The following example shows the same set of nickname characteristics as the previous example, but changes the nickname name, and uses the REMOTE_OBJECT option to specify the BioRS databank for which the nickname is being defined:

```
CREATE NICKNAME NewSP
   (ID VARCHAR(32) OPTIONS (ELEMENT_NAME '_ID_'),
   ALLTEXT VARCHAR(128),
   ENTRYDATE VARCHAR (64))
FOR SERVER brs_server
   OPTIONS (REMOTE_OBJECT 'SwissProt');
```

Repeat this step for each BioRS databank that you want to create a nickname for.

There are no further steps in this sequence of tasks.

Related concepts:
- "BioRS statistical information" on page 87

Related tasks:
- "Updating BioRS nickname cardinality statistics" on page 88
- "Specifying nickname columns for a nonrelational data source" on page 65

Related reference:
- "The BioRS AllText element" on page 81
- "CREATE NICKNAME statement - Examples for BioRS wrapper" on page 75
- "CREATE NICKNAME statement syntax - BioRS wrapper" on page 536

CREATE NICKNAME statement - Examples for BioRS wrapper

This topic provides examples that show you how to use the CREATE NICKNAME statement to register nicknames for the BioRS wrapper.

Example 1:

The following example shows how to create a nickname for a remote BioRS databank that does not conform to DB2 Information Integrator syntax:

```
CREATE NICKNAME SwissFT
 (ID VARCHAR(32) OPTIONS (ELEMENT_NAME '_ID_'),
 ALLTEXT VARCHAR (128),
 ENTRYDATE VARCHAR (64),
 FtLength VARCHAR (16),
   FOR SERVER biors1
 OPTIONS (REMOTE_OBJECT 'SwissProt.Features');
```

The name of this nickname is SwissFT. The table columns are ID, ALLTEXT, ENTRYDATE, and FtLength. The ELEMENT_NAME column option is specified for the ID column. You must specify the ELEMENT_NAME option when the name of a BioRS element does not conform to valid DB2 federated syntax for column names. In this example, the BioRS element _ID_ conforms to DB2 federated syntax, but `_ID_` is a potentially confusing name for DB2 Information Integrator users. The name ID is simple and easy to understand. In general, use the ELEMENT_NAME option under the following circumstances:

- When a BioRS element name does not conform to valid DB2 federated syntax
- When the case sensitivity of a BioRS element name does not conform to your established DB2 federated system standards
- When the name of a BioRS element might not be obvious to DB2 Information Integrator users

Additionally, the REMOTE_OBJECT option is used to specify the name of the BioRS databank to which the nickname equates. You must specify the REMOTE_OBJECT option when the name of a BioRS databank does not conform to valid DB2 federated syntax. In this example, the databank name "SwissProt.Features" does not conform to valid DB2 federated syntax. In general, use the REMOTE_OBJECT option under the following circumstances:

- When the case sensitivity of BioRS databank names does not conform to your established DB2 federated system standards
- When the BioRS databank name does not conform to valid DB2 federated syntax
- When the name of a BioRS databank might not be obvious to DB2 Information Integrator users

Example 2:

The following example shows how to create a nickname for a table that uses a BioRS databank that is linked to another BioRS databank:

```
CREATE NICKNAME SwissFT2
 (ID VARCHAR(32) OPTIONS (ELEMENT_NAME '_ID_'),
 ALLTEXT VARCHAR (1200),
 FtKey VARCHAR (32),
 FtLength VARCHAR (64),
   FtDescription VARCHAR (128),
   Parent VARCHAR (32) OPTIONS (REFERENCED_OBJECT 'SwissProt'))
 FOR SERVER biors1
 OPTIONS (REMOTE_OBJECT 'SwissProt.Features');
```

The name of this nickname is SwissFT2. The table columns are ID, ALLTEXT, FtKey, FtLength, FtDescription, and Parent. The ELEMENT_NAME column option is specified for the ID column. The REMOTE_OBJECT option is used to specify the name of the BioRS databank to which the nickname corresponds.

Additionally, the Parent column uses the REFERENCED_OBJECT option. You must specify this option for columns that correspond to BioRS Reference data type elements. The REFERENCED_OBJECT option specifies the name of the BioRS databank to which the column refers. In this case, the Parent element refers to the BioRS SwissProt databank.

Related tasks:

- "Registering nicknames for BioRS data sources" on page 74

Related reference:

- "CREATE NICKNAME statement syntax - BioRS wrapper" on page 536

Queries and custom functions for BioRS data sources

Custom functions and BioRS queries

The federated environment uses two query engines. For the BioRS wrapper, these query engines are DB2 Universal Database and BioRS. You can specify that predicates get pushed down to the BioRS engine by using the four BioRS custom functions, which are:

- BIORS.CONTAINS
- BIORS.CONTAINS_LE
- BIORS.CONTAINS_GE
- BIORS.SEARCH_TERM

These four custom functions are registered in the BioRS schema. You must use the BioRS schema to refer to the functions.

The custom functions BIORS.CONTAINS, BIORS.CONTAINS_LE and BIORS.CONTAINS_GE require a search term column argument and a query text argument. The following example shows a BIORS.CONTAINS statement:

```
BIORS.CONTAINS (<search term column>,<query term>)
```

The value of the search term column argument must refer to an indexed BioRS column. The use of a non-indexed column produces the error message SQL30090N (″Operation invalid for application execution environment″).

The value of the query term argument can be only a literal, a host variable, or a column reference. You cannot use arithmetic or string concatenation. Also, the value of the query term argument cannot be NULL, even if the search term column that is used is defined as allowing null values.

The case of the query term argument does not matter.

The valid data types and formats of the query term argument depend on the BioRS data type of the search term column that is used. BioRS defines five possible data types: Text, Author, Date, Number, and Reference. The BioRS data types and the valid function query terms for each data type are listed in Table 14.

Table 14. BioRS data types and valid custom function query terms

Data type of search term column	Valid query term	Format
Text	VARCHAR() or CHAR()	BioRS text term, including wildcards.
Author	VARCHAR() or CHAR()	BioRS author reference in the form ″<last>, <init>″. ″<last>″ is the author’s last name. ″<init>″ is the author’s initials, without periods. White space between the comma and initials is accepted. Alternatively, <last> can be specified alone, without the comma or initials.
Date	VARCHAR(), CHAR(), DATE, or TIMESTAMP	If a character string, DB2 format date, yyyy/mm/dd.

Table 14. BioRS data types and valid custom function query terms (continued)

Data type of search term column	Valid query term	Format
Number	VARCHAR() or CHAR(), INTEGER, SMALLINT, BIGINT REAL, DOUBLE, DECIMAL	DB2 format numbers.
Reference	VARCHAR() or CHAR()	BioRS text term.

All other combinations of BioRS data type search term columns and query term arguments produce the error message SQL30090N ("Operation invalid for application execution environment"). You can use only the combinations shown in Table 14 on page 77.

The query term argument for Text, Author, and Reference data type search term columns must match a BioRS query language pattern. In BioRS, query term arguments can consist of alphanumeric strings and wildcards. The BIORS.CONTAINS function supports two wildcards: **?** (question mark) and * (asterisk).

The **?** wildcard matches a single character. For example, the predicate `BioRS.CONTAINS (description, 'bacteri?')=1` matches the term bacteria but not the term bacterial.

The * wildcard character matches zero or more characters. For example, the predicate `BioRS.CONTAINS (description, 'bacteri*')=1` matches the terms bacteri, bacteria, and bacterial.

For detailed information about BioRS query language patterns, see your BioRS documentation.

The BIORS.CONTAINS function can be specified for all BioRS column types.

The BIORS.CONTAINS_GE and BIORS.CONTAINS_LE custom functions only can be specified for columns whose underlying BioRS data type is Number or Date. The BIORS.CONTAINS_GE function selects rows where the column contains a value that is greater than or equal to the value that is represented by the query term argument. The BIORS.CONTAINS_LE function selects rows where the column contains a value that is less than or equal to the value that is represented by the query term argument.

The BIORS.CONTAINS, BIORS.CONTAINS_GE, and BIORS.CONTAINS_LE functions return an integer result. When any of the three CONTAINS functions are used in a predicate, the return value must be compared to the value 1 using the = or <> operators. For example:

```
SELECT * FROM s.MySP WHERE BIORS.CONTAINS (s.AllText, 'muscus') = 1;
```

An expression of the form `NOT (BioRS.Contains (col,value) = 1)` is equivalent to `BioRS.CONTAINS (col,value) <> 1`.

You can run queries that might not otherwise be possible by issuing the BIORS.SEARCH_TERM function. You can use this function to specify a search term using the BioRS format. The BIORS.SEARCH_TERM function requires two arguments. The first argument is a reference to the _ID_ column of the nickname to which the term is to be applied. The second argument is a character string that contains the term without a databank name.

The following example selects all columns for entries in the MyEMBL databank where the SeqLength element contains a value greater than or equal to 100.

```
SELECT * FROM MyEMBL s WHERE
 BIORS.SEARCH_TERM (s.ID, '[SeqLength GREATER number:100;]') = 1;
```

The following example selects the MolWeight column from the Swiss nickname where the value of the MolWeight element is greater than or equal to 100368.

```
SELECT s.molweight FROM Swiss s WHERE
   BIORS.SEARCH_TERM (s.ID, '[MolWeight GREATER number:100368;]') = 1;
```

Related concepts:

- “Pushdown analysis” in the *Federated Systems Guide*
- “Guidelines for optimizing BioRS wrapper performance” on page 86
- “Equijoin predicates for the BioRS wrapper” on page 79

Related tasks:

- “Registering the custom functions for the BioRS wrapper” on page 69

Related reference:

- “BioRS wrapper - Example queries” on page 81
- “Custom function table - BioRS wrapper” on page 70

Equijoin predicates for the BioRS wrapper

You must specify predicates for the BioRS engine when you use the BioRS custom functions, with one exception. The exception is when you perform equijoin operations during a query. A *join* operation involves retrieving data from two or more tables based on matching column values. An *equijoin* is a join operation in which the join condition has the form expression = expression. For BioRS queries, equijoin terms must contain the _ID_ element of one databank and a Reference type element of another databank.

Example:

This example shows sample nickname definitions and an equijoin query that uses the sample nicknames.

You want to query two BioRS databanks, SwissProt and SwissProt.features. The SwissProt.features databank is a child of the SwissProt databank, and contains an element called Parent. The Parent element contains references to entries that are identified by the _ID_ element of SwissProt. You register two nickname definitions for the two databanks.

Nickname definition 1:

```
CREATE NICKNAME tc600sprot (
   ID             VARCHAR (32) OPTIONS (ELEMENT_NAME '_ID_'),
   AllText        VARCHAR (128),
   EntryDate      VARCHAR (128),
   Update         VARCHAR (128),
   Description    VARCHAR (1200),
   Crossreference VARCHAR (32),
   Authors        VARCHAR (256),
   Journal        VARCHAR (256),
   JournalIssue   VARCHAR (64) OPTIONS (IS_INDEXED 'N'),
   PublicationYear VARCHAR (1024),
   Gene            VARCHAR (20) OPTIONS (IS_INDEXED 'Y'),
   Remarks         VARCHAR (1200),
   RemarkType      CHAR (20),
   CatalyticActivity VARCHAR (20),
   CoFactor        VARCHAR (64),
   Disease         VARCHAR (128),
   Function        VARCHAR (128),
   Pathway         VARCHAR (128),
   Similarity      VARCHAR (128),
   Complex         VARCHAR (64),
   FtKey           VARCHAR (32),
   FtDescription   VARCHAR (128),
   FtLength        VARCHAR (256),
   MolWeight       VARCHAR (64),
   ProteinLen      VARCHAR (32) OPTIONS (ELEMENT_NAME 'Protein_length'),
   Sequence        CLOB,
   AccNumber       VARCHAR (32),
   Taxonomy        VARCHAR (128),
   Organelle       VARCHAR (128),
   Organism        VARCHAR (128),
   Keywords        VARCHAR (1200),
   Localization    VARCHAR (128),
   FtKey_count     VARCHAR (32)) FOR SERVER biors_server_600
      OPTIONS (REMOTE_OBJECT 'SwissProt');
```

Nickname definition 2:

```
CREATE NICKNAME tc600feat (
  ID     VARCHAR (32) OPTIONS (ELEMENT_NAME '_ID_'),
  AllText VARCHAR (1200),
  FtKey   VARCHAR (32),
  FtLength VARCHAR (64),
  FtDescription VARCHAR (128),
  Parent        VARCHAR (32) OPTIONS (REFERENCED_OBJECT 'SwissProt'))
    FOR SERVER biors_server_600 OPTIONS (REMOTE_OBJECT 'SwissProt.features');
```

The following query references both of these nicknames in an equijoin:

```
SELECT s.ID, f.ID, f.FtKey FROM tc600sprot s, tc600feat f
   WHERE BioRS.CONTAINS (s.AllText, 'anopheles') = 1
    AND BioRS.CONTAINS (s.PublicationYear, 1997) = 1
 AND BioRS.CONTAINS (f.FtKey, 'signal') = 1
 AND f.Parent = s.ID;
```

In the previous query, two predicates are applied to the tc600sprot nickname (SwissProt databank). These two predicates filter the rows that contain the term anopheles and have a publication year of 1997. One predicate is applied to the tc600feat nickname (SwissProt.features databank), which filters those rows whose FtKey element contains the term signal. The two nicknames are joined using the term f.Parent = s.ID.

The final result set contains only the rows that meet these criteria, and where the features entries reference a matching entry in the SwissProt databank.

Related concepts:

- "Guidelines for optimizing BioRS wrapper performance" on page 86

Related reference:

- "Custom functions and BioRS queries" on page 77
- "BioRS wrapper - Example queries" on page 81

The BioRS AllText element

Every databank in the BioRS system contains an element called AllText. The AllText element is an indexed element that BioRS automatically creates for all databanks.

By using the AllText element, you can search on all of the text in an entry, not just on specific indexed elements. For example, searching on the term `muscus` can return entries where the word `muscus` appears in the title, abstract, description, or organism.

To use the AllText element in a DB2 Information Integrator query, you must map the AllText element to a nickname column. You map the AllText element to a nickname column when you specify columns in the CREATE NICKNAME statement. A nickname column that is mapped to the AllText element returns a NULL value in SELECT statements. When you specify a column as an AllText element, the column must not be the first column declared in a CREATE NICKNAME statement.

After the AllText element is properly mapped to a nickname column, you can use that nickname column in a CONTAINS custom function invocation.

Related tasks:

- “Registering nicknames for BioRS data sources” on page 74

Related reference:

- “BioRS wrapper - Example queries” on page 81

BioRS wrapper - Example queries

This topic provides several sample queries that use the nicknames swiss and swissft.

The nickname swiss was registered with the following CREATE NICKNAME statement:

```
CREATE NICKNAME swiss
  (
  ID                 CHAR (30) OPTIONS (ELEMENT_NAME '_ID_'),
  EntryDate          VARCHAR (15),
  Update             CLOB (15),
  Description        CLOB (15),
  Crossreference     CLOB (15),
  Authors            CLOB (15),
  Journal            VARCHAR (15),
  JournalIssue       VARCHAR (15),
  PublicationYear    CLOB (15),
  PublicationTitle   CLOB (15),
  Gene               CLOB (15),
  Remarks            CLOB (15),
  RemarkType         VARCHAR (15),
  CatalyticActivity  VARCHAR (15),
  CoFactor           VARCHAR (15),
  Disease            VARCHAR (15),
  Function           CLOB (15),
  Pathway            VARCHAR (15),
  Similarity         CLOB (15),
  Complex            VARCHAR (15),
  FtKey              VARCHAR (15),
  FtDescription      CLOB (15),
  FtLength           VARCHAR (15),
  MolWeight          CHAR (15),
  Protein_Length     VARCHAR (15),
  Sequence           CLOB (15),
  AccNumber          VARCHAR (15),
  Taxonomy           CLOB (15),
  Organelle          VARCHAR (15),
  Organism           VARCHAR (15),
  Keywords           VARCHAR (15),
  Localization       VARCHAR (15),
  FtKey_count        VARCHAR (15),
  AllText            CLOB (15)
  )
    FOR SERVER biors_server
      OPTIONS (REMOTE_OBJECT 'swissprot');
```

The nickname swissft was registered with the following CREATE NICKNAME statement:

```
CREATE NICKNAME swissft
  (
  ID                VARCHAR (30) OPTIONS (ELEMENT_NAME '_ID_'),
  FtKey             VARCHAR (15),
  FtLength          VARCHAR (15),
  FtDescription     VARCHAR (15),
  Parent            VARCHAR (30) OPTIONS (REFERENCED_OBJECT 'swissprot'),
  AllText           CLOB (15)
  )
    FOR SERVER biors_server
      OPTIONS (REMOTE_OBJECT 'swissprot.features');
```

The queries and results in Table 15 illustrate how you can structure your queries to optimize the workload between the federated system and the BioRS server.

Table 15. Samples of different queries that produce identical results

Query	Result

select s.id from Swiss s where biors.CONTAINS(s.id, ’100K_RAT’) = 1 fetch first 3 rows only	`ID`
	 `---------------`
	 `100K_RAT`
	
	 `1 record(s) selected.`

select s.id from Swiss s where s.id LIKE ’%100K_RAT%’ fetch first 3 rows only	`ID`
	 `---------------`
	 `100K_RAT`
	
	 `1 record(s) selected.`

Both of the queries in Table 15 produce the same results. However, the first query will run much faster than the second query. The first query uses the BIORS.CONTAINS function to specify the input predicate. As a result, BioRS selects the data in the swissprot databank, then passes the selected data to DB2 Information Integrator. In the second query, the LIKE input predicate is specified directly on the Swiss nickname. As a result, BioRS transfers the entire swissprot databank to DB2 Information Integrator. After the databank contents are transferred, DB2 Information Integrator then selects the data.

The queries and results in Table 16 on page 83 show the use of wildcard characters in the BIORS.CONTAINS function. All of the query results in Table 16 on page 83 are identical, even though different wildcard characters are used.

Table 16. Sample queries that use wildcards in the BIORS.CONTAINS function

Query	Result

select s.crossreference from Swiss s where biors.CONTAINS(s.crossreference, ’MEDLINE’) = 1 fetch first 3 rows only	`CROSSREFERENCE`
	`---------------`
	`NCBI_TaxID=1011`
	`NCBI_TaxID=5875`
	`NCBI_TaxID=4081`
	
	`  3 record(s) selected.`

select s.crossreference from Swiss s where biors.CONTAINS(s.crossreference, ’?ED?IN?’) = 1 fetch first 3 rows only	`CROSSREFERENCE`
	`---------------`
	`NCBI_TaxID=1011`
	`NCBI_TaxID=5875`
	`NCBI_TaxID=4081`
	
	`  3 record(s) selected.`

select s.crossreference from Swiss s where biors.CONTAINS(s.crossreference, ’*D*N*’) = 1 fetch first 3 rows only	`CROSSREFERENCE`
	`---------------`
	`NCBI_TaxID=1011`
	`NCBI_TaxID=5875`
	`NCBI_TaxID=4081`
	
	`  3 record(s) selected`

The queries and results in Table 17 show how you can access information in BioRS Author data type elements with the BIORS.CONTAINS function.

The syntax of all of the queries in Table 17 is nearly identical. The only difference is the presence or absence of the first initial in the query term, and the amount of space between the first name and the last initial.

Table 17. Sample queries that access BioRS Author data type columns

Query	Result

select s.authors from Swiss s where biors.CONTAINS(s.authors, ’Mueller’) = 1 fetch first 3 rows only	`AUTHORS`
	`---------------`
	`Mueller D. Rehb`
	`Mayer K.F.X. Sc`
	`Zemmour J. Litt`
	
	`  3 record(s) selected.`

select s.authors from Swiss s where biors.CONTAINS(s.authors, ’Mueller,D’) = 1 fetch first 3 rows only	`AUTHORS`
	`---------------`
	
	`  0 record(s) selected.`

select s.authors from Swiss s where biors.CONTAINS(s.authors, ’Mueller ,D’) = 1 fetch first 3 rows only	`AUTHORS`
	`---------------`
	
	`  0 record(s) selected.`

select s.authors from Swiss s where biors.CONTAINS(s.authors, ’Mueller, D’) = 1 fetch first 3 rows only	`AUTHORS`
	`---------------`
	`Mueller D. Rehb`
	`Zou P.J. Borovo`
	`Davies J.D. Mue`
	
	`  3 record(s) selected.`

The queries and results in Table 18 illustrate how you can access information in BioRS Date type elements with the BIORS.CONTAINS function.

When a BioRS Date type field contains a sequence of dates, the results can contain extra information, as shown in the second example of Table 18. BioRS Numeric data type elements (Date and Number) can contain multiple values. Therefore, the results of queries run on BioRS Date or Number elements can also contain multiple values. Multiple values are always separated by spaces.

Table 18. Sample queries that access BioRS Date data type columns

Query	Result

select e.entrydate from embl e where biors.CONTAINS(e.entrydate, date('11/01/1997')) = 1 fetch first 3 rows only	`ENTRYDATE`
	`---------------`
	`01-NOV-1997`
	`01-NOV-1997`
	`01-NOV-1997`
	
	`  3 record(s) selected.`

select g.update from gen g where biors.CONTAINS(g.update, date('11/01/1997')) = 1 fetch first 3 rows only	`UPDATE`
	`---------------`
	`01-NOV-1997 11-`
	`01-NOV-1997 12-`
	`01-NOV-1997 06-`
	
	`  3 record(s) selected.`

The queries and results in Table 19 show how you can use the BIORS.CONTAINS_LE and the BIORS.CONTAINS_GE functions.

Table 19. Sample queries that use the BIORS.CONTAINS_LE and BIORS.CONTAINS_GE functions

Query	Result

select s.molweight from Swiss s where biors.CONTAINS_LE(s.molweight, 100368) = 1 fetch first 3 rows only	`MOLWEIGHT`
	`---------------`
	`100368`
	`10576`
	`8523`
	
	`  3 record(s) selected.`

select s.molweight from Swiss s where biors.CONTAINS_GE(s.molweight, 100368) = 1 fetch first 3 rows only	`MOLWEIGHT`
	`---------------`
	`100368`
	`103625`
	`132801`
	
	`  3 record(s) selected.`

select s.journalissue from Swiss s where biors.CONTAINS_GE(s.journalissue, 172) = 1 fetch first 3 rows only	`JOURNALISSUE`
	`---------------`
	`172 21`
	`242`
	`196`
	
	`  3 record(s) selected.`

The queries and results in Table 20 on page 85 show how you can use the BIORS.SEARCH_TERM function to specify a search term using the BioRS format.

Table 20. Sample queries that use the BIORS.SEARCH_TERM function

Query	Result

select s.publicationyear from Swiss s where biors.SEARCH_TERM (s.id, '[PublicationYear EQ number:1997;]')=1 fetch first 10 rows only	`PUBLICATIONYEAR`
	`---------------`
	`1997`
	`1997 2000`
	`1988 1991 1997`
	`1994 1997`
	`1997 1998`
	`1994 1995 1997`
	`1997 1999`
	`1997`
	`1994 1994 1995`
	`1993 1992 1997`
	
	`  10 record(s) selected.`

select s.molweight from Swiss s where biors.SEARCH_TERM (s.id, '[MolWeight EQ number:100368;]') = 1 fetch first 10 rows only	`MOLWEIGHT`
	`---------------`
	`100368`
	`100368`
	
	`  2 record(s) selected.`

select s.molweight from Swiss s where biors.SEARCH_TERM (s.id, '[MolWeight GREATER number:100368;]') = 1 fetch first 10 rows only	`MOLWEIGHT`
	`---------------`
	`100368`
	`103625`
	`132801`
	`194328`
	`130277`
	`287022`
	`289130`
	`135502`
	`112715`
	`112599`
	
	`  10 record(s) selected.`

The following query shows how to use relational predicates to form an equijoin between two databanks that have a parent-child relationship:

```
select s.id, f.id, f.parent from Swiss s, Swissft f
where (f.parent = s.id) fetch first 10 rows only
```

The query results are as follows:

```
ID                   ID                 PARENT
-------------------- ------------------ ------------------------------
100K_RAT             100K_RAT.1         swissprot:100K_RAT
100K_RAT             100K_RAT.2         swissprot:100K_RAT
100K_RAT             100K_RAT.3         swissprot:100K_RAT
100K_RAT             100K_RAT.4         swissprot:100K_RAT
100K_RAT             100K_RAT.5         swissprot:100K_RAT
100K_RAT             100K_RAT.6         swissprot:100K_RAT
100K_RAT             100K_RAT.7         swissprot:100K_RAT
100K_RAT             100K_RAT.8         swissprot:100K_RAT
100K_RAT             100K_RAT.9         swissprot:100K_RAT
104K_THEPA           104K_THEPA.1       swissprot:104K_THEPA

  10 record(s) selected.
```

In the previous query results, the 100K_RAT record is a parent to nine child records (100K_RAT.1 through 100K_RAT.9).

Related concepts:

- "Guidelines for optimizing BioRS wrapper performance" on page 86
- "Equijoin predicates for the BioRS wrapper" on page 79

Related reference:

- "Custom functions and BioRS queries" on page 77
- "CREATE NICKNAME statement - Examples for BioRS wrapper" on page 75
- "CREATE NICKNAME statement syntax - BioRS wrapper" on page 536

Optimizing BioRS wrapper performance

Guidelines for optimizing BioRS wrapper performance

This topic provides guidelines on how to optimize the performance of queries when you use the BioRS wrapper.

Minimize the amount of data that is transferred between search engines.

The federated environment uses two query engines. For the BioRS wrapper, these query engines are DB2® Universal Database and BioRS. The DB2 engine processes predicates (relational operators, such as =, BETWEEN, LIKE, and <>) specified on nickname columns. The BioRS engine processes predicates specified using four custom functions for the BioRS wrapper.

To minimize the amount of data that is transferred between the two search engines, structure your queries so that data processing gets pushed down to the BioRS system whenever possible.

If you need to perform join operations in a query, take advantage of any parent-child relationships that already exist in BioRS databanks and perform equijoin operations whenever possible. Equijoin operations are processed in BioRS, which also minimizes the amount of data transferred between the DB2 and BioRS query engines.

Attention: Do not interrupt DB2 Information Integrator queries to BioRS (for example, using **Ctrl-D** or **Ctrl-Z** in the command line processor, or stopping an application program). Interrupting a query leaves ″dead″ processes running on the BioRS server. These ″dead″ processes will rapidly degrade both BioRS and DB2 Information Integrator system performance. If enough of these ″dead″ processes are running, unexpected errors can occur during DB2 Information Integrator query processing. For example, a valid query might return 0 rows, when rows are expected. In extreme situations, BioRS, DB2 Information Integrator, or both products can stop or abnormally end.

Maintain BioRS statistical information in the federated environment.

In a federated system, the federated database relies on catalog statistics for nicknamed objects to optimize query processing. Maintaining current statistics about the BioRS data sources is essential to optimize the performance of the BioRS wrapper. If the statistical data or structural characteristics for a remote object on which a nickname is defined have changed, you must update the corresponding nickname column cardinality statistics in your federated system.

To optimize BioRS wrapper performance, perform these updates in DB2 Information Integrator at regular intervals.

Related concepts:

- "Tuning query processing" in the *Federated Systems Guide*
- "Equijoin predicates for the BioRS wrapper" on page 79
- "BioRS statistical information" on page 87

Related reference:

- "Custom functions and BioRS queries" on page 77
- "BioRS wrapper - Example queries" on page 81

BioRS statistical information

In a federated system, the federated database relies on catalog statistics for objects with nicknames to optimize query processing. These statistics are retrieved from BioRS data sources when you create a nickname using the CREATE NICKNAME statement. The federated database verifies the presence of the object at the data source, and then attempts to gather existing data source statistical data. Information is read from the data source catalogs and put into the DB2® federated database system catalog on the federated server.

For BioRS data sources, critical statistical information includes:

- The cardinality of a nickname. For BioRS data sources, nickname cardinality is equivalent to the number of entries in the corresponding BioRS databank.
- The cardinality of the column that corresponds to the BioRS _ID_ element. The cardinality of this column must match the cardinality of the nickname in which the column is referenced.
- The cardinality of all columns that the BioRS wrapper might need to use.

You must maintain current statistics about the BioRS data sources to optimize the performance of the BioRS wrapper. If the statistical data or structural characteristics for a remote object on which a nickname is defined change, you must update the corresponding cardinality statistics in your federated system. The cardinality statistics are stored in the SYSSTAT.TABLES catalog view and in the SYSSTAT.COLUMNS catalog view.

You perform the following tasks to maintain BioRS cardinality statistics in your federated system:

1. Determine the cardinality statistics of the required nickname, if necessary.
2. Update the appropriate the cardinality statistics in the required catalog view or catalog views.

Related concepts:

- "Tuning query processing" in the *Federated Systems Guide*

Related tasks:

- "Determining BioRS databank cardinality statistics" on page 88
- "Updating BioRS nickname cardinality statistics" on page 88
- "Updating BioRS column cardinality statistics" on page 89
- "Updating BioRS _ID_ column cardinality" on page 90

Determining BioRS databank cardinality statistics

You must determine BioRS databank cardinality statistics before you can update nickname statistics or update the cardinality of the column that corresponds to the BioRS _ID_ element.

Procedure:

To determine cardinality statistics for a specific databank in BioRS, use the BioRS utility program admin_find or www_find.cgi. Specify the `-c` (cardinality) option. For more information about these two BioRS utility programs, see your BioRS documentation.

Related concepts:

- "BioRS statistical information" on page 87

Related tasks:

- "Updating BioRS nickname cardinality statistics" on page 88
- "Updating BioRS column cardinality statistics" on page 89
- "Updating BioRS _ID_ column cardinality" on page 90

Updating BioRS nickname cardinality statistics

You must update BioRS nickname cardinality statistics when the contents of a BioRS databank for which you create a nickname change significantly. Maintaining correct cardinality statistics for nicknames enables the optimizer and the BioRS wrapper to choose the best performing data access plan.

To update BioRS nickname cardinality statistics, you modify the SYSSTAT.TABLES catalog view with the correct cardinality number.

Prerequisites:

You must determine the cardinality number of the BioRS databank that corresponds to the nickname whose statistics you want to update.

Procedure:

Issue an UPDATE statement using the following syntax:

```
UPDATE sysstat.tables SET card=cardinality
    WHERE tabschema=nickname-schema
    AND tabname=nickname-name;
```

- *cardinality* is the BioRS databank cardinality number that corresponds to the nickname whose statistics you want to update.
- *nickname-schema* is the name of the schema that is associated with the nickname whose statistics you want to update.
- *nickname-name* is the name of the nickname whose statistics you want to update.

Related concepts:

- "BioRS statistical information" on page 87

Related tasks:

- "Determining BioRS databank cardinality statistics" on page 88

- "Updating BioRS column cardinality statistics" on page 89
- "Updating BioRS _ID_ column cardinality" on page 90

Updating BioRS column cardinality statistics

To update BioRS column cardinality statistics in your federated system, you must modify the SYSSTAT.COLUMNS catalog view.

Maintaining correct cardinality statistics for BioRS columns enables the optimizer and the BioRS wrapper to choose the best performing data access plan during query processing.

You can optionally update BioRS column cardinality statistics as part of the larger task of adding BioRS to a federated system. You can also update BioRS column cardinality statistics when you want to improve query performance for BioRS data sources.

Restrictions:

Do not use this procedure to update the cardinality statistics for columns that correspond to the BioRS _ID_ element. You must use a different procedure to update the cardinality statistics for columns that correspond to the BioRS _ID_ element.

Procedure:

To update BioRS column cardinality statistics, issue an UPDATE statement using the following syntax:

```
UPDATE sysstat.columns SET colcard=(SELECT COUNT(DISTINCT column-name)
    FROM nickname-schema.nickname-name)
    WHERE
    tabschema=nickname-schema
    AND tabname=nickname-name
    AND colname=column-name;
```

- *column-name* is the name of the column whose cardinality statistics you want to update.
- *nickname-schema* is the name of the schema that is associated with the nickname where the specified column is used.
- *nickname-name* is the name of the nickname where the specified column is used.

The query might take several minutes to run, because all entries for the databank that is specified in the nickname must be retrieved.

If a column can contain multiple values (for example, the PublicationYear element of the SwissProt database format), the calculation becomes too complex to use an SQL query. For such columns, you must manually calculate the cardinality value, and then update the SYSSTAT.COLUMNS catalog view. To calculate the cardinality value, divide the number of distinct values in the column by the average number of values per row. The calculated cardinality value cannot be greater than the cardinality of the table.

Example:

Suppose you have a nickname with three rows. The values of the PublicationYear column for these three rows are:

- 1997 1992 1985
- 1997 1992 1982
- 1992 1991 1990 1976 1974 1971

There are nine distinct values, and the average number of values in a row is four. The cardinality for this PublicationYear column is 9/4, or 3 (2.25 rounded to the next highest integer). Now that you have the cardinality calculation, you can update the SYSSTAT.COLUMNS catalog view using the following UPDATE statement:

```
UPDATE sysstat.columns SET colcard=3
WHERE
  tabschema=nickname-schema
  AND tabname=nickname-name
  AND colname=column-name
```

- *3* is the column cardinality value.
- *nickname-schema* is the name of the schema that is associated with the underlying nickname where the specified column is used.
- *nickname-name* is the name of the underlying nickname where the specified column is used.
- *column-name* is the name of the column whose cardinality statistics you want to update.

Related concepts:
- "BioRS statistical information" on page 87

Related tasks:
- "Updating BioRS nickname cardinality statistics" on page 88
- "Updating BioRS _ID_ column cardinality" on page 90

Updating BioRS _ID_ column cardinality

Maintaining correct cardinality statistics for the column that maps to the BioRS _ID_ element enables the optimizer and the BioRS wrapper to choose the best performing data access plan.

To update the cardinality number of the column that maps to the BioRS _ID_ element, you must modify the SYSSTAT.COLUMNS catalog view.

Prerequisites:

You must determine the cardinality number of the BioRS databank that corresponds to the nickname in which the column is referenced. The cardinality number of the column that maps to the BioRS _ID_ element must match the cardinality of the nickname in which the column is referenced.

Procedure:

To update BioRS _ID_ column cardinality statistics, issue an UPDATE statement using the following syntax:

```
UPDATE sysstat.columns SET colcard=<cardinality)
WHERE
     tabschema=nickname-schema
  AND tabname=nickname-name
  AND colname IN (SELECT colname FROM syscat.coloptions
                  WHERE
```

```
tabschema=nickname-name
AND tabname=nickname-name
AND option='ELEMENT_NAME';
AND setting='_ID_')
```

- *cardinality* is the BioRS databank cardinality number that corresponds to the nickname of the column.
- *nickname-schema* is the name of the schema that is associated with the nickname of the column.
- *nickname-name* is the name of the nickname in which the column is used.

Related concepts:
- "BioRS statistical information" on page 87

Related tasks:
- "Determining BioRS databank cardinality statistics" on page 88
- "Updating BioRS nickname cardinality statistics" on page 88
- "Updating BioRS column cardinality statistics" on page 89

Messages for the BioRS wrapper

This topic explains the messages that you might receive when you work with the wrapper for BioRS.

Table 21. Messages issued by the wrapper for BioRS

Error Code	Message	Explanation
SQL0604N	The length, precision or scale attribute for a column, distinct type, structured type, attribute of a structured type, function or type mapping <data-item> is not valid.	The data type for a nickname column is not compatible with the BioRS type of the underlying databank element. Check the data type of the column in the CREATE NICKNAME statement.
SQL0901N	The SQL statement failed because of a non-severe system error. Subsequent SQL statements can be processed. (Reason ″Error creating wrapper object.″)	An error occurred when you created a new wrapper object. Contact IBM Software Support.
SQL0901N	The SQL statement failed because of a non-severe system error. Subsequent SQL statements can be processed. (Reason ″BioRS <trace-point>/<code>.″)	This is an internal error. Contact IBM Software Support.
SQL0901N	The SQL statement failed because of a non-severe system error. Subsequent SQL statements can be processed. (Reason ″Memory allocation failed: <trace-point>.″)	An error occurred when memory was allocated. Ensure that sufficient memory is available to the federated server host and submit the query again. If the problem persists, contact IBM Software Support.
SQL0901N	The SQL statement failed because of a non-severe system error. Subsequent SQL statements can be processed. (Reason ″sqlno_crule_save_plans[100]:rc(-214272209) Empty plan list.″)	The optimizer program and the BioRS wrapper could not agree on a plan to run the query. Simplify the query and run it again.

Table 21. Messages issued by the wrapper for BioRS (continued)

Error Code	Message	Explanation
SQL0401N	The data types of the operands for the operation ″=″ are not compatible.	The query is not valid because the expression on the right side in a custom function predicate must be an integer value.
SQL1822N	Unexpected error code ″″ received from data source ″BioRS wrapper.″ Associated text and tokens are ″Databank not found.″	The BioRS databank referenced in a CREATE NICKNAME statement was not found on the BioRS server. Check the CREATE NICKNAME statement and ensure that the name of the referenced databank is correct.
SQL1822N	Unexpected error code ″″ received from data source ″BioRS wrapper.″ Associated text and tokens are ″Connection timed out.″	The BioRS server failed to respond to a communications request within the period specified by the TIMEOUT option.
SQL1822N	Unexpected error code ″<trace_point>″ received from data source ″BioRS wrapper.″ Associated text and tokens are ″Error reading from server.″	A communications error occurred while reading data from the BioRS server. The value of the <trace_point> error code might provide more information about the error.
SQL1822N	Unexpected error code ″<trace_point>″ received from data source ″BioRS wrapper.″ Associated text and tokens are ″Host not found.″	The BioRS server host that is identified in the HOST server option was not found. Check the CREATE SERVER statement and ensure that the HOST server option value is correct.
SQL1822N	Unexpected error code ″<trace_point>″ received from data source ″BioRS wrapper.″ Associated text and tokens are ″Unable to connect to server.″	The wrapper was unable to connect to the server that is identified by the HOST server option. The value of the <trace_point> error code might provide more information about the error.
SQL1822N	Unexpected error code ″<trace_point>″ received from data source ″BioRS wrapper.″ Associated text and tokens are ″Unable to create TCPIP socket.″	The wrapper could not create a TCPIP socket. The value of the <trace_point> error code might provide more information about the error code.
SQL1822N	Unexpected error code ″<trace_point>″ received from data source ″BioRS wrapper.″ Associated text and tokens are ″Error sending to server.″	The wrapper could not to send a request to the BioRS server. The value of the <trace_point> error code might provide more information about the error.
SQL30090N	Operation invalid for application execution environment. Reason code = ″Cannot change case-sensitivty of server.″	You cannot change the value of the CASE_SENSITIVE server option with SQL statements. To change the value of this option, you must drop the server. Then, you must create the server again using the CREATE SERVER statement, and specify the correct value for the CASE_SENSITIVE option.
SQL30090N	Operation invalid for application execution environment. Reason code = ″Multiple joins between two nicknames.″	The query is not valid because only one join predicate is allowed between any two nicknames.

Table 21. Messages issued by the wrapper for BioRS (continued)

Error Code	Message	Explanation
SQL30090N	Operation invalid for application execution environment. Reason code = ″Right side of function predicate must be constant.″	The query is not valid because the expression on the right side in a custom function predicate must be a constant.
SQL30090N	Operation invalid for application execution environment. Reason code = ″Arg 1 of custom function not a column.″	The query is not valid because the first argument of a custom function must reference a column of a BioRS nickname.
SQL30090N	Operation invalid for application execution environment. Reason code = ″Arg 1 of CONTAINS function not indexed.″	The query is not valid. The column referenced in the first argument of a BIORS.CONTAINS, BIORS.CONTAINS_LE, or BIORS.CONTAINS_GE function must be an indexed column.
SQL30090N	Operation invalid for application execution environment. Reason code = ″Bad type for arg1 of <function-name> function.″	The query is not valid. The column referenced in the first argument of a BIORS.CONTAINS, BIORS.CONTAINS_LE, or BIORS.CONTAINS_GE function is not of the correct data type.
SQL30090N	Operation invalid for application execution environment. Reason code = ″Arg 1 of SEARCH_TERM not _ID_ column.″	The query is not valid. The column referenced in the first argument of a SEARCH_TERM function does not map a BioRS _ID_ element.
SQL30090N	Operation invalid for application execution environment. Reason code = ″Bind parameter cannot be NULL.″	A column or host variable value that was referenced in the second argument of a BIORS.CONTAINS function was NULL. The BioRS wrapper cannot process null values.
SQL30090N	Operation invalid for application execution environment. Reason code = ″Cannot convert value to BioRS literal.″	A value was submitted to the wrapper in a literal, column, or host variable, which could not be converted to a valid BioRS literal.
SQL30090N	Operation invalid for application execution environment. Reason code = ″Cannot change server version.″	You cannot change the server version with the ALTER SERVER statement. To change the server version, you must drop the server. Then, you must create the server again with the correct version using the CREATE SERVER statement.
SQL30090N	Operation invalid for application execution environment. Reason code = ″Bad type for arg2 of <function-name> function.″	The query is not valid. The column referenced in the second argument of a BIORS.CONTAINS, BIORS.CONTAINS_LE, or BIORS.CONTAINS_GE function is not of the correct data type.
SQL30090N	Operation invalid for application execution environment. Reason code = ″Nickname has no columns.″	No column declarations were specified on the CREATE NICKNAME statement. Column declarations are required to create nicknames.

Related concepts:

- “Introduction to messages” in the *Message Reference Volume 1*

| **Related reference:**
| • "SQLSTATE messages" in the *Message Reference Volume 2*

Chapter 7. Configuring access to BLAST data sources

This chapter explains how to configure your federated server to access data that is stored in BLAST data sources. You can configure access to BLAST data sources by using the DB2 Control Center or by issuing SQL statements.

This chapter:
- Explains what BLAST is
- Lists the tasks that you need to perform
- Contains examples of the SQL statements that you need
- Lists the error messages associated with the BLAST wrapper

What is BLAST?

BLAST (Basic Local Alignment Search Tool) is a utility that is maintained by the National Center for Biotechnology Information (NCBI). BLAST is used to scan a nucleotide or amino acid sequence database for "hits." A BLAST hit contains one or more high-scoring segment pairs (HSPs). A HSP is a pair of sequence fragments, whose alignment is locally maximal, and whose similarity score exceeds some threshold value. NCBI provides an executable, blastall, that is used to perform BLAST searches on BLAST-able data sources, such as GenBank and SWISS-PROT.

The BLAST wrapper supports all five types of BLAST searches: BLASTn, BLASTp, BLASTx, tBLASTn, and tBLASTx. These are described in Table 22.

Table 22. BLAST search types supported by the BLAST wrapper

BLAST search type	Description
BLASTn	A type of BLAST search in which a nucleotide sequence is compared with the contents of a nucleotide sequence database to find sequences with regions homologous to regions of the original sequence.
BLASTp	A type of BLAST search in which an amino acid sequence is compared with the contents of an amino acid sequence database to find sequences with regions homologous to regions of the original sequence.
BLASTx	A type of BLAST search in which a nucleotide sequence is compared with the contents of an amino acid sequence database to find sequences with regions homologous to regions of the original sequence. The query sequence is translated in all six reading frames, and each of the resulting sequences is used to search the sequence database.
tBLASTn	A type of BLAST search in which an amino acid sequence is compared with the contents of a nucleotide sequence database to find sequences with regions homologous to regions of the original sequence. The sequences in the sequence database are translated in all six reading frames, and the resulting sequences are searched for regions homologous to regions of the query sequence.

Table 22. BLAST search types supported by the BLAST wrapper (continued)

BLAST search type	Description
tBLASTx	A type of BLAST search in which a nucleotide sequence is compared with the contents of a nucleotide sequence database to find sequences with regions homologous to regions of the original sequence. In a tBLASTx search, both the query sequence and the sequence database are translated in all six reading frames, and the resulting sequences are compared to discover homologous regions.

Figure 7 shows how BLAST works with your federated system.

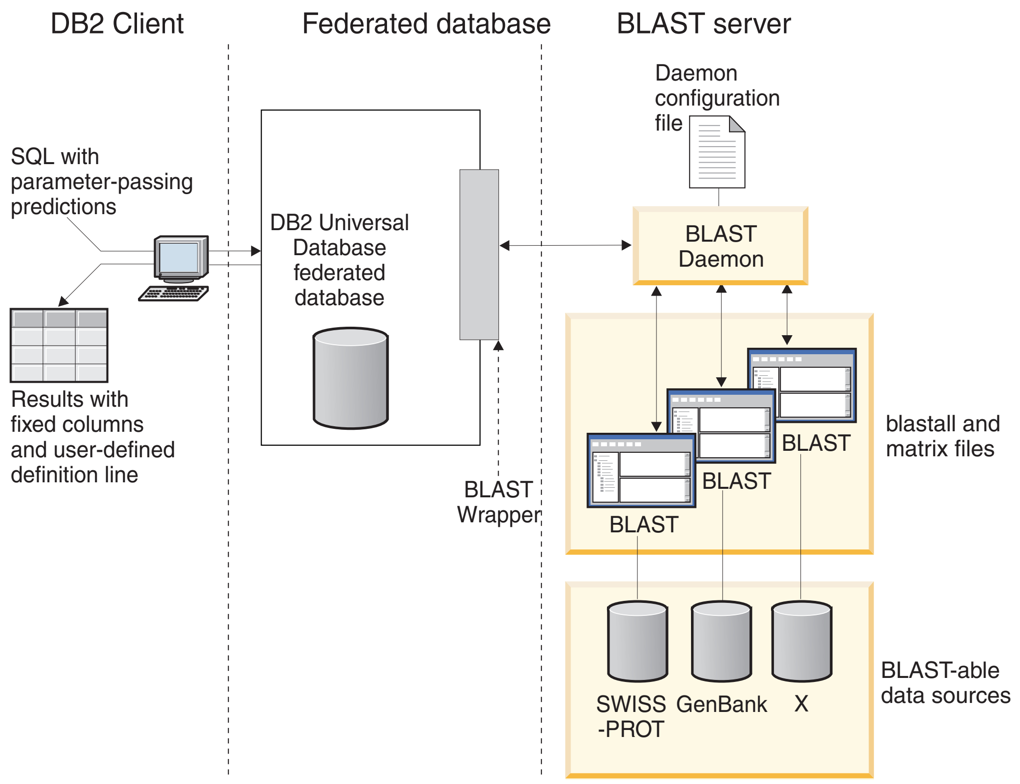

DB2 Client
Federated database
BLAST server
SQL with parameter-passing predictions
Results with fixed columns and user-defined definition line
DB2 Universal Database federated database
BLAST Wrapper
Daemon configuration file
BLAST Daemon
BLAST
BLAST
BLAST
blastall and matrix files
SWISS -PROT
GenBank
X
BLAST-able data sources

Figure 7. How the BLAST wrapper works

On the client side, users or applications submit SQL statements with BLAST-specific parameter-passing predicates that map to standard BLAST options. The SQL statements with the input predicates are sent to your DB2® Universal Database federated database system with the BLAST wrapper installed.

The BLAST wrapper transforms the query into a format understandable by the BLAST application and sends the transformed query to your BLAST server. This server can be a separate machine from the machine with the federated system. A special daemon program runs on your BLAST server. This daemon, using information from a daemon configuration file, receives the query request from the federated system and sends it to the BLAST application. The BLAST application then runs against a BLAST-able data source in the usual manner.

The results are returned to BLAST and then to the daemon. The daemon returns the retrieved data to the BLAST wrapper. The wrapper transforms the data into a relational table format, and returns this table to you or application. The returned data contains two parts:

- A series of standard, fixed columns familiar to BLAST users, and
- User-configured definition line information.

The following example illustrates how relational information is extracted from BLAST-able data sources. Data moves from raw fasta file format to a BLAST-able data set to a relational table that can be joined with other data sources in your federated system.

Figure 8 is a sample fasta file containing four definition line and nucleotide sequence records.

```
>7:4986 PMON5744
GTTCTTCCCAGTGCCCAAGTCCATTCTGACATCAATGAAGAAGGTAAAATCCCTGCGTGATCCCTCTGCC
AAGATGTCGAAATCAGACCCGGATAAACTAGCTGCTGTCAGAATAACAGACAGCCCGGAGGAGATCGTGC
AGAAGTTCCGCAAGGCTGTGACGGACTTCACCTCGGAGGTCACCTACGACCCGGCCAGGCGAGGAGGCGT
GTCCAACTTGGTGGCCATCCACGCGGCAGTGACCGGACTCCCGGTGGAGGAGGTGGTCCGCCGAAGTGCT
GGCATCAACACCGCTGGCTACAAGTTGGTGGTGGCGGAGGCTGTGATTGAGAGATTTGCACCAATTAAGA
GTGAAATTGAAAAACTGAAGAGGAACAAGGACCACCTAGAGAAGGTTTTACAAGTTGGGTCGGCAAAAGC
CAAAGAATTAGCATATCCCGTGTGCCAGGAGGTGAAGAAATTGGTGGGGTTTCTATAGGCAGTCTCACCT
AGTCCCAGAAAATGTTTTTTATCTTGTGGTCTGCTTGCACACTCAGTCTAATAAAGGCAGCTTTCCTAAG
ACGCCAACAATTCCAGTTTGGGGATGCTTAGTTTACT

>8:9747 PMON5699
AAGAAGTTCTTGTTAGAACTTTCCACCTCCGGCTTCCCCTCCACCTCTCTTACTGTCCCAACCTTCTGAG
ACGCTTTTTCTCCTCCCGAGGATTTATCTCTTTCTCTCTCTCTCTCTCTCTCTTTTTTTTTTTTCCCCT
TTTCCCCCCCCCGAGGCTGGTTTTGCTTTGGGGAGGGGGGGTTTTTTAAAGGGGCCGGGGGGGCCCCCTTT
CTCCCCCCTAATGGGGTTAATTAATAATGGGGGGGGGGGTTTTTTTTTTTTAAACCCCTATTTGGTCCGG
CCCGGGGATTTCCCCCCCCCCCCCCTTGCCCGGTTCCGGGGCCCGGAGGAGGGGGGGAAAAGGGCGGGAA
CCTTTGGTAGTTTCCCCTCGGAAAAAAATTTTTCGGGGGGGAAAACCTCCCT

>13:6512 PMON5498
GATAAGAGGCAGAATAGAAGACTGGACTACTTCTCTCCTAAAAACACATTTAAAACTAAGCCTGAGCAAT
CTCCACCCAAATGGACCGGAAACCTTAAAAAAGAATCCTACTCCTGAAGAAAAAGAGGAGGACACATCAA
GAGGTAGAAGGGGCGATTTCATGATATAAACAACCCCATACCTCCAGAGTGGGAAGCTCCACAGACTGAA
AACTAACTGGTTCACAGAAACTCACCTACAGGAGTGAGCCCCACATCAAACCCTCGAATGTGGGGATCTG
GCACTGGTAGAAAGAGCCCCTGGAGCATCTGGCATTGAAGGCCAGTGGGGCTTGTGTGCAGGAGATCCAC
AGGACTAGGGGAAACGGAGACCCCCATTCTTAAAAGGTGCACACAGACTTTTACGTGCACTGGGTCCCAG
TGCAAAGCAAAGTCTCCATAGGAATCTGGGTCAAACCTGACTGCAGTTCTTGGAGGACCTCCTGGGAAAG
CAAGGGTGAATGTGGCTTCTTGTGGGGAAAGGACATTGGAAGCAAAGCTCTTGGGAATATTCATCAGTGT
GC

>15:8924 PMON5426
GGAGAAACTGACTCCTGAGCAGCTGCAATTCATGCGGCAGGTGCAGCTCGCCCAGTGGCAGAAGACGCTG
CCACAGCGGCGGACCCGGAACATCGTGACCGGCCTGGGCATCGGGGCGCTGGTGTTGGCAATTTGTATCC
GTTTGGACTGTAGACTCAGGGAGACCGCATTTAGGGGAACAGGAAGGGCAGCAGGGGCGTGTAGGAGGGC
AGTGTGGGGGTGGTAGAAGGAGCCCGAGATATGAAAACCTTGGCTCCTTTTAACTCTGAATCAAGCGTTT
GGTGTACCTTACGTTGTCATTTTAAAGGTGTATTTTAGTATAATTGATTAATGATTACGGAGTCGGGTGA
GGGCTCCCAGGAGCAGACGGCAGAAGATCGAATTTGGGAGGATGATCAGCAGCGGTGGTTGAGCAAGTGT
GGGAAAAGGGAATGCGCACATTCCACGTGGTTTCCTGAACCCACCTCCCCAGATGGTTACACCTTCTACT
CGGTGTCCCAGGAGCGTTTCTTGGATGAGCTGGAGGATGAGGCCAAAGCTGCTC
```

Figure 8. Sample fasta file, nucleo1

The standard formatdb application transforms the fasta file to a BLAST-able data set. The data is now ready for querying by SQL through a federated system with the BLAST wrapper installed and registered.

The following query, sent by you or an application at the client end, is transformed by the BLAST wrapper. It then runs against the BLAST-able data set.

```
SELECT Unique_ID, Experiment_Number, Organism_Number, HSP_Info, Score
FROM nucleo1
WHERE BlastSeq = 'ACATTCTTATAGAGTATTGCTACTCCTCCAGGATAGAGTCATCTCT
 GGTCTCCAGAGCCACCGCTGGCTACAAGTTGGTGGTGGCGGAGGCTGTGATTGAGAGATTTG
 CACCAATACAGAAACTCACCTACAGGAGTGAGCGGGTGGTAGAAGGAGCCCGAGATATGAAA
 ACCTTGTTTCAAGACCCCATTGTCACCGGGG';
```

The results of the query are transformed by the BLAST wrapper into a relational table format shown in Table 23.

Table 23. BLAST returns results in relational table form when integrated into your federated system

Unique ID	Experiment number	Organism number	HSP_INFO	SCORE
PMON5744	4986	7	Identities = 57/201 (28%), Positives = 57/201 (28%), Gaps = 0/201 (0%)	+1.13487000000000E+002
PMON5426	8924	15	Identities = 35/201 (17%), Positives = 35/201 (17%), Gaps = 0/201 (0%)	+6.98754000000000E+001
PMON5498	6512	13	Identities = 26/201 (13%), Positives = 26/201 (13%), Gaps = 0/201 (0%)	+5.20342000000000E+001

The data is in a fully relational form and can be joined with data from other data sources used by your laboratory. Combining the results of several data sources can lead to insights not readily or efficiently discovered prior to the implementation of your federated system.

Related tasks:
- "Adding BLAST data sources to a federated server" on page 98

Adding BLAST to a federated server

Adding BLAST data sources to a federated server

To configure the federated server to access BLAST data sources, you must provide the federated server with information about the data sources and objects that you want to access.

You can configure the federated server to access BLAST data sources by using the DB2 Control Center or the DB2 command line. The DB2 Control Center includes a wizard to guide you through the steps required to configure the federated server.

Prerequisites:
- DB2 Information Integrator must be installed on a server that will act as the federated server
- A federated database must exist on the federated server

Procedure:

To add BLAST data sources to a federated server:

1. Verify that the correct version of the blastall executable and matrix files are installed.
2. Configure the BLAST daemon.
3. Start the BLAST daemon.
4. Register the wrapper.
5. Register the server definition.
6. Register nicknames for BLAST searches.

Related concepts:

- "DB2 Information Integrator installation process - overview" in the *IBM DB2 Information Integrator Installation Guide for Linux, UNIX, and Windows*
- "Fast track to configuring your data sources" on page 55

Related tasks:

- "Creating a federated database" on page 51
- "Verifying that the correct version of the blastall executable and matrix files are installed" on page 99
- "Configuring the BLAST daemon" on page 100
- "Starting the BLAST daemon" on page 103
- "Registering the BLAST wrapper" on page 105
- "Registering the server for a BLAST data source" on page 106
- "Registering nicknames for BLAST data sources" on page 107
- "Checking the setup of the federated server" on page 37

Related reference:

- "Supported operating systems for DB2 Information Integrator (32-bit)" in the *IBM DB2 Information Integrator Installation Guide for Linux, UNIX, and Windows*
- "Supported operating systems for DB2 Information Integrator (64-bit)" in the *IBM DB2 Information Integrator Installation Guide for Linux, UNIX, and Windows*

Verifying that the correct version of the blastall executable and matrix files are installed

Verifying that the correct version of the blastall executable and matrix files are installed is part of the larger task of adding BLAST to a federated system.

Prerequisites:

Verify that you have the latest version of the blastall executable and BLOSUM62, BLOSUM80, PAM30, and PAM70 matrix files installed on your BLAST server machine. If you do not have the latest version of the blastall executable, you must install the binary files and the matrix files. The matrix files must be in the same directory as the blastall executable.

Procedure:

To check the version level of your blastall executable and matrix files:

1. Run a BLAST search from the command line and note the version number located in the output file.

2. Check this product's Web site for versions of BLAST that have been tested with this wrapper to ensure you have a supported version.

The next task in this sequence of tasks is configuring the BLAST daemon.

Related tasks:

- "Configuring the BLAST daemon" on page 100

Configuring the BLAST daemon

Configuring the BLAST daemon is part of the larger task of adding BLAST to a federated system.

The BLAST wrapper requires a BLAST daemon. The BLAST daemon must be running on a server that you can access through TCP/IP from your federated system. This can be the same server that operates as the federated server, or a separate BLAST server.

The daemon runs separately from the wrapper and the federated database. The daemon listens for BLAST job requests from the wrapper.

Prerequisites:

The BLAST daemon must have:

- Execute access to the `blastall` binary file so that it can run BLAST searches.
- Write access to a directory in which it can write temporary files.
- Read access to at least one BLAST-able data source on which BLAST searches can be run. The `blastall` executable must have read access to both the data file and the BLAST index files generated by the formatdb program.

Restrictions:

The BLAST daemon might not run properly if the executable file or the database paths contain spaces. For example, you should not install the BLAST executable file in `C:\Program Files` on Windows servers.

Procedure:

To configure the BLAST daemon:

1. Ensure that the BLAST daemon executable files are on the proper server. During the installation of DB2 Information Integrator, the daemon executable files are installed in a directory on the federated server:

   On UNIX

   The daemon executable file is `db2blast_daemon`. This file is installed in the `$DB2PATH/bin` directory.

   On Windows

   The daemon executable files are `db2blast_daemon.exe` and `db2blast_daemon_svc.exe`. These files are installed in the `%DB2PATH%\bin` directory.

   If you use a separate BLAST server computer, you must copy the daemon executable files from the directory on the federated server to a directory on the

BLAST server computer. The daemon executable files can run in any directory on the BLAST server computer that does not contain spaces in the names in the directory path.

2. Ensure that the BLAST daemon configuration file is on the proper server. During the installation of DB2 Information Integrator, a sample daemon configuration file, `BLAST_DAEMON.config`, is installed in a directory on the federated server:

   On UNIX
   The daemon configuration file is installed in the `$DB2PATH/bin` directory. `$DB2PATH` is the directory in which DB2 Information Integrator is installed.

   On Windows
   The daemon configuration file is installed in the `%DB2PATH%\bin` directory. `%DB2PATH%` is the directory in which DB2 Information Integrator is installed, usually C:\SQLLIB\bin.

   By default, the daemon expects to find the configuration file in the working directory from which the daemon is started. You can copy the configuration file to another location. If you use a BLAST server computer, you must copy the daemon configuration file from the directory on the federated server to a directory on the BLAST server computer. You can copy the daemon configuration file to any directory on the BLAST server computer that the daemon can access.

3. Edit the daemon configuration file to work with your data source. You can also rename the configuration file.

   - The first line in the configuration file must be an equal sign. If the equal sign is missing, the daemon will not start. An error message will indicate that the DAEMON_PORT was not specified.
   - The last line in the configuration file must end with a new line. The sample configuration file that is provided with DB2 Information Integrator ends with a new line. When you edit the file, you must ensure that the last line in the file ends with a new line. If the last line does not end with a new line, you will receive an error message when you attempt to run your first BLAST query using the data source listed on the last line.
   - Specify the following options in the configuration file. For options that require paths, you can specify relative paths. Relative paths are relative to the directory from which the daemon process was started.

   DAEMON_PORT
   This is the network port on which the daemon listens for BLAST job requests submitted by the wrapper.

   MAX_PENDING_REQUESTS
   This is the maximum number of BLAST job requests that can be blocking on the daemon at any one time. This number does not represent the number of BLAST jobs that are running concurrently, only the number of job requests that can block at one time. It is recommended that you set this to a number greater than five. The BLAST daemon does not restrict the number of BLAST jobs that can run concurrently.

   DAEMON_LOGFILE_DIR
   This is the directory in which the daemon creates its log file. This file contains useful status and error information generated by the BLAST daemon.

Q_SEQ_DIR_PATH
: This is the directory in which a temporary query sequence data file is created by the daemon. This temporary file is cleaned up once the BLAST job completes.

BLAST_OUT_DIR_PATH
: This is the directory in which the daemon creates the temporary file to store the BLAST output data. Data is read from this file and passed back to the wrapper through the network connection. After the data is passed to the wrapper, the daemon cleans up the temporary file

BLASTALL_PATH
: This is the fully-qualified name of the BLAST executable file on the computer that is running the daemon.

database specification entry
: Specifies the location of a BLAST-able data source. Make note of the database *data_source_name* that you specify in the configuration file. For the daemon to function properly, you must specify the database *data_source_name* when you create the nickname for the data source. The name is case-sensitive. The database *data_source_name* is specified in the DATASOURCE option of the CREATE NICKNAME statement.

  The configuration file must contain at least one database specification entry in the following form:

  ```
  data_source_name = path to BLAST-able_data_source
  ```

  On UNIX
  : For example, to specify the GenBank BLAST-able data source you would add the following line to the daemon configuration file:

    ```
    genbank=/dsk/1/nucl_data/genbank
    ```

  On Windows
  : For example, to specify the GenBank BLAST-able data source you would add the following line to the daemon configuration file:

    ```
    c:\vnr_data\genbank_nonest1.fasta
    ```

  The path indicated in a database specification entry must contain the three index files.

  - For nucleotide data sources, the index files have these extensions:
    - .nhr
    - .nin
    - .nsq
  - For amino acid data sources, the index files have these extensions:
    - .phr
    - .pin
    - .psq

  The database specification entry must indicate the name of the file that contains the original Fasta-formatted data. The three index files must have the same root name as the file containing the original Fasta-formatted data.

The next task in this sequence of tasks is starting the BLAST daemon.

Related tasks:

- "Starting the BLAST daemon" on page 103

Related reference:

- "BLAST daemon configuration file - examples" on page 103

BLAST daemon configuration file - examples

The following examples show the contents of a sample configuration file.

Example – BLAST daemon configuration file (UNIX):

This example shows the required options and the BLAST-able data source specifications for GenBank and SWISS-PROT.

```
=
DAEMON_PORT=4007
MAX_PENDING_REQUESTS=10
DAEMON_LOGFILE_DIR=./
Q_SEQ_DIR_PATH=./
BLAST_OUT_DIR_PATH=./
BLASTALL_PATH=./blastall
genbank=/dsk/1/nucl_data/genbank
swissprot=/dsk/1/prot_data/swissprot
```

| **Example – BLAST daemon configuration file (Windows):**

| This example shows the required options and the BLAST-able data source
| specifications for GenBank and SWISS-PROT.

```
| =
| DAEMON_PORT=4007
| MAX_PENDING_REQUESTS=10
| DAEMON_LOGFILE_DIR=.\
| Q_SEQ_DIR_PATH=.\
| BLAST_OUT_DIR_PATH=.\
| BLASTALL_PATH=.\blastall.exe
| genbank=c:\vnr_data\genbank_nonest1.fasta
| swissprot=c:\vnr_data\swissprot
```

| **Related tasks:**

- "Adding BLAST data sources to a federated server" on page 98
- "Configuring the BLAST daemon" on page 100
- "Starting the BLAST daemon" on page 103

Starting the BLAST daemon

Starting the BLAST daemon is part of the larger task of adding BLAST to a federated system. Before you can access BLAST data sources, you must start the BLAST daemon.

Prerequisites:

Before you start the BLAST daemon, you must have write access to all paths listed under the `DAEMON_LOGFILE_DIR`, `BLAST_OUT_DIR_PATH`, and `Q_SEQ_DIR_PATH` entries in the configuration file.

Procedure:

To start the BLAST daemon on a UNIX server computer:

1. Open the directory where the daemon executable file is located.
2. Issue the `db2blast_daemon` command:
   - If you did not change the name of the daemon configuration file and the configuration file is in the same directory as the daemon executable file, type the following command at the command line:

     ```
     db2blast_daemon
     ```
   - If you changed the name of the daemon configuration file or if the daemon configuration file is not in the same directory as the daemon executable file, you must use the -c option on the wrapper daemon command to point the daemon executable to the new name or location.

     For example, the following command causes the wrapper daemon to look for the daemon configuration information in a file called `BLAST_D.config` in the subdirectory `cfg`.

     ```
     db2blast_daemon -c cfg/BLAST_D.config
     ```

   The executable file starts a new process in which the BLAST daemon runs.

To start the BLAST daemon on a Windows server computer:

1. Open the directory where the daemon executable file is located.
2. Issue the `db2blast_daemon` command with the parameters that you need. For example, to install the daemon service with debugging turned on:

   ```
   db2blast_daemon -a install -d 2
   db2blast_daemon -a start
   ```

The next task in this sequence of tasks is registering the BLAST wrapper.

Related tasks:

- "Registering the BLAST wrapper" on page 105

db2blast_daemon command - syntax and examples

The `db2blast_daemon` command can be used on UNIX and Windows servers. Some of the arguments listed in the syntax can be used only on Windows servers.

The syntax for the `db2blast_daemon` command is:

```
db2blast_daemon -a action -c config_file -d debug_level
    -u user_id -p password
```

-a *action*
: Performs the specified activity. Valid actions are *status*, *install*, *start*, *stop*, and *remove*.

  You can specify this argument only on Windows servers.

-c *config_file*
: Instructs the daemon service to use the specified configuration file. If you do not specify the configuration file, the daemon searches for the `BLAST_DAEMON.config`file in the directory where the daemon executable files are installed. You can use this option with the *install* and *start* actions.

  You can specify this argument on UNIX and Windows servers.

-d *debug_level*
Sets the daemon service debug level to the specified value. The valid values are 1, 2, or 3. You can use this option with the *install* and *start* actions.

You can specify this argument on UNIX and Windows servers.

-u *user_id*
Sets the daemon service to run under the specified user ID. You can use this option with the *install* action.

You can specify this argument only on Windows servers.

-p *password*
Specifies the password for the specified user ID. The password is valid and required only when you specify the -u option. If the -p option is not specified when you set the -u option, the program prompts you for the password. You can use this option with the *install* action.

You can specify this argument only on Windows servers.

The options that are specified with the *start* action affect only the current run of the daemon, and override the values that are specified with the *install* action.

Examples:

The following examples show daemon actions. These examples assume that the `BLAST_DAEMON.config` file is in the same directory as `db2blast_daemon.exe`.

- To check the status of the daemon:
  ```
  db2blast_daemon -a status
  ```
- To install the daemon service with debugging turned on:
  ```
  db2blast_daemon -a install -d 2
  ```
- To start the daemon:
  ```
  db2blast_daemon -a start
  ```
- To stop the daemon:
  ```
  db2blast_daemon -a stop
  ```
- To remove the daemon service:
  ```
  db2blast_daemon -a remove
  ```

Related tasks:
- "Starting the BLAST daemon" on page 103

Registering the BLAST wrapper

Registering the BLAST wrapper is part of the larger task of adding BLAST data sources to a federated server.

You must register a wrapper to access BLAST data sources. Wrappers are used by federated servers to communicate with and retrieve data from data sources. Wrappers are implemented as a set of library files.

Procedure:

To register a wrapper, issue the CREATE WRAPPER statement with the name of the wrapper and the name of the wrapper library file.

For example, to register a wrapper with the name `blast_wrapper` on the federated server that uses the AIX operating system, issue the following statement:

```
CREATE WRAPPER blast_wrapper LIBRARY 'libdb2lsblast.a';
```

The name of the wrapper library file that you specify depends on the operating system of the federated server. See the list of BLAST wrapper library files for the correct name to specify in the CREATE WRAPPER statement.

The next task in this sequence of tasks is registering the server definition for the BLAST wrapper.

Related reference:
- "BLAST wrapper library files" on page 106
- "CREATE WRAPPER statement" in the *SQL Reference, Volume 2*

BLAST wrapper library files

The following table lists the directory paths and library file names for the BLAST wrapper.

When you install DB2 Information Integrator, 3 library files are added to the directory path listed in the table. For example, if the federated server is running on AIX, the wrapper library files added to the directory path are `libdb2lsblast.a`, `libdb2lsblastF.a`, and `libdb2lsblastU.a`.

When you register a wrapper, specify only the library file name that is listed in the table.

Table 24. BLAST wrapper library locations and file names

Operating system	Directory path	Wrapper library file
AIX	/usr/opt/db2_08_01/lib/	libdb2lsblast.a
Linux	/opt/IBM/db2/V8.1/lib	libdb2lsblast.so
Solaris	/opt/IBM/db2/V8.1/lib	libdb2lsblast.so
Windows	%DB2PATH%\bin	db2lsblast.dll

%DB2PATH% is the environment variable that is used to specify the directory path where DB2 Information Integrator is installed on Windows. The default Windows directory path is `C:\Program Files\IBM\SQLLIB`.

Related tasks:
- "Registering the BLAST wrapper" on page 105

Registering the server for a BLAST data source

Registering the server for a BLAST data source is part of the larger task of adding BLAST to a federated system. After the wrapper is registered, you must register a corresponding server.

Procedure:

To register the BLAST server to the federated system, use the CREATE SERVER statement.

For each machine on which the BLAST executable and daemon are installed in your environment, you must register one server for each type of BLAST search you want to run using that instance of the BLAST executable and daemon.

For example, to register a server called blast_server1 for the my_blast wrapper created using the CREATE WRAPPER statement that will be used for BLASTn searches, submit the following statement:

```
CREATE SERVER blast_server1
 TYPE blastn
   VERSION 2.1.2
   WRAPPER my_blast
   OPTIONS (NODE 'big_rs.company.com', DAEMON_PORT '4007')
```

The next task in this sequence of tasks is registering nicknames for BLAST data sources.

Related tasks:

- "Registering nicknames for BLAST data sources" on page 107

Related reference:

- "ALTER SERVER statement" in the *SQL Reference, Volume 2*
- "CREATE SERVER statement" in the *SQL Reference, Volume 2*
- "CREATE SERVER statement arguments - BLAST wrapper" on page 538

Registering nicknames for BLAST data sources

Registering nicknames for BLAST data sources

Registering nicknames for BLAST data sources is part of the larger task of adding BLAST to a federated system.

After you register a server, you must register a corresponding nickname. Nicknames are used when you refer to a BLAST data source in a query.

Procedure:

To register a BLAST nickname, use the CREATE NICKNAME statement.

Since each type of BLAST search is handled by a separate server, you must define a separate nickname for each type of BLAST search that you want to run on a given BLAST-able data source.

When you create the nickname you specify column information for the definition line portion of the data source. All other columns are fixed.

There are no further tasks in this sequence of tasks.

Related concepts:

- "Definition line parsing" on page 108

Related tasks:

- "Adding BLAST data sources to a federated server" on page 98
- "Specifying nickname columns for a nonrelational data source" on page 65

Related reference:

- “CREATE NICKNAME statement syntax - BLAST wrapper” on page 539
- “CREATE NICKNAME statement - Examples for BLAST wrapper” on page 112
- “Fixed columns for BLAST nicknames” on page 108

Definition line parsing

The definition line is like a key for each sequence in the BLAST-able data source and is returned as part of each BLAST hit. The definition line is also called the *defline*.

The value that is returned and parsed by the BLAST wrapper for a definition line will not always be identical to the definition line in the original FASTA file. For example, if there is data in the Accession Number field of a BLAST hit, the definition line that is returned contains the Accession Number data followed by the Definition field data. The wrapper then parses the data that is returned.

Recommendation: To determine how the wrapper will return and parse the definition line, create a nickname with a single definition line column. Then run a query to see the format that is returned by the wrapper of the definition line for your particular data source.

To include the definition line information in your results table, you must specify the definition line columns in the CREATE NICKNAME statement. Each column that you specify must include the INDEX option and the DELIMITER option. You can omit the DELIMITER option on the last column that you specify, if you want the last column to contain the remainder of the definition line information.

| Valid data types for the definition line columns are CLOB, DOUBLE, FLOAT,
| INTEGER, and VARCHAR.

Related concepts:
- “Defline parsing user-defined functions - overview” on page 453

Related tasks:
- “Registering nicknames for BLAST data sources” on page 107

Related reference:
- “CREATE NICKNAME statement syntax - BLAST wrapper” on page 539
- “CREATE NICKNAME statement - Examples for BLAST wrapper” on page 112
- “Fixed columns for BLAST nicknames” on page 108

Fixed columns for BLAST nicknames

When you issue the CREATE NICKNAME statement for a BLAST data source, a set of fixed columns are automatically created with the nickname.

The fixed columns are part of the definition for the nickname and are created in the federated database system catalog. You can reference the fixed columns in SQL queries. There are two types of fixed columns, input fixed columns and output fixed columns.

Fixed input columns for BLAST nicknames: The fixed input columns are specified in the WHERE clause. Input columns are used as parameter-passing predicates in SQL queries. They pass standard BLAST switches to BLAST. BLAST then runs on the specified data source using these switches. Fixed input columns can also be referenced in the query SELECT list and are returned as part of the results table.

The following table lists the fixed columns that you can use in the WHERE clause.

Table 25. Fixed input columns for BLAST nicknames

Name	Data type	Operators	Description
BlastSeq	VARCHAR (32000) or CLOB	=	Passes the query sequence to the BLAST wrapper.
E_Value	DOUBLE	<	Both an input and an output parameter. As an input parameter, this column indicates to the BLAST wrapper the upper limit of expect values that should be returned from blastall.
QueryStrands	INTEGER	=	Specifies which strands should be compared when performing a BLASTn search. A value of 1 indicates that the top strand should be used, 2 indicates the bottom strand, and 3 indicates that both strands should be compared.
GapAlign	CHAR(1)	=	Indicates to the wrapper whether gapped alignments are permitted in the BLAST output.
Matrix	VARCHAR(50)	=	Determines which substitution matrix is used by blastall to determine the degree of similarity between pairings of amino acids. Only those BLAST search types that compare amino acids to amino acids use this predicate.
NMisMatchPenalty	INTEGER	=	Specifies the value that blastall deducts from the score of an alignment if one of the pairs of nucleotides in the homologous region does not match. Only those BLAST search types that compare nucleotides to nucleotides use this predicate.
NMatchReward	INTEGER	=	Specifies the value that blastall adds to the score of an alignment for each of the pairs of nucleotides in the homologous region that do match. Only those BLAST search types that compare nucleotides to nucleotides use this predicate.
FilterSequence	CHAR(1)	=	Indicates to blastall whether to perform filtering to remove biologically uninteresting segments from the query sequence. If the search type is BLASTn, the filter used is DUST. Otherwise, filtering is performed by SEG.

Table 25. Fixed input columns for BLAST nicknames (continued)

Name	Data type	Operators	Description
NumberOfAlignments	INTEGER	=	Specifies how many HSP alignments to include in the BLAST output.
GapCost	INTEGER	=	Specifies the value that blastall deducts from the score of an alignment if a gap must be introduced in either the query sequence or the hit sequence to allow the length of the alignment to grow.
ExtendedGapCost	INTEGER	=	Specifies the value that blastall deducts from the score of an alignment if a gap that was already introduced in either the query sequence or the hit sequence must be extended by one nucleotide or amino acid to allow the length of the alignment to grow.
WordSize	INTEGER	=	Indicates to blastall the length of the initial hits that blastall initially searches in the database.
ThresholdEx	INTEGER	=	Indicates the score threshold below which BLAST does not attempt to extend a hit any further.

| You can override the default data type for a column when you create a nickname. For example, some columns can return a large amount of data, such as the HSP_H_Seq and HSP_Midline columns. To return the first 50 bytes of a column, you can define the column with the data type VARCHAR(50). Only the first 50 bytes will be copied into the output column.

BLAST search types and switches for fixed input columns: The supported BLAST search types and switches for each fixed input column are listed in the following table.

Table 26. BLAST search types and switches supported by the input fixed columns

Name	BLAST search types	BLAST switch	Required	Default
BlastSeq	n, p, x, tn, tx	–l	Yes	N/A
E_Value	n, p, x, tn, tx	–e	No	10
QueryStrands	n	S	No	3
GapAlign	n, p, x, tn, tx	–g	No	T
Matrix	p, x, tn, tx	–n	No	BLOSUM62
NMisMatchPenalty	n	–q	No	–3
NMatchReward	n	–r	No	1
FilterSequence	n, p, x, tn, tx	–F	No	T
NumberOfAlignments	n, p, x, tn, tx	–b	No	250
GapCost	n, p, x, tn, tx	–G	No	11
ExtendedGapCost	n, p, x, tn, tx	–E	No	1

Table 26. BLAST search types and switches supported by the input fixed columns (continued)

Name	**BLAST search types**	**BLAST switch**	**Required**	**Default**
WordSize (for Blastn, a value less than 7 is invalid)	n, p, x, tn, tx	–W	No	11 –BLASTn 3 –BLASTp
ThresholdEx	n, p, x, tn, tx	–f	No	0

Fixed output columns for BLAST nicknames: The following table lists the fixed columns that you can use in the WHERE clause.

Table 27. Fixed output columns for BLAST nicknames

Name	**Data type**	**Description**
Score	DOUBLE	The computed score for an HSP as reported in the BLAST results.
E_value	DOUBLE	Both an input and an output parameter. As an output parameter, this column provides the computed score for an HSP as reported in the BLAST results.
Length	INTEGER	The length of the hit sequence as reported in the BLAST results.
HIT_NUM	INTEGER	The hit number as reported in the BLAST results, starting with 1.
HSP_NUM	INTEGER	The HSP number as reported in the BLAST results, starting with 1.
HSP_Info	VARCHAR(100)	The information string for the given HSP, as reported by BLAST. This string contains information about the number of nucleotides or amino acids that matched between the query sequence and the hit sequence.
HSP_ALIGNMENT_LENGTH	INTEGER	The length of the HSP alignment.
HSP_IDENTITY	INTEGER	The percent identity of the alignment defined as the number of identities divided by the alignment length.
HSP_GAPS	INTEGER	The percent gaps in the alignment defined as the number of gaps divided by the alignment length.
HSP_POSITIVE	INTEGER	The percent positives of the alignment defined as the number of positives divided by the alignment length.
HSP_QUERY_FRAME	INTEGER	The reading frame of the alignment in the query sequence. Only available for blastx, tblastn, and tblastx type servers.
HSP_HIT_FRAME	INTEGER	The reading frame of the alignment in the hit sequence. Only available for blastx, tblastn, and tblastx type servers.

Table 27. Fixed output columns for BLAST nicknames (continued)

Name	Data type	Description
HSP_Q_Start	INTEGER	The numeric position of the first homologous nucleotide or amino acid on the query sequence.
HSP_Q_End	INTEGER	The numeric position of the last homologous nucleotide or amino acid on the query sequence.
HSP_Q_Seq	VARCHAR(32000)	The segment of the query sequence beginning at HSP_Q_Start and ending at HSP_Q_End. You can override the default data type for this column and specify CLOB, with a maximum length of 5 megabytes.
HSP_H_Start	INTEGER	The numeric position of the first homologous nucleotide or amino acid on the hit sequence.
HSP_H_End	INTEGER	The numeric position of the last homologous nucleotide or amino acid on the hit sequence.
HSP_H_Seq	VARCHAR(32000)	The segment of the hit sequence beginning at HSP_H_Start and ending at HSP_H_End. You can override the default data type for this column and specify CLOB, with a maximum length of 5 megabytes.
HSP_Midline	VARCHAR(32000)	The string output by BLAST that indicates the degree of homology between the amino acids or nucleotides at each position in the homologous regions of the query and hit sequences. You can override the default data type for this column and specify CLOB, with a maximum length of 5 megabytes.

Related concepts:
- "Definition line parsing" on page 108

Related tasks:
- "Registering nicknames for BLAST data sources" on page 107

Related reference:
- "CREATE NICKNAME statement syntax - BLAST wrapper" on page 539
- "CREATE NICKNAME statement - Examples for BLAST wrapper" on page 112

CREATE NICKNAME statement - Examples for BLAST wrapper

The following CREATE NICKNAME statement defines the nickname genbank.

It assumes the definition field in a BLAST result contains the following information:

```
>276342 15:8924 PMON5426
```

where:

276342 The accession field of the BLAST result.

15:8924 PMON5426
The definition field in a BLAST result containing an organism number followed by an experiment number and then a unique identifier.

With this information, the following nickname is created:

```
CREATE NICKNAME genbank (
     acc_num integer   OPTIONS(INDEX '1', DELIMITER ' '),
     org_num integer   OPTIONS(INDEX '2', DELIMITER ':'),
     exp_num  integer   OPTIONS(INDEX '3', DELIMITER ' '),
     u_id  varchar(10)  OPTIONS(INDEX '4'))
     FOR SERVER blast_server1
       OPTIONS(DATASOURCE 'genbank', TIMEOUT '300');
```

The column `acc_num` would contain `276342`, the column `org_num` would contain `15`, the column `exp_num` would contain `8924`, and the column `u_id` would contain `PMON5426`.

After you submit the CREATE NICKNAME statement, you can use the nickname genbank to query your federated system. You can also join the genbank nickname with other nicknames and tables in your federated system.

Related tasks:

- "Registering nicknames for BLAST data sources" on page 107

Related reference:

- "ALTER NICKNAME statement" in the *SQL Reference, Volume 2*
- "CREATE NICKNAME statement" in the *SQL Reference, Volume 2*
- "CREATE NICKNAME statement syntax - BLAST wrapper" on page 539

Setting up TurboBlast to work with the BLAST wrapper

Restrictions:

TurboBlast does not support certain blastall command options. For example, the gapped alignment option `-g F` is not supported. If you specify `F` for the value of the GapAlign's column in your BLAST nickname, TurboBlast generates an error. For a complete list of unsupported options, refer to the *TurboBlast 2.0 User Guide*.

Procedure:

To set up TurboBlast to work with the BLAST wrapper:

1. Install and configure the BLAST wrapper. Run a query on a blastable database to test your setup.
2. The BLAST wrapper and TurboBlast support AIX, Linux, Solaris and Windows operating systems. However, the BLAST daemon is not available on Windows operating systems. The daemon will work with TurboBlast on Windows operating systems when the BLAST daemon is available on those operating systems.

3. Install and configure TurboBlast according to the *TurboBlast 2.0 Installation and Reference Guide*. You can install and set up the TurboBlast system in various ways. To allow the BLAST wrapper to work with TurboBlast, you need to install and set up the TurboBlast Client on the computer on which you have your BLAST daemon. The BLAST daemon can invoke the `tblastall` command.
4. Be sure to test the TurboBlast system after you have installed and configured TurboBlast. Follow the instructions in the *TurboBlast 2.0 Installation and Reference Guide*.
5. Change your `BLAST_DAEMON.config` file as follows:
   a. Specify the `BLASTALL_PATH` parameter as the complete path of `tblastall`. For example: `BLASTALL_PATH=/home/blasttst/turboblast/TBlast-2.1/tblastall`
   b. Specify the blastable database specification entry as the blastable database name that you used to upload your blastable database to TurboBlast. The database names are shown when you enter the `listdatabase -l` command under TurboBlast. This TurboBlast database name should be used instead of the path to the blastable data source. For example: `genbank=<the genbank database name in TurboBlast>`
6. Restart the BLAST daemon. The BLAST daemon invokes `tblastall` instead of `blastall` to do search work on the blastable databases.
7. The log files related to `tblastall` are written to the `DAEMON_LOGFILE_DIR` specified in your `BLAST_DEAMON.config` file. Also check the `STDERR.log` and `STDOUT.log` produced by the BLAST daemon in the same directory.

Related tasks:

- "Adding BLAST data sources to a federated server" on page 98
- "Configuring the BLAST daemon" on page 100

Constructing BLAST SQL queries

Predicates on input columns are used to pass standard BLAST switches to the blastall executable. Predicates on the output columns are processed by the federated server.

Restrictions:

To be valid, every query passed to the BLAST wrapper must contain at least the `BlastSeq` input predicate. All other predicates are optional.

Procedure:

To construct a BLAST query, use the input predicates in the WHERE clause of your SQL statement.

The following example shows three input predicates: `BlastSeq`, `GapCost`, and `NMisMatchPenalty`.

```
Select * from blast b where
BlastSeq = 'GTCCAGCC...' AND
GapCost = -10 AND
NMisMatchPenalty = -4;
```

Related tasks:

- "Registering nicknames for BLAST data sources" on page 107

Related reference:

- "BLAST data source – Example queries" on page 115

BLAST data source – Example queries

Several sample BLAST queries are provided to illustrate how queries are constructed for BLAST data sources.

To run queries, use the examples as a guide.

In these queries, the name used for each nickname indicates the type of BLAST search and the data source. This is done so that the registration statements do not need to be listed with each sample query. Also, some of the queries make use of other hypothetical data sources so that these examples can illustrate the behavior of the wrapper when joined with other data sources.

Query 1

```
select *
from blastn_genbank
where BlastSeq =
 'caacccctccagccgagttgtcaatggcgaggaagctgttccccac';
```

When this SQL statement is executed, the wrapper will perform a BLASTn search of GenBank using the indicated sequence. The wrapper will return all of the available columns, including both the input parameter columns and the BLAST result columns.

Query 2

```
select *
from blastn_genbank
where BlastSeq =
 'caacccctccagccgagttgtcaatggcgaggaagctgttccccac'
 and GapCost = 8 and NmisMatchPenalty = -4;
```

When this SQL statement is executed, the wrapper will perform a BLASTn search of GenBank using the indicated sequence. In addition, the wrapper will pass the two indicated parameters to the daemon, and they will be passed to the blastall command line. The wrapper will return all of the available columns, including both the input parameter columns and the BLAST result columns.

Query 3

```
select blp.*
from blastp_swissprot blp, protein_db prdb
where prdb.keyword = 'malic enzyme'
 and blp.BlastSeq = prdb.sequence;
```

When this SQL statement is executed, the wrapper will perform zero or more BLASTp searches of SWISS-PROT, depending on the number of sequences returned from a hypothetical protein sequence database. This statement will be broken into two separate queries by DB2, and one BLASTp search will be run for each row that is returned from the hypothetical protein database. The wrapper will return all of the available columns, including both the input parameter columns and the BLAST result columns.

Query 4

```
select Score, E_Value, HSP_Info, HSP_Q_Seq, HSP_H_Seq, HSP_Midline
from blastx_swissprot
where BlastSeq = 'gagttgtcaatggcgagg'
 and GapCost = 8;
```

When this SQL statement is executed, the wrapper will perform a BLASTx search of SWISS-PROT using the indicated sequence. In this case, blastall will translate the input sequence in all six reading frames and perform the homology search using each of the six newly created protein sequences. The HSPs in the results will contain amino acid-amino acid alignments, rather than nucleotide-nucleotide alignments. The supplied parameter will be passed to the daemon and then to blastall via the command line. The wrapper will return only those columns that are specifically requested in the query.

Query 5

```
select tblx.Score, tblx.E_Value, tblx.HSP_Info tblx.HSP_Q_Seq,
 HSP_H_Seq, HSP_Midline
from tblastx_genbank tblx, gen_exp_database gedb
where tblx.BlastSeq = gedb.sequence
 and gedb.organism = 'interesting organism'
 and GapCost = 8
 and FilterSequence = 'F';
```

When this SQL statement is executed, the wrapper will perform zero or more tBLASTx searches of GenBank, depending on the number of sequences returned from a hypothetical gene expression database. The statement will be broken into two separate queries by DB2, and one tBLASTx search will be run for each row that is returned from the hypothetical gene expression database. In this case, blastall will translate the input sequence and all of the sequences in GenBank in all six reading frames and perform the homology search using each of the six newly created query protein sequences and all of the newly created database protein sequences. The HSPs in the results will contain amino acid-amino acid alignments, rather than nucleotide-nucleotide alignments. The supplied parameters will be passed to the daemon and then to blastall via the command line. The wrapper will return only those columns that are specifically requested in the query.

Related reference:

- “Documentum data source – Example queries” on page 187
- “Excel data source – Example queries” on page 221

Optimization tips for the BLAST wrapper

| To improve network communication performance, the federated server and the
| BLAST server should be on separate hardware. The BLAST daemon should reside
| on the BLAST server.

Related tasks:

- “Configuring the BLAST daemon” on page 100

Messages for the BLAST wrapper

This section lists and describes messages that you might encounter when working with the wrapper for BLAST.

Table 28. Messages issued by the wrapper for BLAST

Error Code	Message	Explanation
SQL0901N	The SQL statement failed because of a non-severe system error. Subsequent SQL statements can be processed. (Reason "sqlno_crule_save_plans [100]:rc (–2144272209) Empty plan list detect".)	The SQL query submitted to DB2 could not be processed by the wrapper. Correct the syntax and resubmit.
SQL1816N	Wrapper "BLAST_WRAPPER" cannot be used to access the "type" of data source ("<server type>" "") that you are trying to define to the federated database.	The CREATE SERVER statement used an invalid TYPE. The type must be one of the supported BLAST types.
SQL1817N	The CREATE SERVER statement does not identify the "version" of data source that you want defined to the federated database.	The CREATE SERVER statement did not specify the version.
SQL1822N	Unexpected error code "Unspecified Error" received from data source "Blast Wrapper". Associated text and tokens are "Unable to connect to daemon".	The blast wrapper was not able to connect to the daemon. The daemon might not be running. It might be misconfigured. The machine that it is running on might be unreachable.
SQL1822N	Unexpected error code "Unspecified Error" received from data source "Blast Wrapper". Associated text and tokens are "Blast daemon timeout expired".	No results were received from the daemon before the timeout as specified on the CREATE NICKNAME statement elapsed. Increase the timeout or check to see if there is a problem with the daemon.
SQL1822N	Unexpected error code "Unspecified Error" received from data source "Blast Wrapper". Associated text and tokens are "Blast Daemon Failed".	The daemon stopped communicating or the results returned were not properly formatted.
SQL1822N	Unexpected error code "Unspecified Error" received from data source "Blast Wrapper". Associated text and tokens are "Unknown error from the blast daemon".	The blast wrapper received an error code from the daemon that it doesn't recognize. The daemon version might not be compatible with the wrapper version.
SQL1822N	Unexpected error code "Unspecified Error" received from data source "Blast Wrapper". Associated text and tokens are "Column rename not allowed".	An ALTER NICKNAME statement was issued trying to rename one of the columns. Renaming a column is not allowed.
SQL1822N	Unexpected error code "Unspecified Error" received from data source "Blast Wrapper". Associated text and tokens are "XML parser error".	The Xerces parser is in an invalid state or has thrown an exception.

Table 28. Messages issued by the wrapper for BLAST (continued)

Error Code	Message	Explanation
SQL1823N	No data type mapping exists for data type "<data type name>" from server "<server name>".	The data type specified is not supported by this column.
SQL1881N	"DEFAULT" is not a valid "COLUMN" option for "<column-name>"	The DEFAULT option was used on a column that does not support it. Output only columns and definition line columns do not have default values.
SQL1882N	The "COLUMN" option "DEFAULT" cannot be set to "<option-value>" for "<column-name>".	The value specified for the DEFAULT option is of an incompatible type for the column or is incorrectly formatted.

Related concepts:

- "Introduction to messages" in the *Message Reference Volume 1*

Related reference:

- "SQLSTATE messages" in the *Message Reference Volume 2*

Chapter 8. Configuring access to business application data sources

This section explains how to add business application data sources to your federated system by using the Webspshere Business Integration wrapper.

The WebSphere Business Integration wrapper

The WebSphere® Business Integration wrapper is a read-only wrapper, and uses the WebSphere Business Integration Adapters to access business applications. See the *IBM DB2 Information Integrator Federated Systems Guide* for a list of supported adapters and applications.

The WebSphere Business Integration wrapper provides an SQL interface to business applications, such as those produced by SAP, Siebel, and PeopleSoft. By using the WebSphere Business Integration wrapper, you can use the federated systems functions to join business data from business applications with data on other federated data sources. The WebSphere Business Integration wrapper extracts business object definitions into a hierarchy of nicknames.

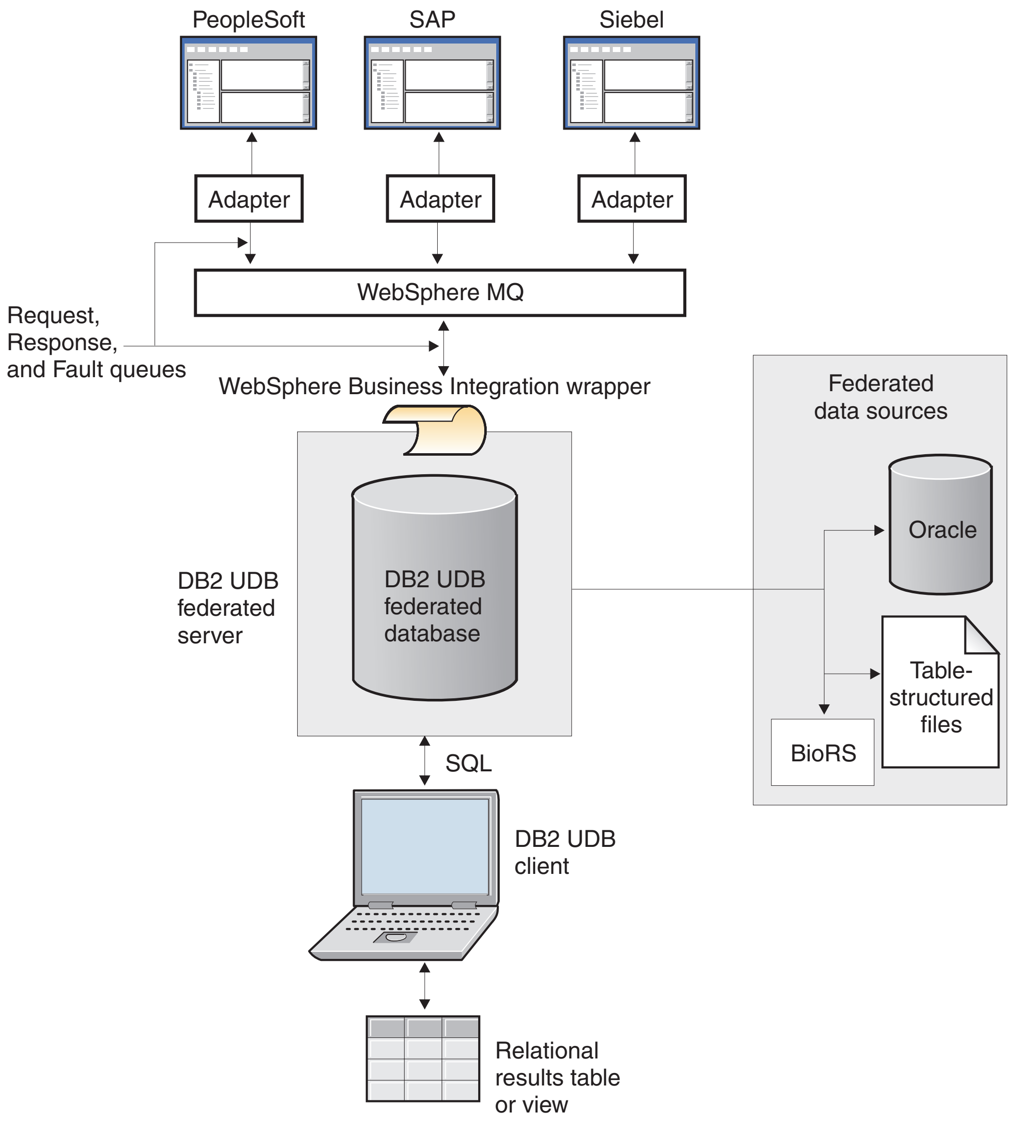


Figure 9. WebSphere Business Integration wrapper in the DB2® Universal Database environment

Figure 9 shows the relationship between the WebSphere Business Integration wrapper and the adapters in the DB2 Universal Database™ environment. The following steps describe the process for accessing business application data in a federated system:

1. A user sends a query to the federated server to access a nickname that maps to a data source such as a Siebel application.
2. The wrapper transforms the query into a business object.
3. The wrapper places the business object on a WebSphere MQ message queue.
4. The WebSphere Business Integration adapter for the particular application reads the business object, which is the request, from the message queue.
5. The WebSphere Business Integration adapter works with the business application to prepare a response business object.
6. The WebSphere Business Integration adapter puts the response business object on the message queue.
7. The wrapper reads the response business object from the response queue.

8. The wrapper extracts the response business object into a result set based on the relational schema that is defined with the nickname definition.

Related concepts:

- "Business object definitions" on page 121

Related tasks:

- "Adding business application data sources to a federated system" on page 125
- "Registering the WebSphere Business Integration wrapper" on page 126
- "Configuring the WebSphere Business Integration Adapters" on page 122
- "Registering nicknames for business application data sources" on page 129

Related reference:

- "Business application data sources – example queries" on page 155
- "CREATE NICKNAME statement – examples for the WebSphere Business Integration wrapper" on page 138

Business object definitions

A business object definition is a template from which the WebSphere® Business Integration Adapter creates an instance of a business object. A business object definition represents a business application data entity.

A business object is an instance of a business object definition and can be either a flat or hierarchical structure. A flat business object contains only simple attributes. A hierarchical business object contains one or more complex attributes. A repository of business object definitions exists within each WebSphere Business Integration Adapter for the particular application that is supported.

The following example shows a flat business object:

```
Customer
     Gomez
     Juanita
     Apt 2C
     123 Main Street
     Big City
     California
     91234
     888
     1111111
```

The following example shows a hierarchical business object with some complex attributes:

```
Contact (Parent)
     ID
     Customer ID
     Date
     Text
     Authorization
     Line items (there are 0 or more Line item elements)
          (Child elements)
          Business object 1
          Business object 2
          Business object 3
```

Business object definitions must be generated by using the Object Discovery Agent tool that is packaged with each WebSphere Business Integration Adapter. The Object Discovery Agent tool generates an XML schema definition file for a business object definition. The Object Discovery Agent tool might generate multiple schema files if the business object that is being defined has a hierarchical structure.

The XML schema definition is a file with file type `.xsd` in a directory that is specified in the WebSphere Business Integration configuration. You must generate the business object definition before you create the nicknames for the WebSphere Business Integration wrapper. For more information about the family of WebSphere Business Integration Adapters, see: www.ibm.com/websphere/integration/wbiadapters.

To create nicknames, you use the xsd file that the Object Discovery Agent tool creates. Nicknames provide a relational schema representation of the business object definition. The WebSphere Business Integration wrapper maps a hierarchical business object into a hierarchy of relational nicknames. For example, each child business object of cardinality 'n' is mapped to a separate nickname that is linked to the nickname of the parent business object with a foreign key constraint.

The WebSphere Business Integration business objects that are accessible to IBM® DB2® Information Integrator map to the specific application entities in the following table:

Table 29. Business objects and the related application entities

Business objects	Application entities
Siebel	Business Component
PeopleSoft	Component Interface
SAP	BAPI

Related concepts:
- "The WebSphere Business Integration wrapper" on page 119

Related tasks:
- "Adding business application data sources to a federated system" on page 125
- "Configuring the WebSphere Business Integration Adapters" on page 122

Related reference:
- "Business application data sources – example queries" on page 155

Configuring the WebSphere Business Integration Adapters

For each business application that you want to access with SQL statements by using the federated wrapper functions, you must install and configure a WebSphere Business Integration Adapter. Each adapter maps to a federated server definition.

Prerequisites:
- See the *IBM DB2 Information Integrator Federated Systems Guide* for a list of the adapters that are supported.
- See the IBM WebSphere Business Integration Information Center for installation information for each adapter.

- See the WebSphere Business Integration Adapters documentation for help with a specific business application.
- Install all of the most current fix packs for the particular adapter that you want to use. You can get the pertinent support information for the Adapters from the WebSphere Business Integration Adapters Support site.
- See the *WebSphere MQ System Administration Guide* for information about configuring the message queues.
- See the installation information in the IBM WebSphere Business Integration Information Center for information about the adapters and the configuration properties.

Procedure:

To configure a WebSphere Business Integration Adapter:

1. Configure the Object Discovery Agent tool and the Business Object Designer tool, and build the business object definitions.

   When you configure the business object definitions in the Business Object Designer tool, specify the following verb values depending on the business application:

   Table 30. Verb values that are used with business applications

Business application	Verb
SAP	Retrieve
PeopleSoft	Retrieve
Siebel	RetrieveByContent

   For more information about how to configure and use an Object Discovery Agent tool, see the documentation for the adapter that you are configuring.
2. Use the Connector Configurator tool from the WebSphere Business Integration Adapter interface to define a configuration file that contains the following information:
   - The business objects that the adapter supports.
   - The configuration properties for the adapter. There are standard configuration properties and application-specific configuration properties.

     Standard configuration properties
     You must customize some property values to use the adapter with IBM DB2 Information Integrator. Some specific properties to configure are included in the following list:
     - Specify the value of the integration broker as `WMQI`.
     - Specify the location of the metadata repository that is owned by the adapter. The XML schema definition files, which contain the business object definitions, are saved in this location.
     - Specify the type of delivery transport as `WMQI-MQ`.
     - Specify the name of the queue manager that manages the queues that are used by the adapter.
     - Specify the names of the eight queues that are required to run the adapter.

     Application-specific configuration properties
     These properties specify values for a particular application-specific component. The values that you provide help to establish a session

with the application. The properties also direct the processing behavior for the application-specific components.

3. Define the three WebSphere MQ message queues that the wrapper requires: request_queue, response_queue, fault_queue.

   WebSphere MQ is the messaging and transport layer between the adapter and the wrappers.

   request_queue
      Delivers request messages from DB2 Information Integrator to the adapter.

   response_queue
      Delivers response messages from the adapter to DB2 Information Integrator.

   fault_queue
      Delivers fault messages from the adapter to DB2 Information Integrator. The adapter places a message on this queue when it is unable to place the message on the reply-to queue.

   These queues are static queues that are used to exchange messages, including data objects and error messages, between the adapter and the wrapper.

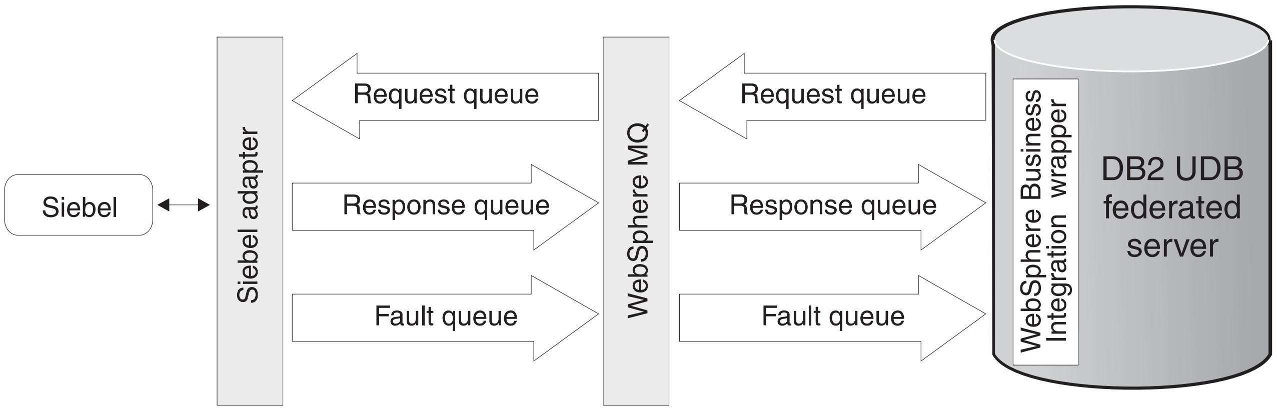


Figure 10. The topology of the WebSphere message queues that transport information between the Siebel business applications and the DB2 federated server

4. Define the five additional message queues that are required by the adapter:
   - AdminInQueue
   - AdminOutQueue
   - SynchronousRequestQueue
   - SynchronousResponseQueue
   - DeliveryQueue

   The WebSphere Business Integration adapters require five additional queues that are used when the adapter is used with a WMQI broker instead of DB2 Information Integrator. You must create and configure these additional message queues so that the adapter can be started.
5. Define the WebSphere MQ user authorization by using either of the following methods:
   - Define the DB2 instance owner ID as part of the MQManager group.

- Ensure that the MQManager administrator sets the MCAUSER value while creating the ServerConnection channel. The value of MCAUSER must be a user ID that is part of the MQManager group or the Administrator group.

Related concepts:
- "Business object definitions" on page 121
- "The WebSphere Business Integration wrapper" on page 119

Related tasks:
- "Adding business application data sources to a federated system" on page 125
- "Registering the WebSphere Business Integration wrapper" on page 126

Related reference:
- "CREATE NICKNAME statement – examples for the WebSphere Business Integration wrapper" on page 138

Adding business applications to a federated server

Adding business application data sources to a federated system

To configure the federated server to access business application data sources, you must provide the federated server with information about the data sources and objects that you want to access.

You can configure the federated server to access business application data sources by using the DB2 Control Center or the DB2 command line. The DB2 Control Center includes a wizard to guide you through the steps that are required to configure the federated server.

Prerequisites:
- DB2 Information Integrator must be installed on a server that will act as the federated server.
- A federated database that uses a 32 bit DB2 UDB instance must exist on the federated server.

Procedure:

To add business application data sources to a federated system:
1. Register the WebSphere Business Integration wrapper.
2. Register the server definition.
3. Register nicknames for business application data sources.
4. Optional: Create federated views for the WebSphere Business Integration nicknames.

Related concepts:
- "DB2 Information Integrator installation process - overview" in the *IBM DB2 Information Integrator Installation Guide for Linux, UNIX, and Windows*
- "Fast track to configuring your data sources" on page 55
- "Business object definitions" on page 121
- "The WebSphere Business Integration wrapper" on page 119

Related tasks:

- “Editing the Oracle genclntsh script and creating the libclntsh file after you install DB2 Information Integrator (HP-UX, Linux, Solaris)” in the *IBM DB2 Information Integrator Installation Guide for Linux, UNIX, and Windows*
- “Registering the WebSphere Business Integration wrapper” on page 126
- “Registering the server definition for business application data sources” on page 127
- “Registering nicknames for business application data sources” on page 129
- “Creating federated views for business application nicknames” on page 137
- “Checking the setup of the federated server” on page 37
- “Creating a federated database” on page 51
- “Registering the WebSphere Business Integration wrapper” on page 126
- “Registering the server definition for business application data sources” on page 127
- “Creating federated views for business application nicknames” on page 137
- “Configuring the WebSphere Business Integration Adapters” on page 122
- “Registering nicknames for business application data sources” on page 129

Related reference:

- “Supported operating systems for DB2 Information Integrator (32-bit)” in the *IBM DB2 Information Integrator Installation Guide for Linux, UNIX, and Windows*

Registering the WebSphere Business Integration wrapper

Registering the WebSphere Business Integration wrapper is part of the larger task of adding business application data sources to a federated server.

You must register a wrapper to access business application data sources. Wrappers are used by federated servers to communicate with and retrieve data from data sources. Wrappers are implemented as a set of library files. You can register the wrapper by using the DB2 Control Center or the DB2 command line. The DB2 Control Center includes a wizard to guide you through the steps that are required to register the wrapper.

Prerequisites:

1. Install and configure the appropriate adapter.
2. Install and configure WebSphere MQ Version 5.3 (CSD level 5).
3. Create the WebSphere MQ message queues.
4. If the WebSphere MQ manager is not installed on the same system as DB2 Information Integrator, install the WebSphere MQ Version 5.3 (CSD level 5) client on the same system on which you installed a DB2 Information Integrator server instance.

Procedure:

To register a wrapper, issue the CREATE WRAPPER statement with the name of the wrapper and the name of the wrapper library file.

For example, to register a wrapper with the name `wbi_wrapper` on the federated server that uses the Windows operating system, issue the following statement:

```
CREATE WRAPPER wbi_wrapper LIBRARY 'db2wbi.dll';
```

The name of the wrapper library file that you specify depends on the operating system of the federated server. See the list of WebSphere Business Integration wrapper library files for the correct name to specify in the CREATE WRAPPER statement.

The next task in this sequence of tasks is registering the server definitions for the WebSphere Business Integration wrapper.

Related concepts:

- "The WebSphere Business Integration wrapper" on page 119

Related tasks:

- "Adding business application data sources to a federated system" on page 125

Related reference:

- "WebSphere Business Integration wrapper library files" on page 127
- "WebSphere Business Integration wrapper library files" on page 127

WebSphere Business Integration wrapper library files

The following table lists the directory paths and library file names for the WebSphere Business Integration wrapper.

When you install DB2 Information Integrator, 3 library files are added to the directory path listed in the table. For example, if the federated server is running on AIX, the wrapper library files added to the directory path are `libdb2wbi.a`, `libdb2wbiF.a`, and `libdb2wbiU.a`.

When you register a wrapper, specify only the library file name that is listed in the table.

Table 31. WebSphere Business Integration wrapper library locations and file names

Operating system	Directory path	Wrapper library file
AIX	/usr/opt/db2_08_01/lib/	libdb2wbi.a
Windows	%DB2PATH%\bin	db2wbi.dll

%DB2PATH% is the environment variable that is used to specify the directory path where DB2 Information Integrator is installed on Windows. The default Windows directory path is `C:\Program Files\IBM\SQLLIB`.

Related tasks:

- "Registering the WebSphere Business Integration wrapper" on page 126

Registering the server definition for business application data sources

Registering the server definition for a business object data source is part of the larger task of adding a business object application to a federated system.

After you register the wrapper, you must register a corresponding server.

Restrictions:

You can specify the option MQ_SVRCONN_CHANNELNAME only if you specify the option MQ_CONN_NAME. You cannot drop the option MQ_CONN_NAME until you drop option MQ_SVRCONN_CHANNELNAME. If MQ_CONN_NAME is not specified, the federated system uses the value of the MQSERVER environment variable. Set the MQSERVER environment variable in the db2dj.ini file. If you edit the db2dj.ini file, you must stop DB2 Universal Database and then restart it.

Procedure:

To register the server definition for a business application to the federated system, issue the CREATE SERVER statement.

For example, to register a server definition for the Siebel business applications:

```
CREATE SERVER siebel_server
  VERSION 2.4
  WRAPPER wbi_wrapper
  OPTIONS ( App_Type 'siebel',
    Request_Queue 'myqueue3',
    Response_Queue 'myqueue4',
    Fault_Queue 'myqueue5',
    MQ_Manager 'mymq'
    MQ_REPONSE_TIMEOUT '55000',
    MQ_CONN_NAME '9.30.76.151(1420)',
    MQ_SVRCONN_CHANNELNAME 'SYSTEM.DEF.SVRCONN'
  )
```

In the example, the business application is a Siebel application, which is identified with the APP_TYPE option. The valid values are SIEBEL, PSOFT, and SAP. The VERSION option represents the version of the WebSphere Business Integration Adapters that you are using. Valid values are 2.3 and 2.4. The server options must include the queue definitions as described in the topic Configuring the WebSphere Business Integration Adapters. The default value for MQ_RESPONSE_TIMEOUT is set to 50000 milliseconds. A value of −1 specifies that there is no timeout limit.

The next task in this sequence of tasks is registering the nicknames for business application data sources.

Related concepts:
- "The WebSphere Business Integration wrapper" on page 119

Related tasks:
- "Configuring the WebSphere Business Integration Adapters" on page 122
- "Adding business application data sources to a federated system" on page 125
- "Registering the WebSphere Business Integration wrapper" on page 126
- "Registering nicknames for business application data sources" on page 129

Related reference:
- Appendix D, "Server options for federated systems," on page 575
- "CREATE NICKNAME statement – examples for the WebSphere Business Integration wrapper" on page 138

Registering nicknames for business application data sources

Registering nicknames for business application data sources

Registering nicknames for business application data sources is part of the larger task of adding business applications to a federated system.

You can register the nickname by using the DB2 Control Center or the DB2 command line. The DB2 Control Center includes a wizard to guide you through the steps that are required to register the nickname.

You must create nicknames that correspond to the structural hierarchy of your business object definition. Parent nicknames contain at least one child nickname. Child nicknames correspond to the elements that contain a cardinality greater than 1 that are nested within the element for the parent nickname.

Procedure:

To register nicknames for business application data sources from the DB2 command line, issue a CREATE NICKNAME statement.

For example, to register a nickname for a Siebel business object definition that is called sieb_ssa_Contact_Contact, issue the following statement:

```
CREATE NICKNAME sieb_ssa_Contact_Contact_NN(
 Id VARCHAR(15)  OPTIONS(XPATH './ns1:Id/text()',
                            TEMPLATE '<ns1:Id>&column</ns1:Id>'),
 FirstName VARCHAR(50) OPTIONS(XPATH './ns1:FirstName/text()',
             TEMPLATE '<ns1:FirstName>&column</ns1:FirstName>'),
 LastName VARCHAR(50)  OPTIONS(XPATH './ns1:LastName/text()',
             TEMPLATE '<ns1:LastName>&column</ns1:LastName>'),
 AccountId VARCHAR(255)  OPTIONS(XPATH './ns1:AccountId/text()'),
 PrimaryAccountName VARCHAR(100)
      OPTIONS(XPATH './ns1:PrimaryAccountName/text()'),
 PrimaryPostalCode VARCHAR(30)
      OPTIONS(XPATH './ns1:PrimaryPostalCode/text()'),
 PrimaryStreetAddress VARCHAR(200)
      OPTIONS(XPATH './ns1:PrimaryStreetAddress/text()'),
 SalesRep VARCHAR(255)  OPTIONS(XPATH './ns1:SalesRep/text()'),
 State VARCHAR(255)  OPTIONS(XPATH './ns1:State/text()'))
  FOR SERVER siebel_server
  OPTIONS(XPATH '//ns1:sieb_ssa_Contact_Contact',
  TEMPLATE '<ns1:sieb_ssa_Contact_Contact>
                   &Id[1,1] &FirstName[0,1] &LastName[0,1]
             </ns1:sieb_ssa_Contact_Contact>',
  BUSOBJ_NAME 'sieb_ssa_Contact_Contact',
  NAMESPACES  'ns1="http://www.ibm.com/websphere/
                crossworlds/2002/BOSchema/
                  sieb_ssa_Contact_Contact"');
```

The BUSOBJ_NAME nickname option is the name of the XML schema definition (XSD) file that represents the business object definition.

Required step: flagging required input column in the XSD files

The WebSphere Business Integration Adapters can return only a single business object instance in response to a retrieve request. If a retrieve request in the form of an input business object to the adapter identifies more than a single business object in the application, the adapter returns only the first business object. The wrapper issues an error that indicates it could not retrieve the full result set.

To ensure that only a single business object is identified in response to a retrieve request, sufficient predicates must be provided to the adapter in the request business object. The wrapper must send all of the input predicates that are necessary for the identification of a single business object. Therefore, the columns must be identified in the nickname definitions by using the correct template references. The following steps describe the actions you must perform to identify the correct required input columns before using the DB2 Control Center to generate nickname definitions:

1. Identify the columns in the SAP, Siebel, or PeopleSoft application repository that represent a unique key for the application entity being mapped.

   SAP You can use the SAP Business Object Repository to identify the required input parameters for the BAPI that is being mapped to a WebSphere Business Integration business object definition by the WebSphere Business Integration Object Discovery Agent tool.

   Siebel Use one of the following approaches:

   - The Siebel application has a unique identifier column associated with each Business Component and generates hexadecimal values for this column for each instance of the Business Component. This identifier column exists at the highest level of the Business Component hierarchy and is already flagged by the isKey="true" specification (in the appSpecificInfo section of the xml annotation) for the element in the generated xsd file.
   - You can use Siebel tools to identify the database columns that represent a composite unique key for the Business Component that is being mapped. These columns must all be at the highest or root level of the business object hierarchy.

   PeopleSoft
   Use the Application Designer tool to identify the getKey columns in the Component Interface for the highest level of the hierarchy that is being mapped to a WebSphere Business Integration business object definition.
2. Edit the XSD files that are generated for the business object definition by the WebSphere Business Integration Object Discovery Agent tool to flag the required input columns. The guidelines for flagging the columns are located in topic The TEMPLATE option at the nickname and column levels.
3. Generate the nickname DDL for the business object definition from the DB2 Control Center.

To register nicknames for business application data sources from the DB2 Control Center:

1. Expand the **Federated Database Objects** folder.
2. Expand the wrapper folder for which you want to register nicknames.
3. Expand the **Server Definitions** folder.
4. Expand the server folder for which you want to register nicknames.
5. Right click the **Nicknames** folder and select **Create**.
6. In the Create Nicknames window, click **Discover** to define search criteria to help you select objects at the data source.
7. Specify the XML schema definition file that contains the definition of the business objects that you want DB2 Information Integrator users to access.
8. Click **OK** to create the nickname according to the selected XML schema definition file.

The DB2 Control Center extracts the schema file into multiple create nickname DDL statements, with the appropriate parent-child relationship definitions. The nicknames that are created represent the business object hierarchy that is defined in the XML schema definition file.

Optional: The next task in this sequence of tasks is creating federated views for the business application nicknames.

Related concepts:
- "The TEMPLATE option at the nickname and column levels" on page 131
- "Business object definitions" on page 121
- "The WebSphere Business Integration wrapper" on page 119
- "The TEMPLATE option at the nickname and column levels" on page 131
- "Nicknames and XPATH expressions" on page 136

Related tasks:
- "Specifying nickname columns for a nonrelational data source" on page 65
- "Adding business application data sources to a federated system" on page 125
- "Registering the WebSphere Business Integration wrapper" on page 126
- "Creating federated views for business application nicknames" on page 137

Related reference:
- "Business application data sources – example queries" on page 155
- "CREATE NICKNAME statement – examples for the WebSphere Business Integration wrapper" on page 138
- "Query restrictions for wrappers for business applications and Web services" on page 151

The TEMPLATE option at the nickname and column levels

This topic applies to the WebSphere® Business Integration wrapper and the Web services wrapper.

The WebSphere® Business Integration wrapper and the Web services wrapper build XML documents that are required by the WebSphere Business Integration Adapter and the Web services environment. The wrappers need the nickname level and the column level template fragments, which is the TEMPLATE option on the CREATE NICKNAME statement, at the time that the nickname is created. The wrappers use this information during the query planning and the query execution phases.

Web services wrapper

For the Web services wrapper, the required and optional attributes vary according to the definitions in the WSDL document and how a column is derived. A column can be derived from either an element or an attribute of an element.
- If the column is derived from an element, then the minOccurs value determines if a column is optional.
- If the value of minOccurs equals 0, then the column is optional.
- If the value of minOccurs equals 1, then the column is required.
- If the column is derived from an attribute of an element, then the value of use on the attribute determines if a column is optional.
- If an attribute contains the value use=optional, then the column is optional.

- If an attribute contains the value use=required, then the column is required.

The following example is an attribute in a schema definition that is associated with a column:

```
<xsd:complexType>
  <xsd:sequence>
    <xsd:element ref="tns:ZooName"/>
    <xsd:element ref="tns:Count"/>
    <xsd:element ref="tns:LastModified"/>
    <xsd:element maxOccurs="unbounded" minOccurs="0" ref="tns:Zookeeper"/>
  </xsd:sequence>
  <xsd:attribute name="id" type="xsd:string" use="optional"/>
</xsd:complexType>
```

WebSphere Business Integration wrapper

For the WebSphere Business Integration wrapper, the required and optional columns vary according to the application and the associated adapter. You need to identify the required and optional input columns by specifying the appropriate template option values for those columns. Before you use the DB2® Control Center to create the nicknames, you must modify the XML schema definition file to flag the required and optional input columns.

SAP BAPI

The IBM® DB2 Control Center determines the required and optional input columns based on the value of specific flags in the XML schema definition (XSD) files that represent the business object definition

In the annotation section of an element at any level of the business object hierarchy (parent or child business objects), an `I` prefix in the appSpecificInfo value indicates an import parameter for the SAP BAPI to which the business object definition maps. An `E` prefix indicates an export parameter for the SAP BAPI. Some elements can be both import and export parameters for a BAPI. The following example shows an element which is both an import and an export parameter:

```
<bx:appSpecificInfo>ICOMPANYCODE:ECOMPANYCODE</bx:appSpecificInfo>
```

The prefixes are generated automatically by the WebSphere Business Integration Object Discovery Agent tool based on information that is extracted from the SAP business object repository.

If an element that represents an import parameter (an `I` prefix in the appSpecificInfo value) is specified with the attribute `minOccurs=1`, the DB2 Control Center identifies the element as a required input parameter and flags the elements as a required input column in the nickname definition. The WebSphere Business Integration Object Discovery Agent tool does not automatically set the value of `minOccurs` to 1 for the required input parameters of the SAP BAPI. You must reference the SAP Business Object Repository to determine all the required input parameters for the BAPI that you want to access. Then, you must edit the corresponding elements in the XML schema file by manually setting the attribute to `minOccurs=1`. If the `minOccurs` attribute value for an input parameter remains as the default value of 0, then the DB2 Control Center specifies the column as an optional input column in the nickname hierarchy that is generated.

The following example shows an optional input column:

```
<xsd:element name="Company_code" minOccurs="0">
  <xsd:annotation>
   <xsd:appinfo>
     <bx:boAttribute>
       <bx:appSpecificInfo>ICOMPANYCODE:</bx:appSpecificInfo>
       <bx:attributeInfo isForeignKey="false" isKey="true" />
     </bx:boAttribute>
   </xsd:appinfo>
  </xsd:annotation>
  <xsd:simpleType>
   <xsd:restriction base="xsd:string">
     <xsd:maxLength value="4" />
   </xsd:restriction>
  </xsd:simpleType>
</xsd:element>
```

The following example shows a required input column:

```
<xsd:element name="Company_id" minOccurs="1">
  <xsd:annotation>
   <xsd:appinfo>
     <bx:boAttribute>
       <bx:appSpecificInfo>ICOMPANYID:</bx:appSpecificInfo>
       <bx:attributeInfo isForeignKey="true" isKey="true" />
     </bx:boAttribute>
   </xsd:appinfo>
  </xsd:annotation>
  <xsd:simpleType>
   <xsd:restriction base="xsd:string">
     <xsd:maxLength value="4" />
   </xsd:restriction>
  </xsd:simpleType>
</xsd:element>
```

The required and optional input columns for SAP business applications are designated by the syntax shown in the following table:

Table 32. Flagging schema for SAP input column information

Flags used in SAP XSD files	Required input column	Column reference in nickname template
Any element anywhere in the hierarchy with the prefix = 'I' and minOccurs=1	Yes	&columnname[1,1]
Any element anywhere in the hierarchy with the prefix = 'I' and minOccurs=0	No	&columnname[0,1]

Siebel and PeopleSoft

The DB2 Control Center determines the required and optional input columns based on the existence and the value of the `isRequired` flag in the attributeInfo section of the annotation for the element. If there is no isRequired flag, then the column is not an input column. The WebSphere Business Integration Object Discovery Agent tool does not automatically generate these flags in the XSD file. You must identify the required and optional input columns, and flag them appropriately in the XSD file before you use the DB2 Control Center to generate the nickname DDL.

The following example shows the flags for a required input column and optional input columns in the XSD file for a Siebel or PeopleSoft business object definition.

```
<xsd:element name="sieb_ssa_Contact_Contact">
 <xsd:annotation>
  <xsd:appinfo>
   <bx:boDefinition version="1.0.0">
   <bx:appSpecificInfo>ON=Contact;CN=Contact</bx:appSpecificInfo>
   </bx:boDefinition>
  </xsd:appinfo>
 </xsd:annotation>
<xsd:complexType>
<xsd:sequence>
<xsd:element name="Id" minOccurs="0">
 <xsd:annotation>
  <xsd:appinfo>
   <bx:boAttribute>
    <bx:appSpecificInfo>FN=Id</bx:appSpecificInfo>
    <bx:attributeInfo isForeignKey="false"
       isKey="true" isRequired="true" />
   </bx:boAttribute>
  </xsd:appinfo>
 </xsd:annotation>
...
</xsd:element>

...
```

Figure 11. Portion of a Siebel business object definition (Part 1 of 2)

```
...
<xsd:element name="FirstName" minOccurs="1">
<xsd:annotation>
 <xsd:appinfo>
  <bx:boAttribute>
  <bx:appSpecificInfo>FN=First Name</bx:appSpecificInfo>
  <bx:attributeInfo isForeignKey="false" isKey="false"
        isRequired="false" />
  </bx:boAttribute>
 </xsd:appinfo>
</xsd:annotation>
<xsd:simpleType>
 <xsd:restriction base="xsd:string">
  <xsd:maxLength value="50" />
 </xsd:restriction>
</xsd:simpleType>
</xsd:element>
<xsd:element name="LastName" minOccurs="1">
<xsd:annotation>
 <xsd:appinfo>
  <bx:boAttribute>
   <bx:appSpecificInfo>FN=Last Name</bx:appSpecificInfo>
   <bx:attributeInfo isForeignKey="false" isKey="false"
         isRequired="false" />
  </bx:boAttribute>
 </xsd:appinfo>
</xsd:annotation>
...
```

Figure 11. Portion of a Siebel business object definition (Part 2 of 2)

The required and optional input columns for Siebel and PeopleSoft business applications are designated by the syntax shown in the following table:

Table 33. Flagging schema for Siebel and PeopleSoft input column information

Flags used in Siebel and PeopleSoft XSD files	Required input column	Column reference in nickname template
isRequired="true"	Yes	&columnname[1,1]
isRequired="false"	No	&columnname[0,1]

The following example shows the DDL that the DB2 Control Center creates based on the XSD file that is shown in the figure labeled *Portion of a Siebel business object definition*. The XSD file in that figure included a value of false for the isRequired attribute.

```
CREATE NICKNAME sieb_ssa_Contact_Contact_NN(
 Id VARCHAR(15)  OPTIONS(XPATH './ns1:Id/text()',
           TEMPLATE '<ns1:Id>&column</ns1:Id>'),
 FirstName VARCHAR(50)  OPTIONS(XPATH './ns1:FirstName/text()',
           TEMPLATE '<ns1:FirstName>&column</ns1:FirstName>'),
 LastName VARCHAR(50)  OPTIONS(XPATH './ns1:LastName/text()',
           TEMPLATE '<ns1:LastName>&column</ns1:LastName>'),
 AccountId VARCHAR(255)  OPTIONS(XPATH './ns1:AccountId/text()'),
 PrimaryAccountName VARCHAR(100)
           OPTIONS(XPATH './ns1:PrimaryAccountName/text()'),
 PrimaryPostalCode VARCHAR(30)
           OPTIONS(XPATH './ns1:PrimaryPostalCode/text()'),
 PrimaryStreetAddress VARCHAR(200)
           OPTIONS(XPATH './ns1:PrimaryStreetAddress/text()'),
 SalesRep VARCHAR(255)  OPTIONS(XPATH './ns1:SalesRep/text()'),
 State VARCHAR(255)  OPTIONS(XPATH './ns1:State/text()'))
  FOR SERVER siebel_server
  OPTIONS(XPATH '//ns1:sieb_ssa_Contact_Contact',
  TEMPLATE '<ns1:sieb_ssa_Contact_Contact>
               &Id[1,1] &FirstName[0,1] &LastName[0,1]
            </ns1:sieb_ssa_Contact_Contact>',
  BUSOBJ_NAME 'sieb_ssa_Contact_Contact',
  NAMESPACES  'ns1="http://www.ibm.com/websphere/
               crossworlds/2002/BOSchema/sieb_ssa_Contact_Contact"');
```

Related concepts:

- “The WebSphere Business Integration wrapper” on page 119
- “The Web services wrapper and the Web services description language document” on page 381

Related tasks:

- “Adding business application data sources to a federated system” on page 125
- “Adding Web services data sources to a federated server” on page 387
- “Registering nicknames for Web services data sources” on page 390

Related reference:

- “Business application data sources – example queries” on page 155
- “CREATE NICKNAME statement – examples for the WebSphere Business Integration wrapper” on page 138
- “CREATE NICKNAME statement – examples for the Web services wrapper” on page 398
- “Messages for the Web services wrapper” on page 411
- “Web services data sources – example queries” on page 408

- "Query restrictions for wrappers for business applications and Web services" on page 151

Nicknames and XPATH expressions

This topic applies to the WebSphere® Business Integration wrapper and the Web services wrapper.

Nicknames correspond to the tree structure of your XML document data. Parent nicknames and child nicknames correspond to the root structure and nested elements of the data tree structure. These parent and child nicknames are connected by primary and foreign keys that are specified with the CREATE NICKNAME statement.

Each nickname is defined by XPath expressions that represent output values. The WebSphere Business Integration wrapper and the Web services wrapper use XPath expressions to establish a correspondence between the data in an XML document and the rows in a relational table. These XPath expressions identify the values in the XML document and determine how these values correspond to the columns of each row. The WebSphere Business Integration wrapper and the Web services wrapper read the XML document data only. The wrappers do not update the data. The XPATH option contains the information to find the SOAP messages through the SOAP envelope and SOAP body tags. The getQuote message is contained in the SOAP envelope and body elements.

The NICKNAME option XPATH expression points to repeating tags that are in the output element. The XPath expression determines how many or which rows will be in the nickname. The column option XPATH expression is relative to the NICKNAME XPATH expression. The column option XPATH identifies the values in a row. A NICKNAME option XPATH in a child nickname is relative to a NICKNAME option XPATH expression in a parent nickname.

When you create a nickname, you choose options that specify the association between the nickname and the XML document. Nicknames created for WebSphere Business Integration wrappers are associated with an XML schema definition (XSD) document. Nicknames that are created for Web services wrappers are associated with a Web services description language (WSDL) document.

Related concepts:
- "What is XML?" on page 415
- "The Web services wrapper and the Web services description language document" on page 381
- "Data associations between nicknames and XML documents" on page 422

Related tasks:
- "Adding XML to a federated system" on page 418
- "Registering nicknames for XML data sources" on page 424
- "Creating federated views for nonroot nicknames (XML wrapper)" on page 430
- "Adding business application data sources to a federated system" on page 125
- "Registering nicknames for business application data sources" on page 129
- "Adding Web services data sources to a federated server" on page 387
- "Registering nicknames for Web services data sources" on page 390

Related reference:

- "Business application data sources – example queries" on page 155
- "CREATE NICKNAME statement – examples for the WebSphere Business Integration wrapper" on page 138
- "CREATE NICKNAME statement – examples for the Web services wrapper" on page 398
- "Web services data sources – example queries" on page 408
- "Query restrictions for wrappers for business applications and Web services" on page 151
- "CREATE NICKNAME statement - Examples for XML wrapper" on page 425

Creating federated views for business application nicknames

Creating federated views for business application nicknames is part of the larger task of adding business applications to a federated system.

You can define federated views over the hierarchy of nicknames that describe a business object hierarchy. Defining federated views ensures that the queries that join pieces of the business application nickname hierarchy run properly.

Procedure:

To create federated views for business application nicknames:

1. Define a view for each business application nickname as a join of all the nicknames on the path to the parent nickname.
2. In the WHERE clause of the view, define the PRIMARY_KEY and FOREIGN_KEY columns as the join predicates.
3. In the SELECT list of the view, include all of the columns of the business application nickname except the column that is designated with the FOREIGN_KEY nickname column option. Do not include the columns in the SELECT list that are designated as PRIMARY_KEY in the parent nicknames along the hierarchy
4. Include the required input columns for the hierarchy in the select list. These columns might belong to some other nickname in the hierarchy.

The following example shows a view that is based on nicknames that are generated from a business object. The WHERE clause contains the primary and foreign keys that are defined in a CREATE NICKNAME statement.

```
CREATE VIEW view1 (
   customer, bankkey, bankact, customerno )
 AS (SELECT b.customer, b.bank_key, b.bank_acct,
     a.customerno
  FROM sap_bapi_customer_getdetail2_NN a,
       sap_bapi_customer_getdetail2_sap_customerbankdetail_NN b
  WHERE a.NN__PKEY=b.NN__FKEY);
```

Queries that use view view1 must include predicate values for the required column, such as in the following example:

```
SELECT * FROM view1
  WHERE customerno='1234567890';
```

There are no further tasks in this sequence of tasks.

Related tasks:

- "Adding business application data sources to a federated system" on page 125

Related reference:

- "Business application data sources – example queries" on page 155

CREATE NICKNAME statement – examples for the WebSphere Business Integration wrapper

Example 1: Flat business object

Figure 12 on page 139 is a portion of an xsd file that represents a WebSphere Business Integration business object definition for a Siebel Business Component. The business object definition hierarchy consists of a single level, which contains only the root business object. The DB2 Control Center creates a single relational nickname to represent this business object definition.

In the xsd file, the ID element is flagged as a required input column by adding the isRequired="true" flag in the annotation section for the element. The FirstName and LastName columns are designated as optional input columns by adding the isRequired="false" flag.

```
<?xml version="1.0" encoding="utf-8" standalone="no"?>
<xsd:schema elementFormDefault="qualified"
  targetNamespace="http://www.ibm.com/websphere/
         crossworlds/2002/BOSchema/sieb_ssa_Contact_Contact"
  ...
 <xsd:element name="sieb_ssa_Contact_Contact">
  <xsd:annotation>
    <xsd:appinfo>
     <bx:boDefinition version="1.0.0">
     <bx:appSpecificInfo>ON=Contact;CN=Contact</bx:appSpecificInfo>
     </bx:boDefinition>
    </xsd:appinfo>
   </xsd:annotation>
<xsd:complexType>
 <xsd:sequence>
   <xsd:element name="Id" minOccurs="0">
  <xsd:annotation>
   <xsd:appinfo>
    <bx:boAttribute>
     <bx:appSpecificInfo>FN=Id</bx:appSpecificInfo>
     <bx:attributeInfo isForeignKey="false"
                     isKey="true" isRequired="true" />
    </bx:boAttribute>
   </xsd:appinfo>
  </xsd:annotation>
 ...
 </xsd:simpleType>
</xsd:element>
<xsd:element name="FirstName" minOccurs="1">
  <xsd:annotation>
  ...
  <bx:appSpecificInfo>FN=First Name</bx:appSpecificInfo>
  <bx:attributeInfo isForeignKey="false"
      isKey="false" isRequired="false" />
 ...
  </xsd:annotation>
  ...
</xsd:element>
<xsd:element name="LastName" minOccurs="1">
  <xsd:annotation>
 ...
  <bx:appSpecificInfo>FN=Last Name</bx:appSpecificInfo>
  <bx:attributeInfo isForeignKey="false"
      isKey="false" isRequired="false" />
  ...
 </xsd:annotation>
...
</xsd:element>
```

Figure 12. XML schema file for a flat business object (Part 1 of 2)

```
<xsd:element name="AccountId" minOccurs="0">
 <xsd:annotation>
 ...
 <bx:appSpecificInfo>FN=Account Id</bx:appSpecificInfo>
 <bx:attributeInfo isForeignKey="false"
        isKey="false" />
 ...
  </xsd:annotation>
...
</xsd:element>
<xsd:element name="PrimaryAccountName" minOccurs="0">
 <xsd:annotation>
  ...
    <bx:appSpecificInfo>FN=Primary Account Name</bx:appSpecificInfo>
  <bx:attributeInfo isForeignKey="false" isKey="false" />
  ...
 </xsd:annotation>
 ...
</xsd:element>
<xsd:element name="PrimaryPostalCode" minOccurs="0">
 <xsd:annotation>
 ...
  <bx:appSpecificInfo>FN=Primary Postal Code</bx:appSpecificInfo>
  <bx:attributeInfo isForeignKey="false" isKey="false" />
 ...
 </xsd:annotation>
</xsd:element>
<xsd:element name="PrimaryStreetAddress" minOccurs="0">
 <xsd:annotation>
  ...
 <bx:appSpecificInfo>FN=Primary Street Address</bx:appSpecificInfo>
   <bx:attributeInfo isForeignKey="false" isKey="false" />
 ...
 </xsd:annotation>
...
</xsd:element>
<xsd:element name="SalesRep" minOccurs="0">
<xsd:annotation>
...
 <bx:boAttribute>
<bx:appSpecificInfo>FN=Sales Rep</bx:appSpecificInfo>
<bx:attributeInfo isForeignKey="false" isKey="false" />
 ...
 </xsd:annotation>
 ...
</xsd:element>
<xsd:element name="State" minOccurs="0">
 <xsd:annotation>
 ...
 <bx:boAttribute>
 <bx:appSpecificInfo>FN=State</bx:appSpecificInfo>
 <bx:attributeInfo isForeignKey="false" isKey="false" />
 ...
 </xsd:annotation>
...
</xsd:element>
<xsd:element name="ObjectEventId"
    type="xsd:string" minOccurs="0" />
</xsd:sequence>
...
</xsd:schema>
```

Figure 12. XML schema file for a flat business object (Part 2 of 2)

DB2 Control Center generates the following CREATE NICKNAME statement from the XSD file that is shown in Figure 12 on page 139. A TEMPLATE option value is specified for each input column. The column option templates are associated with the nickname option template. The input columns are also referenced in the nickname level TEMPLATE option value. The nickname option template provides the structure for the input business object. The value of the minOccurs attribute for each of the input column references in the nickname template value determines whether the input column is a required or optional column. The reference for the ID column is specified as &Id[1,1]. The references for the FirstName and LastName columns are specified as &FirstName [0,1] and &LastName [0,1]. All output columns include an XPATH column option value. The nickname is for a flat business object that does not contain any children (elements of cardinality 'n'):

```
CREATE NICKNAME sieb_ssa_Contact_Contact_NN(
 Id VARCHAR(15)  OPTIONS(XPATH './ns1:Id/text()',
     TEMPLATE '<ns1:Id>&column</ns1:Id>'),
 FirstName VARCHAR(50)  OPTIONS(XPATH './ns1:FirstName/text()',
     TEMPLATE '<ns1:FirstName>&column</ns1:FirstName>'),
 LastName VARCHAR(50)  OPTIONS(XPATH './ns1:LastName/text()',
     TEMPLATE '<ns1:LastName>&column</ns1:LastName>'),
 AccountId VARCHAR(255)
     OPTIONS(XPATH './ns1:AccountId/text()'),
 PrimaryAccountName VARCHAR(100)
     OPTIONS(XPATH './ns1:PrimaryAccountName/text()'),
 PrimaryPostalCode VARCHAR(30)
     OPTIONS(XPATH './ns1:PrimaryPostalCode/text()'),
 PrimaryStreetAddress VARCHAR(200)
     OPTIONS(XPATH './ns1:PrimaryStreetAddress/text()'),
 SalesRep VARCHAR(255)  OPTIONS(XPATH './ns1:SalesRep/text()'),
 State VARCHAR(255)  OPTIONS(XPATH './ns1:State/text()'))
  FOR SERVER siebel_server
  OPTIONS(XPATH '//ns1:sieb_ssa_Contact_Contact',
  TEMPLATE '<ns1:sieb_ssa_Contact_Contact>
               &Id[1,1] &FirstName[0,1] &LastName[0,1]
            </ns1:sieb_ssa_Contact_Contact>',
  BUSOBJ_NAME 'sieb_ssa_Contact_Contact',
  NAMESPACES  'ns1="http://www.ibm.com/websphere/
               crossworlds/2002/BOSchema/sieb_ssa_Contact_Contact"');
```

Example 2: Hierarchical business object

In this example, the business object definition consists of a two level hierarchy comprised of the root business object and two child business objects, or three xsd files. Only two nicknames are generated to represent the business object definition hierarchy. The sap_customeraddress child business object has a cardinality of 1, which is indicated by the absence of a maxOccurs attribute specification in the element definition. All of the columns of sap_customeraddress are included in the root nickname, sap_bapi_customer_getdetail2_NN. The sap_customerbankdetail child business object has a cardinality of n, which is indicated by the maxOccurs=″unbounded″ specification in the element definition. It is mapped to a separate child nickname, sap_bapi_customer_getdetail2_sap_customerbankdetail_NN. The child nickname is associated with the root nickname by a special primary key-foreign key relationship.

```
...
<xsd:element name="sap_bapi_customer_getdetail2">
  <xsd:annotation>
   <xsd:appinfo>
    <bx:boDefinition version="3.0.0" />
   </xsd:appinfo>
  </xsd:annotation>

...
<xsd:element name="COMPANYCODE" minOccurs="0">
 <xsd:annotation>
  <xsd:appinfo>
   <bx:boAttribute>
    <bx:appSpecificInfo>ICOMPANYCODE:</bx:appSpecificInfo>
    <bx:attributeInfo isForeignKey="false" isKey="true" />
   </bx:boAttribute>
  </xsd:appinfo>
</xsd:annotation>
...
</xsd:element>
<xsd:element name="CUSTOMERNO" minOccurs="1">
 <xsd:annotation>
  <xsd:appinfo>
   <bx:boAttribute>
    <bx:appSpecificInfo>ICUSTOMERNO:</bx:appSpecificInfo>
    <bx:attributeInfo isForeignKey="false" isKey="false" />
   </bx:boAttribute>
  </xsd:appinfo>
 </xsd:annotation>
...
</xsd:element>
```

Figure 13. SAP hierarchical business object: customer_getdetail2 (Part 1 of 2)

```
<xsd:element name="sap_customeraddress" minOccurs="0">
 <xsd:annotation>
  <xsd:appinfo>
   <bx:boAttribute>
    <bx:appSpecificInfo>:ECUSTOMERADDRESS</bx:appSpecificInfo>
    <bx:attributeInfo isForeignKey="false" isKey="false" />
    <bx:childObjectInfo relationship="Containment" version="3.0.0" />
   </bx:boAttribute>
  </xsd:appinfo>
 </xsd:annotation>
...
<xsd:element ref="sap_customeraddress:sap_customeraddress" />
</xsd:sequence>

</xsd:complexType>
</xsd:element>
<xsd:element name="sap_customerbankdetail" minOccurs="0">
 <xsd:annotation>
  <xsd:appinfo>
   <bx:boAttribute>
    <bx:appSpecificInfo>
        ICUSTOMERBANKDETAIL:ECUSTOMERBANKDETAIL
    </bx:appSpecificInfo>
    <bx:attributeInfo isForeignKey="false" isKey="false" />
    <bx:childObjectInfo relationship="Containment" version="3.0.0" />
   </bx:boAttribute>
  </xsd:appinfo>
 </xsd:annotation>
<xsd:complexType>
<xsd:sequence>
 <xsd:element ref="sap_customerbankdetail:sap_customerbankdetail"
     maxOccurs="unbounded" />
</xsd:sequence>
<xsd:attribute name="size" type="xsd:positiveInteger"
   default="1" />
</xsd:complexType>
</xsd:element>
<xsd:element name="ObjectEventId" type="xsd:string"
   minOccurs="0" />
</xsd:sequence>
...
 <xsd:annotation>
  <xsd:appinfo>
   <bx:boVerb>
    <bx:appSpecificInfo>
        bapi.client.Bapi_customer_getdetail2
    </bx:appSpecificInfo>
   </bx:boVerb>
  </xsd:appinfo>
 </xsd:annotation>
</xsd:enumeration>
<xsd:enumeration value="Update" />
</xsd:restriction>
</xsd:simpleType>
</xsd:attribute>
</xsd:complexType>
</xsd:element>
...
```

Figure 13. SAP hierarchical business object: customer_getdetail2 (Part 2 of 2)

```
<xsd:element name="sap_customeraddress">
 <xsd:annotation>
  <xsd:appinfo>
   <bx:boDefinition version="3.0.0">
    <bx:appSpecificInfo>:ECUSTOMERADDRESS</bx:appSpecificInfo>
   </bx:boDefinition>
  </xsd:appinfo>
 </xsd:annotation>
<xsd:complexType>
<xsd:sequence>
<xsd:element name="CUSTOMER" minOccurs="0">
 <xsd:annotation>
  <xsd:appinfo>
   <bx:boAttribute>
    <bx:appSpecificInfo>:ECUSTOMER</bx:appSpecificInfo>
    <bx:attributeInfo isForeignKey="false" isKey="true" />
   </bx:boAttribute>
  </xsd:appinfo>
 </xsd:annotation>
...
</xsd:element>
<xsd:element name="NAME" minOccurs="0">
 <xsd:annotation>
  <xsd:appinfo>
   <bx:boAttribute>
    <bx:appSpecificInfo>:ENAME</bx:appSpecificInfo>
    <bx:attributeInfo isForeignKey="false" isKey="false" />
   </bx:boAttribute>
  </xsd:appinfo>
 </xsd:annotation>
...
</xsd:element>
<xsd:element name="CITY" minOccurs="0">
 <xsd:annotation>
  <xsd:appinfo>
   <bx:boAttribute>
    <bx:appSpecificInfo>:ECITY</bx:appSpecificInfo>
    <bx:attributeInfo isForeignKey="false" isKey="false" />
   </bx:boAttribute>
  </xsd:appinfo>
 </xsd:annotation>
...
</xsd:element>
```

Figure 14. SAP hierarchical business object: customer_address (Part 1 of 2)

```
<xsd:element name="POSTL_CODE" minOccurs="0">
 <xsd:annotation>
  <xsd:appinfo>
   <bx:boAttribute>
    <bx:appSpecificInfo>:EPOSTL_CODE</bx:appSpecificInfo>
    <bx:attributeInfo isForeignKey="false" isKey="false" />
   </bx:boAttribute>
  </xsd:appinfo>
 </xsd:annotation>
...
</xsd:element>
<xsd:element name="STREET" minOccurs="0">
 <xsd:annotation>
  <xsd:appinfo>
   <bx:boAttribute>
    <bx:appSpecificInfo>:ESTREET</bx:appSpecificInfo>
    <bx:attributeInfo isForeignKey="false" isKey="false" />
   </bx:boAttribute>
</xsd:appinfo>
</xsd:annotation>
...
</xsd:element>
<xsd:element name="REGION" minOccurs="0">
 <xsd:annotation>
  <xsd:appinfo>
   <bx:boAttribute>
    <bx:appSpecificInfo>:EREGION</bx:appSpecificInfo>
    <bx:attributeInfo isForeignKey="false" isKey="false" />
   </bx:boAttribute>
  </xsd:appinfo>
 </xsd:annotation>
...
</xsd:element>
</xsd:sequence>
...
```

Figure 14. SAP hierarchical business object: customer_address (Part 2 of 2)

```
...
<xsd:element name= "sap_customerbankdetail ">
 <xsd:annotation>
  <xsd:appinfo>
   <bx:boDefinition version= "3.0.0 ">
    <bx:appSpecificInfo>
       ICUSTOMERBANKDETAIL:ECUSTOMERBANKDETAIL
    </bx:appSpecificInfo>
   </bx:boDefinition>
   </xsd:appinfo>
  </xsd:annotation>
<xsd:complexType>
<xsd:sequence>
<xsd:element name= "CUSTOMER " minOccurs= "0 ">
 <xsd:annotation>
  <xsd:appinfo>
   <bx:boAttribute>
    <bx:appSpecificInfo>ICUSTOMER:ECUSTOMER</bx:appSpecificInfo>
    <bx:attributeInfo isForeignKey= "false " isKey= "true " />
   </bx:boAttribute>
  </xsd:appinfo>
</xsd:annotation>
...
</xsd:element>
```

Figure 15. SAP hierarchical business object: bank_detail (Part 1 of 2)

```
<xsd:element name= "BANK_KEY " minOccurs= "0 ">
 <xsd:annotation>
  <xsd:appinfo>
   <bx:boAttribute>
    <bx:appSpecificInfo>IBANK_KEY:EBANK_KEY</bx:appSpecificInfo>
    <bx:attributeInfo isForeignKey= "false " isKey= "false " />
   </bx:boAttribute>
  </xsd:appinfo>
 </xsd:annotation>
...
</xsd:element>
<xsd:element name= "BANK_ACCT " minOccurs= "0 ">
 <xsd:annotation>
  <xsd:appinfo>
   <bx:boAttribute>
    <bx:appSpecificInfo>IBANK_ACCT:EBANK_ACCT</bx:appSpecificInfo>
    <bx:attributeInfo isForeignKey= "false " isKey= "false " />
   </bx:boAttribute>
  </xsd:appinfo>
 </xsd:annotation>
...
</xsd:element>
<xsd:element name= "CTRL_KEY " minOccurs= "0 ">
 <xsd:annotation>
  <xsd:appinfo>
   <bx:boAttribute>
   <bx:appSpecificInfo>ICTRL_KEY:ECTRL_KEY</bx:appSpecificInfo>
   <bx:attributeInfo isForeignKey= "false " isKey= "false " />
   </bx:boAttribute>
  </xsd:appinfo>
 </xsd:annotation>
...
</xsd:element>
<xsd:element name= "BANK_REF " minOccurs= "0 ">
 <xsd:annotation>
  <xsd:appinfo>
   <bx:boAttribute>
    <bx:appSpecificInfo>IBANK_REF:EBANK_REF</bx:appSpecificInfo>
    <bx:attributeInfo isForeignKey= "false " isKey= "false " />
   </bx:boAttribute>
  </xsd:appinfo>
 </xsd:annotation>
...
</xsd:element>
...
```

Figure 15. SAP hierarchical business object: bank_detail (Part 2 of 2)

DB2 Control Center generates two nicknames from the three SAP XSD files, as shown in Figure 16 on page 148 and Figure 17 on page 149. Column customerno is marked as a required input column in the nickname level template of the sap_bapi_customer_getdetail2_NN nickname because of the XSD file specifications for the customerno element. Customerno is flagged with an *I* prefix and a minOccurs=1 attribute value.

```
CREATE NICKNAME sap_bapi_customer_getdetail2_NN(
 CUSTOMER VARCHAR(10)
     OPTIONS(XPATH './ns3:sap_customeraddress/
           ns1:sap_customeraddress/ns1:CUSTOMER/text()'),
 NAME VARCHAR(35)
     OPTIONS(XPATH './ns3:sap_customeraddress/
           ns1:sap_customeraddress/ns1:NAME/text()'),
 CITY VARCHAR(35)
     OPTIONS(XPATH './ns3:sap_customeraddress/
           ns1:sap_customeraddress/ns1:CITY/text()'),
 POSTL_CODE VARCHAR(10)
     OPTIONS(XPATH './ns3:sap_customeraddress/
           ns1:sap_customeraddress/ns1:POSTL_CODE/text()'),
 STREET VARCHAR(35)
     OPTIONS(XPATH './ns3:sap_customeraddress/
           ns1:sap_customeraddress/ns1:STREET/text()'),
 REGION VARCHAR(3)
     OPTIONS(XPATH './ns3:sap_customeraddress/
           ns1:sap_customeraddress/ns1:REGION/text()'),
 NN__PKEY VARCHAR(16)  FOR BIT DATA NOT NULL OPTIONS(PRIMARY_KEY 'YES'),
 COMPANYCODE VARCHAR(4)  OPTIONS(XPATH './ns3:COMPANYCODE/text()',
       TEMPLATE '<ns3:COMPANYCODE>&column</ns3:COMPANYCODE>'),
 CUSTOMERNO VARCHAR(10)  OPTIONS(XPATH './ns3:CUSTOMERNO/text()',
       TEMPLATE '<ns3:CUSTOMERNO>&column</ns3:CUSTOMERNO>'),
 ObjectEventId VARCHAR(48)  OPTIONS(XPATH './ns3:ObjectEventId/text()'))
  FOR SERVER sap_server
  OPTIONS(XPATH '//ns3:sap_bapi_customer_getdetail2',
  TEMPLATE '<ns3:sap_bapi_customer_getdetail2>
                   &sap_bapi_customer_getdetail2_sap_customerbankdetail_NN[0,1]
                   &COMPANYCODE[0,1]
                   &CUSTOMERNO[1,1]
              </ns3:sap_bapi_customer_getdetail2>',
  BUSOBJ_NAME 'sap_bapi_customer_getdetail2',
  NAMESPACES  'ns1="http://www.ibm.com/websphere/
                  crossworlds/2002/BOSchema/sap_customeraddress",
                 ns2="http://www.ibm.com/websphere/
                  crossworlds/2002/BOSchema/sap_customerbankdetail",
                 ns3="http://www.ibm.com/websphere/
                  crossworlds/2002/BOSchema/sap_bapi_customer_getdetail2"');
```

Figure 16. getdetail2 nickname

```
CREATE NICKNAME sap_bapi_customer_getdetail2_sap_customerbankdetail_NN(
 CUSTOMER VARCHAR(10)  OPTIONS(XPATH './ns2:CUSTOMER/text()',
          TEMPLATE '<ns2:CUSTOMER>&column</ns2:CUSTOMER>'),
 BANK_KEY VARCHAR(15)  OPTIONS(XPATH './ns2:BANK_KEY/text()',
          TEMPLATE '<ns2:BANK_KEY>&column</ns2:BANK_KEY>'),
 BANK_ACCT VARCHAR(18)  OPTIONS(XPATH './ns2:BANK_ACCT/text()',
          TEMPLATE '<ns2:BANK_ACCT>&column</ns2:BANK_ACCT>'),
 CTRL_KEY VARCHAR(2)  OPTIONS(XPATH './ns2:CTRL_KEY/text()',
          TEMPLATE '<ns2:CTRL_KEY>&column</ns2:CTRL_KEY>'),
 BANK_REF VARCHAR(20)  OPTIONS(XPATH './ns2:BANK_REF/text()',
          TEMPLATE '<ns2:BANK_REF>&column</ns2:BANK_REF>'),
 NN__FKEY VARCHAR(16)  FOR BIT DATA NOT NULL
      OPTIONS(FOREIGN_KEY 'SAP_BAPI_CUSTOMER_GETDETAIL2_NN'))
  FOR SERVER sap_server
  OPTIONS(XPATH './ns3:sap_customerbankdetail/ns2:sap_customerbankdetail',
  TEMPLATE '<ns3:sap_customerbankdetail>
              <ns2:sap_customerbankdetail>
                &CUSTOMER[0,1]
                &BANK_KEY[0,1]
                &BANK_ACCT[0,1]
                &CTRL_KEY[0,1]
                &BANK_REF[0,1]
              </ns2:sap_customerbankdetail>
            </ns3:sap_customerbankdetail>',
  NAMESPACES  'ns1="http://www.ibm.com/websphere/
                  crossworlds/2002/BOSchema/sap_customeraddress",
               ns2="http://www.ibm.com/websphere/
                  crossworlds/2002/BOSchema/sap_customerbankdetail",
               ns3="http://www.ibm.com/websphere/
                  crossworlds/2002/BOSchema/sap_bapi_customer_getdetail2"');
```

Figure 17. Customer bank detail nickname

Example 3: Primary and foreign keys

The PRIMARY_KEY and FOREIGN_KEY columns are used to define relationships between the parent and child nicknames. Each parent nickname must have a primary key column option. You define the children of a parent nickname with the foreign key column option that references the primary key column of a parent nickname. A nickname can have multiple children, but a nickname can have only one parent.

Primary and foreign key values for the WebSphere Business Integration wrapper are only valid and unique within a single query. A primary and foreign key cannot be used to retrieve a row with a second query. The values cannot persist into another table, because the uniqueness of the value is not guaranteed if that table is populated with multiple queries.

The following CREATE NICKNAME statements are derived from the XML schema definition files that are shown in Figure 13 on page 142, Figure 14 on page 144, and Figure 15 on page 146. The foreign key, nn_fkey, uniquely associates the child nickname, sap_bapi_customer_getdetail2_sap_customerbankdetail_nn, with the parent nickname sap_bapi_customer_getdetail2_nn. The parent nickname also uses a reference to the child nickname in the nickname options template structure.

```
CREATE NICKNAME sap_bapi_customer_getdetail2_NN(
 CUSTOMER VARCHAR(10)
     OPTIONS(XPATH './ns3:sap_customeraddress/
         ns1:sap_customeraddress/ns1:CUSTOMER/text()'),
 NAME VARCHAR(35)
```

```
    OPTIONS(XPATH './ns3:sap_customeraddress/
        ns1:sap_customeraddress/ns1:NAME/text()'),
 ...
 NN__PKEY VARCHAR(16)  FOR BIT DATA NOT NULL
           OPTIONS(PRIMARY_KEY 'YES'),
...

TEMPLATE '<ns3:sap_bapi_customer_getdetail2>
           &sap_bapi_customer_getdetail2_sap_customerbankdetail_NN[0,1]
           &COMPANYCODE[0,1]
           &CUSTOMERNO[1,1]
         </ns3:sap_bapi_customer_getdetail2>',

...
CREATE NICKNAME sap_bapi_customer_getdetail2_sap_customerbankdetail_NN(
 CUSTOMER VARCHAR(10)  OPTIONS(XPATH './ns2:CUSTOMER/text()',
           TEMPLATE '<ns2:CUSTOMER>&column</ns2:CUSTOMER>'),
 BANK_KEY VARCHAR(15)  OPTIONS(XPATH './ns2:BANK_KEY/text()',
           TEMPLATE '<ns2:BANK_KEY>&column</ns2:BANK_KEY>'),
 BANK_ACCT VARCHAR(18)  OPTIONS(XPATH './ns2:BANK_ACCT/text()',
           TEMPLATE '<ns2:BANK_ACCT>&column</ns2:BANK_ACCT>'),
 CTRL_KEY VARCHAR(2)  OPTIONS(XPATH './ns2:CTRL_KEY/text()',
           TEMPLATE '<ns2:CTRL_KEY>&column</ns2:CTRL_KEY>'),
 BANK_REF VARCHAR(20)  OPTIONS(XPATH './ns2:BANK_REF/text()',
           TEMPLATE '<ns2:BANK_REF>&column</ns2:BANK_REF>'),
 NN__FKEY VARCHAR(16) FOR BIT DATA NOT NULL
        OPTIONS(FOREIGN_KEY 'SAP_BAPI_CUSTOMER_GETDETAIL2_NN'))
  FOR SERVER sap_server
  OPTIONS(XPATH './ns3:sap_customerbankdetail/ns2:sap_customerbankdetail',
  TEMPLATE '<ns3:sap_customerbankdetail>
              <ns2:sap_customerbankdetail>
                &CUSTOMER[0,1]
                &BANK_KEY[0,1]
                &BANK_ACCT[0,1]
                &CTRL_KEY[0,1]
                &BANK_REF[0,1]
              </ns2:sap_customerbankdetail>
            </ns3:sap_customerbankdetail>',

...
```

Example 4: Using namespaces to resolve XPath expression prefixes

The NAMESPACES option is a comma separated list of name-value pairs. It resolves the prefixes that are used in XPath expressions with the namespace URIs that are defined in the XML schemas. These XPath expressions are applied on the business objects (the XML document) that are returned from WebSphere Business Integration Adapter. The following example includes namespace prefixes and the definitions of those prefixes:

```
CREATE NICKNAME sap_customer
  (
sap_customeraddress_CUSTOMER VARCHAR(10)
  OPTIONS(XPATH './ns5:sap_customeraddress/
        ns2:sap_customeraddress/ns2:CUSTOMER/text()'),
sap_customeraddress_NAME VARCHAR(35)
  OPTIONS(XPATH './ns5:sap_customeraddress/
        ns2:sap_customeraddress/ns2:NAME/text()'),

...

sap_bapi_customer_getdet1_PKEY VARCHAR(16) FOR BIT DATA NOT NULL
  OPTIONS(PRIMARY_KEY 'YES'),
COMPANYCODE VARCHAR(4)
  OPTIONS(XPATH './ns5:COMPANYCODE/text()',
         TEMPLATE '<ns5:COMPANYCODE>&column</ns5:COMPANYCODE>'),
CUSTOMERNO VARCHAR(10)
  OPTIONS(XPATH './ns5:CUSTOMERNO/text()',
  TEMPLATE '<ns5:CUSTOMERNO>&column</ns5:CUSTOMERNO>'),
ObjectEventId VARCHAR(48)
  OPTIONS(XPATH './ns5:ObjectEventId/text()')
 )
FOR SERVER SAP_SOURCE
  OPTIONS (
  XPATH '//ns5:sap_bapi_customer_getdetail2',
  TEMPLATE
         '<ns5:sap_bapi_customer_getdetail2>
              &customerbankdetail_NN[0,1] &COMPANYCODE[0,1] &CUSTOMERNO[1,1]
          </ns5:sap_bapi_customer_getdetail2>',
  BUSOBJ_NAME 'sap_bapi_customer_getdetail2',
  NAMESPACES  '
ns2="http://www.ibm.com/websphere/
          crossworlds/2002/BOSchema/sap_customeraddress",
...
ns5="http://www.ibm.com/websphere/
          crossworlds/2002/BOSchema/sap_bapi_customer_getdetail2",
ns6="http://www.ibm.com/websphere/
           crossworlds/2002/BOSchema/sap_return"'
);
```

Related concepts:

- "The WebSphere Business Integration wrapper" on page 119

Related tasks:

- "Adding business application data sources to a federated system" on page 125
- "Registering nicknames for business application data sources" on page 129

Related reference:

- "Business application data sources – example queries" on page 155
- "Query restrictions for wrappers for business applications and Web services" on page 151

Query restrictions for wrappers for business applications and Web services

Equal predicates

The only valid predicates on input columns are equal predicates. For output columns, any predicate is valid.

The following example returns an error with a message that indicates that the predicate is not supported on that column. In this example, the column `zipcode` is an input column:

```
SELECT return FROM gettemp WHERE zipcode<'95141'
```

The following example shows a valid query using an equal predicate on the input columns. The customers nickname is joined with a local DB2 UDB table that contains customer IDs. The query contains an additional predicate on the Sales column, which is an output only column.

```
SELECT a.name, a.address
  FROM customers a, local_table b
  WHERE
    a.customer_id=b.custid AND
    a.Sales > 300000;
```

Predicates for required input columns

You must provide equality predicate values for all required input columns in your SQL queries for the nickname hierarchy that you reference. The wrapper returns an SQLCODE 901 for all queries that violate this restriction.

IN or OR predicates

For WebSphere Business Integration wrappers and Web services wrappers, no IN or OR predicates are allowed for input columns.

The following examples show invalid queries. The customers nickname has one required input column, customer_id:

```
SELECT * FROM customers
  WHERE customer_id IN (12345, 67890, 11223);
SELECT * FROM customers
  WHERE customer_id IN (SELECT custid FROM local_table);  )
```

However, for the WebSphere Business Integration wrappers, you can use IN list predicates with required input columns if you define a unique index with the SPECIFICATION ONLY parameter for the required input columns:

```
CREATE UNIQUE INDEX myuindex ON customers(customer_id) SPECIFICATION ONLY;
```

Joins on optional input columns

The following example demonstrates a restriction on joining optional input columns. You cannot join optional input columns from a local table or nickname. If the WSDL generates an input nickname column as optional and you need to use that column in a join, then you must edit the DDL to change the column to a required input column.

In this example, a Web service wrapper nickname named `order` is created with `shipping_method` as an optional input column. The following statement is a valid query because it uses a literal in the predicate:

```
SELECT * FROM order
  WHERE part="hammer" AND shipping_method="FEDEX";
```

However, if you include a local table named `orderparts`, which defines parts and shipping methods, in the query, and the table contains a column named `shipping_method` that is optional, the statement is invalid:

```
SELECT * FROM
   order o, orderparts op
  WHERE
     o.part="hammer" AND
     o.shipping_method=op.shipping_method
```

For a WebSphere Business Integration wrapper, predicates on optional input columns of a nickname might be pushed down to the WebSphere Business Integration Adapter. DB2 UDB can decide to apply those predicates locally on the rows fetched from the application data source. To ensure that predicates for a given input column always get pushed down to the adapter, declare the input column as a required input column. Every query on the nickname hierarchy must include predicate values for the required input columns.

To ensure valid results, joined input columns must be required columns for Web Services wrappers.

Outer joins

Outer joins between nicknames using the primary key from a parent nickname and the foreign key from child nickname columns are not supported.

When a parent element in an XML document contains no child elements, and if you use an inner join between the parent nickname and the child nickname, then no rows are returned for that element. For example, for a given customer, if there is no bankdetail information in the SAP system, then no rows are returned for the sap_bapi_customer_getdetail2_sap_customerbankdetail_NN nickname for the particular customer.

The following CREATE NICKNAME statements define the columns that are used in the example query:

```
CREATE NICKNAME sap_bapi_customer_getdetail2_NN(
 ...
 NAME VARCHAR(35)
     OPTIONS(XPATH './ns3:sap_customeraddress/
           ns1:sap_customeraddress/ns1:NAME/text()'),
 ...
 NN__PKEY VARCHAR(16)  OPTIONS(PRIMARY_KEY 'YES'),
 COMPANYCODE VARCHAR(4)  OPTIONS(XPATH './ns3:COMPANYCODE/text()',
       TEMPLATE '<ns3:COMPANYCODE>&column</ns3:COMPANYCODE>'),
 CUSTOMERNO VARCHAR(10)  OPTIONS(XPATH './ns3:CUSTOMERNO/text()',
       TEMPLATE '<ns3:CUSTOMERNO>&column</ns3:CUSTOMERNO>'),
 ...
  FOR SERVER sap_server
  OPTIONS(XPATH '//ns3:sap_bapi_customer_getdetail2',
  TEMPLATE '<ns3:sap_bapi_customer_getdetail2>
                 &sap_bapi_customer_getdetail2_sap_customerbankdetail_NN[0,1]
                 &COMPANYCODE[0,1]
                 &CUSTOMERNO[1,1]
            </ns3:sap_bapi_customer_getdetail2>',
  ...
```

Figure 18. Excerpt from getdetail2 nickname

```
CREATE NICKNAME sap_bapi_customer_getdetail2_sap_customerbankdetail_NN(
 CUSTOMER VARCHAR(10)  OPTIONS(XPATH './ns2:CUSTOMER/text()',
          TEMPLATE '<ns2:CUSTOMER>&column</ns2:CUSTOMER>'),
 BANK_KEY VARCHAR(15)  OPTIONS(XPATH './ns2:BANK_KEY/text()',
          TEMPLATE '<ns2:BANK_KEY>&column</ns2:BANK_KEY>'),
 BANK_ACCT VARCHAR(18)  OPTIONS(XPATH './ns2:BANK_ACCT/text()',
          TEMPLATE '<ns2:BANK_ACCT>&column</ns2:BANK_ACCT>'),
 CTRL_KEY VARCHAR(2)  OPTIONS(XPATH './ns2:CTRL_KEY/text()',
          TEMPLATE '<ns2:CTRL_KEY>&column</ns2:CTRL_KEY>'),
 BANK_REF VARCHAR(20)  OPTIONS(XPATH './ns2:BANK_REF/text()',
          TEMPLATE '<ns2:BANK_REF>&column</ns2:BANK_REF>'),
 NN__FKEY VARCHAR(16)  OPTIONS(FOREIGN_KEY 'SAP_BAPI_CUSTOMER_GETDETAIL2_NN'))
  FOR SERVER sap_server
  OPTIONS(XPATH './ns3:sap_customerbankdetail/ns2:sap_customerbankdetail',
  TEMPLATE '<ns3:sap_customerbankdetail>
               <ns2:sap_customerbankdetail>
                 &CUSTOMER[0,1]
                 &BANK_KEY[0,1]
                 &BANK_ACCT[0,1]
                 &CTRL_KEY[0,1]
                 &BANK_REF[0,1]
               </ns2:sap_customerbankdetail>
             </ns3:sap_customerbankdetail>',
 ...
```

Figure 19. Excerpt from customer bank detail nickname

In the following example, the query returns no rows because there is an inner join condition between the two nicknames:

```
SELECT a.name, b.bank_key
  FROM sap_bapi_customer_getdetail2_NN a,
     sap_bapi_customer_getdetail2_sap_customerbankdetail_NN b
  WHERE a.customerno='1234567890'
  AND a.NN__PKEY=b.NN__FKEY;
```

If a WebSphere Business Integration wrapper or a Web services wrapper nickname definition contains required input columns, then a left outer join between this nickname and any other local DB2 UDB table or other nicknames is not supported.

Related concepts:

- “The TEMPLATE option at the nickname and column levels” on page 131

Related tasks:

- “Adding business application data sources to a federated system” on page 125
- “Adding Web services data sources to a federated server” on page 387

Related reference:

- “Business application data sources – example queries” on page 155
- “CREATE NICKNAME statement – examples for the WebSphere Business Integration wrapper” on page 138
- “CREATE NICKNAME statement – examples for the Web services wrapper” on page 398
- “Web services data sources – example queries” on page 408

Business application data sources – example queries

Example 1: Joining parent and child nicknames

If a select statement contains child nickname columns, then you must specify a join predicate with the parent nickname. The join conditions are necessary to maintain the parent-child relationships along the nickname hierarchy. Specify the primary and foreign key join conditions for each parent-child nickname pair in the hierarchy by including the hierarchy association from the child nickname that is referenced to the parent nickname for the hierarchy.

The following queries are invalid because they do not contain all of the elements that are necessary to maintain the nickname hierarchy:

```
SELECT * FROM <child_nickname>;
SELECT b.col1
  FROM <parent_nickname> a,<child_nickname> b
  WHERE a.required_column=<value>;
```

The following is an example of a valid query that maintains the nickname hierarchy:

```
SELECT b.col1, a.cola
  FROM <parent_nickname> a,<child_nickname> b
  WHERE a.primary_key_column=b.foreign_key_column
   AND a.required_column=<value>;
```

In the following example, all of the required parent and child input columns are included in the predicates in the WHERE clause. The WHERE clause includes join predicates that specify an equality between the parent primary key column and a child foreign key column :

```
SELECT a.customer, a.name, b.bank_key, b.bank_acct
  FROM sap_bapi_customer_getdetail2_NN a,
       sap_bapi_customer_getdetail2_sap_customerbankdetail_NN b
   WHERE a.nn_pkey=b.nn_fkey
   AND a.customer = 'ABC'
```

Example 2: Views

You can create two types of views:

- A view that is derived from columns in a child nickname so that you can issue queries directly on the child nickname without including the parent-child join conditions in the queries.

  ```
  CREATE VIEW view1 (
     customer, bankkey, bankact, customerno )
   AS (SELECT b.customer, b.bank_key, b.bank_acct,
       a.customerno
    FROM sap_bapi_customer_getdetail2_NN a,
         sap_bapi_customer_getdetail2_sap_customerbankdetail_NN b
    WHERE a.NN__PKEY=b.NN__FKEY);
  ```

  The view definition includes the required input column, customerno, for the nickname hierarchy. Queries that use view view1 must include predicate values for the required column, such as in the following example:

  ```
  SELECT * FROM view1
    WHERE customerno='1234567890';
  ```

- A global view that includes all columns of the child and parent nicknames except for the primary and foreign key columns.

Example 3: Required input columns

Queries must include predicate values for all required input columns. Required input columns are those columns with TEMPLATE column options definitions and a reference value of [1,1] in the nickname TEMPLATE option value. The wrapper returns an error for any query that does not include the required input columns in the predicates.

The following example shows an invalid query. The customers nickname contains a required input column customer_id.

```
SELECT * FROM customers;
```

The following example shows a valid query.

```
SELECT * FROM customers WHERE customer_id = 123;
```

The following example shows a local table in DB2 UDB that contains customer IDs in the custid column of table local_table. This example is an inner join between the WebSphere Business Integration nickname and the local table.

```
SELECT a.name, a.address
  FROM customers a, local_table b
  WHERE a.customer_id=b.custid;
```

Related concepts:

- "The WebSphere Business Integration wrapper" on page 119

Related tasks:

- "Adding business application data sources to a federated system" on page 125
- "Registering nicknames for business application data sources" on page 129

Related reference:

- "CREATE NICKNAME statement – examples for the WebSphere Business Integration wrapper" on page 138
- "Query restrictions for wrappers for business applications and Web services" on page 151

Chapter 9. Configuring access to DB2 family data sources

This chapter explains how to configure your federated server to access data that is stored in DB2 family databases. These databases include:

- DB2 UDB for Linux, UNIX, and Windows
- DB2 UDB for z/OS and OS/390
- DB2 UDB for iSeries
- DB2 Server for VM and VSE

This chapter explains how to configure your federated server to access data that is stored in DB2 family data sources. You can configure access to DB2 family data sources by using the DB2 Control Center or by issuing SQL statements.

This chapter lists the tasks that you need to perform when you configure the federated server, contains examples of the SQL statements that you need, and provides tuning and troubleshooting information for configuring the federated server.

Adding DB2 family data sources to a federated server

Adding DB2 family data sources to a federated server

To configure the federated server to access DB2 family data sources, you must provide the federated server with information about the data sources and objects that you want to access.

| You can configure the federated server to access DB2 family data sources by using
| the DB2 Control Center or the DB2 command line. The DB2 Control Center
| includes a wizard to guide you through the steps required to configure the
| federated server.

Prerequisites:

| • A DB2 server that is configured for federation.
| • A federated database must exist on the federated server

Restriction:

You cannot create a nickname for a DB2 data source alias if you are accessing data that is stored in DB2 for Linux, UNIX and Windows, Version 8.1.

Procedure:

To add DB2 data sources to a federated server:

1. Catalog the node.
2. Catalog the remote database.
3. Register the wrapper.
4. Register the server definition.
5. Create the user mappings.
6. Test the connection to the DB2 server.

7. Register nicknames for DB2 tables and views.

Related concepts:

- "Fast track to configuring your data sources" on page 55

Related tasks:

- "Checking the FEDERATED parameter" on page 42
- "Creating a federated database" on page 51
- "Cataloging a node entry in the federated node directory" on page 158
- "Cataloging the remote database in the federated system database directory" on page 159
- "Registering the DB2 wrapper" on page 160
- "Registering the server definitions for a DB2 data source" on page 161
- "Creating the user mapping for a DB2 data source" on page 163
- "Testing the connection to the DB2 data source server" on page 165
- "Registering nicknames for DB2 tables and views" on page 166
- "Checking the setup of the federated server" on page 37

Related reference:

- "Supported operating systems for DB2 Information Integrator (32-bit)" in the *IBM DB2 Information Integrator Installation Guide for Linux, UNIX, and Windows*
- "Supported operating systems for DB2 Information Integrator (64-bit)" in the *IBM DB2 Information Integrator Installation Guide for Linux, UNIX, and Windows*

Cataloging a node entry in the federated node directory

Cataloging a node entry in the federated node directory is part of the larger task of adding DB2 family data sources to federated servers.

To point to the location of the DB2 data source, catalog an entry in the node directory of the federated server. The federated server uses this entry to determine the proper access method to connect to a DB2 data source.

Procedure:

To catalog a node entry in the federated node directory:

1. Determine the communication protocol that you will be using.
2. Issue the appropriate command to catalog the node entry.
   - If your communication protocol is Transmission Control Protocol/Internet Protocol (TCP/IP), issue the **CATALOG TCPIP NODE** command.

     For example:

     ```
     CATALOG TCPIP NODE DB2NODE REMOTE SYSTEM42 SERVER DB2TCP42
     ```

     The *DB2NODE* value is the name that you assign to the node that you are cataloging. REMOTE *SYSTEM42* is the host name of the system where the data source resides. SERVER *DB2TCP42* is the service name or primary port number of the server database manager instance. If a service name is used, it is case sensitive.
   - If your communication protocol is SNA, issue the **CATALOG APPC NODE** command.

     For example:

```
CATALOG APPC NODE DB2NODE REMOTE DB2CPIC SECURITY PROGRAM
```

The *DB2NODE* value is the name that you assign to the node that you are cataloging. REMOTE *DB2CPIC* is the SNA partner logical unit (LU) name of the remote partner node. SECURITY *PROGRAM* specifies that both a user name and a password are to be included in the allocation request that is sent to the partner LU.

The next task in this sequence of tasks is cataloging the remote database in the federated system database directory.

Related tasks:

- "Cataloging the remote database in the federated system database directory" on page 159

Cataloging the remote database in the federated system database directory

Cataloging the remote database in the federated system database directory is part of the larger task of adding DB2 family data sources to federated servers.

You specify which DB2 data source database that the federated server connects to by cataloging the remote database in the federated server system database directory,

Procedure:

To catalog the remote database in the federated server system database directory:

1. Use the Client Configuration Assistant (CCA).

   For federated servers on UNIX, you can alternatively use the **CATALOG DATABASE** command. For example:

   ```
   CATALOG DATABASE DB2DB390 AS CLIENTS390 AT NODE DB2NODE AUTHENTICATION DCS
   ```

   The value *DB2DB390* is the name of the remote database that you are cataloging in the federated server system database directory. AS *CLIENTS390* is the alias for the database being cataloged. If you do not specify an alias, the database manager uses the database name (for example DB2DB390) as the alias. AT NODE *DB2NODE* is the name of the node that you specified when cataloging the node entry in the node directory. AUTHENTICATION SERVER specifies that authentication takes place on the DB2 data source node.
2. If the name of the remote database is more than eight characters, you must create a DCS directory entry by issuing the **CATALOG DCS DATABASE** command. For example:

   ```
   CATALOG DCS DATABASE SALES400 AS SALES_DB2DB400
   ```

   The value *SALES400* is the alias of the remote database to catalog. This name should match the name of an entry in the federated server system database directory that is associated with the remote node. It is the same name you entered in the **CATALOG DATABASE** command. AS *SALES_DB2DB400* is the name of the target host database that you want to catalog.

The next task in this sequence of tasks is registering the DB2 wrapper.

Related tasks:

- "Cataloging a node entry in the federated node directory" on page 158
- "Registering the DB2 wrapper" on page 160

Registering the DB2 wrapper

Registering the DB2 wrapper is part of the larger task of adding DB2 family data sources to a federated server.

You must register a wrapper to access DB2 family data sources. Wrappers are used by federated servers to communicate with and retrieve data from data sources. Wrappers are implemented as a set of library files.

Procedure:

To register a wrapper, issue the CREATE WRAPPER statement and specify the default name for the wrapper.

For example:

```
CREATE WRAPPER DRDA
```

Recommendation: Use the default wrapper name called DRDA. When you register the wrapper by using the default name, the federated server automatically takes the default library name that is associated with that wrapper name.

If the wrapper name conflicts with an existing wrapper name in the federated database, you can substitute the default wrapper name with a name you choose. If you use a name that is different from the default name, you must include the LIBRARY parameter in the CREATE WRAPPER statement.

For example, to register a wrapper with the name `db2_wrapper` on the federated server that uses the AIX operating system, issue the following statement:

```
CREATE WRAPPER db2_wrapper LIBRARY 'libdb2drda.a'
```

The name of the wrapper library file that you specify depends on the operating system of the federated server. See the list of DB2 wrapper library files for the correct name to specify in the CREATE WRAPPER statement.

The next task in this sequence of tasks is registering the server definitions for the DB2 wrapper.

Related tasks:
- "Registering the server definitions for a DB2 data source" on page 161

Related reference:
- "DB2 wrapper library files" on page 160
- "CREATE WRAPPER statement" in the *SQL Reference, Volume 2*

DB2 wrapper library files

The following table lists the directory paths and library file names for the DB2 wrapper.

When you install DB2 Information Integrator, 3 library files are added to the directory path listed in the table. For example, if the federated server is running on AIX, the wrapper library files added to the directory path are `libdb2drda.a`, `libdb2drdaF.a`, and `libdb2drdaU.a`.

When you register a wrapper, specify only the library file name that is listed in the table.

Table 34. DB2 wrapper library locations and file names

Operating system	Directory path	Wrapper library file
AIX	/usr/opt/db2_08_01/lib/	libdb2drda.a
HP-UX	/opt/IBM/db2/V8.1/lib	libdb2drda.sl
Linux	/opt/IBM/db2/V8.1/lib	libdb2drda.so
Solaris	/opt/IBM/db2/V8.1/lib	libdb2drda.so
Windows	%DB2PATH%\bin	db2drda.dll

%DB2PATH% is the environment variable that is used to specify the directory path where DB2 Information Integrator is installed on Windows. The default Windows directory path is `C:\Program Files\IBM\SQLLIB`.

Related tasks:
- "Registering the DB2 wrapper" on page 160

Registering the server definitions for a DB2 data source

Registering the server definitions for a DB2 data source is part of the larger task of adding DB2 family data sources to federated servers.

In the federated database, you must define each DB2 server that you want to access. When you register the server definition, the federated server connects to the DB2 server and binds packages to the database. Because the information for authorization and password are not stored in the federated global catalog, you must include them in the server definition.

Procedure:

To register a server definition for a DB2 data source, issue the CREATE SERVER statement.

For example:

```
CREATE SERVER server_name TYPE DB2/ZOS VERSION 6 WRAPPER DRDA
       AUTHORIZATION "name1" PASSWORD "passwd1"
        OPTIONS (DBNAME 'db_name')
```

The name that you assign to a server must be unique. Duplicate server names are not allowed.

The VERSION option that you specify is the version of the DB2 database server that you want to access.

The name of the WRAPPER parameter must be the name that you specified in the CREATE WRAPPER statement.

Although the database name is specified as an option in the CREATE SERVER statement, it is required for DB2 data sources.

When you issue the CREATE SERVER statement, the federated server will test the connection to the DB2 data source server.

After you register the server definition, you can add or drop server options by issuing the ALTER SERVER statement.

The next task in this sequence of tasks is creating the user mapping for a DB2 data source.

Related tasks:

- "Adding DB2 family data sources to a federated server" on page 157
- "Creating the user mapping for a DB2 data source" on page 163

Related reference:

- "CREATE SERVER statement" in the *SQL Reference, Volume 2*
- "Supported data sources" on page 5
- "CREATE SERVER statement - Examples for DB2 wrapper" on page 162

CREATE SERVER statement - Examples for DB2 wrapper

This topic provides examples that show you how to use the CREATE SERVER statement to register servers for wrappers on DB2 family data sources. This topic includes a complete example, which shows how to create a server with all required parameters, and an example with additional server options.

Complete example:

The following example shows you how to create a server definition for a DB2 wrapper by using the CREATE SERVER statement:

```
CREATE SERVER DB2SERVER TYPE DB2/ZOS VERSION 6 WRAPPER DRDA
       AUTHORIZATION "spalten" PASSWORD "db2guru"
        OPTIONS (DBNAME 'CLIENTS390')
```

DB2SERVER
: A name that you assign to the DB2 database server. This name must be unique. Duplicate server names are not allowed.

TYPE *DB2/ZOS*
: Specifies the type of data source server to which you are configuring access.

VERSION *6*
: The version of the DB2 database server that you want to access.

WRAPPER *DRDA*
: The name that you specified in the CREATE WRAPPER statement.

AUTHORIZATION *"spalten"*
: The authorization ID at the data source. This ID must have BINDADD authority at the data source. This value is case sensitive.

PASSWORD *"db2guru"*
: The password that is associated with the authorization ID at the data source. This value is case sensitive.

DBNAME *'CLIENTS390'*

The alias for the DB2 database that you want to access. You defined this alias when you cataloged the database using the **CATALOG DATABASE** command. This value is case sensitive.

This database name is required for DB2 data sources.

Server option example:

When you register the server definition, you can specify additional server options in the CREATE SERVER statement. These options include general server options and DB2 data source-specific server options.

The following example shows a server definition with the CPU_RATIO option.

```
CREATE SERVER DB2SERVER TYPE DB2/ZOS VERSION 6 WRAPPER DRDA
       AUTHORIZATION "spalten" PASSWORD "db2guru"
        OPTIONS (DBNAME 'CLIENTS390', CPU_RATIO '0.001')
```

If you set the CPU_RATIO option to '0.001', this indicates the CPU at the remote data source 1000 times more available capacity than the federated server.

Related tasks:

- "Registering the server definitions for a DB2 data source" on page 161

Related reference:

- "CREATE SERVER statement" in the *SQL Reference, Volume 2*
- Appendix C, "Valid server types in SQL statements," on page 569

Creating the user mapping for a DB2 data source

Creating the user mapping for a DB2 data source is part of the larger task of adding DB2 family data sources to federated servers.

When you attempt to access a DB2 server, the federated server establishes a connection to the data source using a user ID and password that are valid for that data source. You must define an association (a user mapping) between the federated server user ID and password and the corresponding data source user ID and password. Create a user mapping for each user ID that will access the federated system to send distributed requests.

Procedure:

To map the local user ID to the DB2 server user ID and password, issue a CREATE USER MAPPING statement.

For example:

```
CREATE USER MAPPING FOR USERID SERVER DB2SERVER
       OPTIONS (REMOTE_AUTHID 'remote_id', REMOTE_PASSWORD 'remote_password')
```

The REMOTE_AUTHID is the connect authorization ID, not the bind authorization ID.

The next task in this sequence of tasks is testing the connection to the DB2 data source server.

Related tasks:

- “Testing the connection to the DB2 data source server” on page 165

Related reference:

- “CREATE USER MAPPING statement” in the *SQL Reference, Volume 2*
- “CREATE USER MAPPING statement - Examples for DB2 wrapper” on page 164

CREATE USER MAPPING statement - Examples for DB2 wrapper

This topic provides examples that show you how to use the CREATE USER MAPPING statement to map a local user ID to the DB2 server user ID and password. This topic includes a complete example with all required parameters and an example that shows you how to use the DB2 special register USER with the CREATE USER MAPPING statement.

Complete example:

The following example shows how to map a local user ID to the DB2 server user ID:

```
CREATE USER MAPPING FOR DB2USER SERVER DB2SERVER
      OPTIONS (REMOTE_AUTHID 'db2admin', REMOTE_PASSWORD 'day2night')
```

DB2USER
: Specifies the local user ID that you are mapping to a user ID that is defined at a DB2 family data source server.

SERVER *DB2SERVER*
: Specifies the name of the DB2 family data source server that you defined in the CREATE SERVER statement.

REMOTE_AUTHID *′db2admin′*
: Specifies the connect authorization user ID at the DB2 family data source server to which you are mapping *DB2USER*. Use single quotation marks to preserve the case of this value unless you set the FOLD_ID server option to ′U′ or ′L′ in the CREATE SERVER statement.

REMOTE_PASSWORD *′day2night′*
: Specifies the password that is associated with *′db2admin′*. Use single quotation marks to preserve the case of this value unless you set the FOLD_PW server option to ′U′ or ′L′ in the CREATE SERVER statement.

Special register example:

The following is an example of the CREATE USER MAPPING statement which includes the special register USER:

```
CREATE USER MAPPING FOR USER SERVER DB2SERVER
      OPTIONS (REMOTE_AUTHID 'db2admin', REMOTE_PASSWORD 'day2night')
```

You can use the DB2 special register USER to map the authorization ID of the person issuing the CREATE USER MAPPING statement to the data source authorization ID specified in the REMOTE_AUTHID user option.

Related tasks:

- “Creating the user mapping for a DB2 data source” on page 163

Related reference:

- “CREATE USER MAPPING statement” in the *SQL Reference, Volume 2*

Testing the connection to the DB2 data source server

Testing the connection to the DB2 data source server is part of the larger task of adding DB2 family data sources to federated servers.

You can test the connection to the DB2 server by using the server definition and user mappings that you defined.

Procedure:

To test the connection:

1. Open a pass-through session to issue an SQL SELECT statement on the DB2 system tables.

   For example:

   - On DB2 for z/OS and OS/390:

     ```
     SET PASSTHRU server_name
     SELECT count(*) FROM sysibm.systables
     SET PASSTHRU RESET
     ```

   - On DB2 for iSeries:

     ```
     SET PASSTHRU remote_server_name
     SELECT count(*) FROM qsys2.systables
     SET PASSTHRU RESET
     ```

   If the SQL SELECT statement returns a count, your server definition and your user mapping are set up properly.
2. If the SQL SELECT statement returns an error, you might need to:
   - Check the remote server to make sure that it is started.
   - Check the listener on the remote server to make sure that it is configured for incoming connections.
   - Check your user mapping to make sure that the settings for the REMOTE_AUTHID and REMOTE_PASSWORD options are valid for connections to the DB2 server.
   - Check the DB2 catalog entries for the node and database.
   - Check the settings of your DB2 federated variables to verify that you can access the remote DB2 server. These variables include the system environment variables, the db2dj.ini variables, and the DB2 Profile Registry (db2set) DB2COMM variable.
   - Check your server definition. If necessary, drop the server definition, and create it again.
   - Check your user mapping. If necessary, alter the user mapping, or create another user mapping.

The next task in this sequence of tasks is registering nicknames for DB2 tables and views.

Related tasks:

- “Registering nicknames for DB2 tables and views” on page 166
- “Setting the data source environment variables” on page 58

Related reference:

- “ALTER USER MAPPING statement” in the *SQL Reference, Volume 2*

Registering nicknames for DB2 tables and views

Registering nicknames for DB2 tables and views is part of the larger task of adding DB2 family data sources to a federated server.

The federated database relies on catalog statistics for nicknamed objects to optimize query processing. These statistics are gathered when you register a nickname for a data source object using the CREATE NICKNAME statement. The federated database verifies the presence of the object at the data source, and then attempts to gather existing data source statistical data. Information useful to the optimizer is read from the data source catalogs and put into the global catalog on the federated server. Because some or all of the data source catalog information might be used by the optimizer, it is advisable to update statistics (using the data source command equivalent to the **RUNSTATS** command) at the data source before you register a nickname.

Use the CREATE NICKNAME statement to register a nickname for a view or table that is located at your DB2 family data source. Use these nicknames, instead of the names of the data source objects, when you query the DB2 family data source.

Restrictions:

| You cannot create a nickname on a DB2 alias.

Procedure:

To register a nickname, issue the CREATE NICKNAME statement.

For example:

```
CREATE NICKNAME DB2NICKNAME FOR DB2SERVER.remote_schema.remote_table
```

Nicknames can be up to 128 characters in length.

Repeat this step for each DB2 table or view for which you want to register a nickname.

When you register the nickname, the federated server will use the connection to query the data source catalog. This query tests your connection to the data source by using the nickname. If the connection does not work, you will receive an error message.

Related reference:
- "RUNSTATS Command" in the *Command Reference*
- "CREATE NICKNAME statement" in the *SQL Reference, Volume 2*
- "CREATE NICKNAME statement - Examples for DB2 wrapper" on page 166

CREATE NICKNAME statement - Examples for DB2 wrapper

This topic provides an example that shows you how to use the CREATE NICKNAME statement to register a nickname for a DB2 table or view that you want to access.

The following example shows a CREATE NICKNAME statement:

```
CREATE NICKNAME DB2SALES FOR DB2SERVER.SALESDATA.EUROPE
```

DB2SALES
A unique nickname that is used to identify the DB2 table or view.

Note: The nickname is a two-part name that includes the schema and the nickname. If you omit the schema when you register the nickname, the schema of the nickname will be the authorization ID of the user creating the nickname.

DB2SERVER.SALESDATA.EUROPE
A three-part identifier for the remote object:

- *DB2SERVER* is the name that you assigned to the DB2 database server in the CREATE SERVER statement.
- *SALESDATA* is the name of the remote schema to which the table or view belongs. This value is case sensitive.
- *EUROPE* is the name of the remote table or view that you want to access.

Related tasks:

- "Altering a nickname" on page 523
- "Registering nicknames for DB2 tables and views" on page 166

Related reference:

- "CREATE NICKNAME statement" in the *SQL Reference, Volume 2*

Chapter 10. Configuring access to Documentum data sources

This chapter explains how to configure your federated server to access data that is stored in Documentum data sources. You can configure access to Documentum data sources by using the DB2 Control Center or by issuing SQL statements.

This chapter:
- Explains what Documentum is
- Lists the tasks that you need to perform
- Contains examples of the SQL statements that you need
- Lists the error messages associated with the Documentum wrapper

What is Documentum?

Documentum is document management software that provides management of document content and attributes such as check-in, check-out, workflow, and version management. The Documentum product is a three-tier, client-server system built on top of a relational database.

A Docbase is a Documentum repository that stores document content, attributes, relationships, versions, renditions, formats, workflow, and security. Documentum Query Language (DQL), an extended SQL dialect, is used to query Documentum data. A Docbase is the equivalent of an Oracle instance or a DB2® database plus document content files. The metadata is stored in the underlying relational database management system (RDBMS), and the content is stored as binary large objects (BLOBs) in the database or as files stored within the file-system of the server system. For more information on Documentum, refer to the Documentum manuals.

The wrapper for Documentum allows you to add a Documentum data source to a DB2 federated system. By adding the Documentum data source to a federated system, you can use SQL statements to access and query objects and registered tables in a Documentum Docbase. You can then integrate this data with other data sources in your federated system without having to move the data out of the native data source. The Documentum wrapper uses a client library to interface with the Documentum server. The Documentum wrapper provides access to two versions of the Documentum server: EDMS 98 (also referred to as version 3) and 4i. Figure 20 on page 170 illustrates how the Documentum wrapper works.

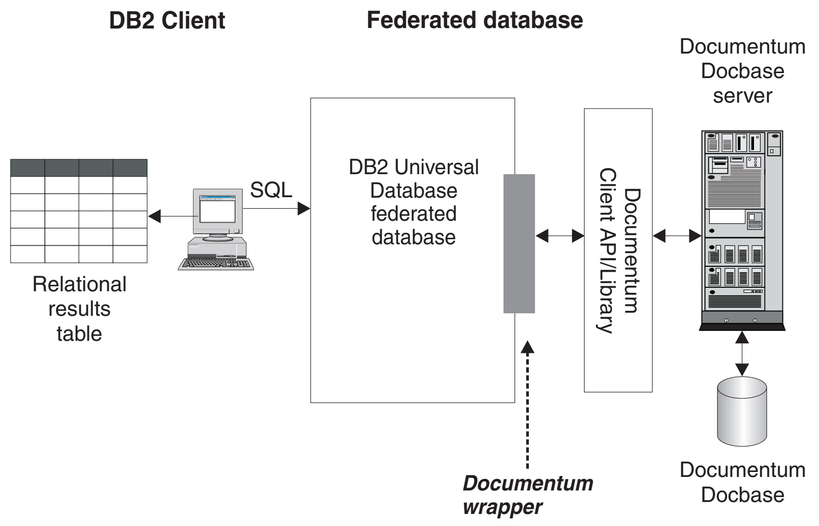

DB2 Client
Federated database
Documentum Docbase server
SQL
DB2 Universal Database federated database
Documentum Client API/Library
Relational results table
Documentum wrapper
Documentum Docbase

Figure 20. How the Documentum wrapper works

After the Documentum wrapper is registered, you can map Documentum Docbase objects and registered tables as relational tables. This is done by mapping Docbase attributes to column names in a DB2 relational table.

For example, Table 35 lists a subset of attributes for the Documentum Docbase default document type, dm_document, along with the associated data. You have determined that this attribute subset is important to you, and you would like to connect these attributes into your federated database system. You named this subset of data DrugAB_data.

Table 35. DrugAB_data

Title	Subject	Authors	Keywords
The effect of drug A on rabbits	Drug A	Curran, L.	rabbits, drug A
Toxicity results for drug A	Drug A	Abelite, P., McMurtrey, K.	toxicity, drug A
Drug B interactions	Drug B	DeNiro, R., Stone, S.	interactions, drug B
Chemical structure of drug B	Drug B	Boyslim, F.	structure, drug B

After you register the Documentum wrapper, the data can be queried using SQL statements.

The following query displays the titles and authors whose subject is Drug A. The result table is shown in Table 36.

```
SELECT title, authors
 FROM drugAB_data
 WHERE subject = 'Drug A'
```

Table 36. Query results

Title	Authors
The effect of drug A on rabbits	Curran, L.
Toxicity results for drug A	Abelite, P., McMurtrey, K.

Related tasks:
- "Adding Documentum data sources to a federated server" on page 171

Adding Documentum to a federated server

Adding Documentum data sources to a federated server

To configure the federated server to access Documentum data sources, you must provide the federated server with information about the data sources and objects that you want to access.

| You can configure the federated server to access Documentum data sources by
| using the DB2 Control Center or the DB2 command line. The DB2 Control Center
| includes a wizard to guide you through the steps required to configure the
| federated server.

Prerequisites:
- DB2 Information Integrator must be installed on a server that will act as the federated server
- A federated database must exist on the federated server

Procedure:

To add Documentum data sources to a federated server:
1. Making the Documentum client library available to the wrapper.
2. Set the Documentum environment variables
3. Register the wrapper.
4. Register the server definition.
5. Create the user mappings.
6. Register nicknames for Documentum Docbase objects and registered tables.
7. Register custom functions for the Documentum wrapper.

Related concepts:
- "DB2 Information Integrator installation process - overview" in the *IBM DB2 Information Integrator Installation Guide for Linux, UNIX, and Windows*
- "Fast track to configuring your data sources" on page 55

Related tasks:
- "Creating a federated database" on page 51
- "Making the Documentum client library available to the wrapper" on page 172
- "Setting the Documentum environment variables" on page 172
- "Registering the Documentum wrapper" on page 174
- "Registering the server for Documentum data sources" on page 175
- "Registering user mappings for Documentum data sources" on page 176
- "Registering nicknames for Documentum data sources" on page 176
- "Registering the custom functions for the Documentum wrapper" on page 182
- "Checking the setup of the federated server" on page 37

Related reference:

- "Supported operating systems for DB2 Information Integrator (32-bit)" in the *IBM DB2 Information Integrator Installation Guide for Linux, UNIX, and Windows*
- "Supported operating systems for DB2 Information Integrator (64-bit)" in the *IBM DB2 Information Integrator Installation Guide for Linux, UNIX, and Windows*

Making the Documentum client library available to the wrapper

Making the Documentum client library available to the wrapper is part of the larger task of adding Documentum to a federated system.

The client library must be made available to the wrapper for the wrapper to function correctly.

Prerequisites:

The Documentum wrapper uses Version 3.1.7a of the client library. If you are using Documentum 4i , you will need to acquire the older version of the client library from Documentum (if it is not already installed).

Procedure:

To make the Documentum client library available to the wrapper, create a symbolic link or copy the client library into the appropriate directory on the federated server. The following table lists the directory that you should copy the library into.

Table 37. Client library and copy to directory by operating system

Federate server operating system	Client library	Copy to directory
AIX	libdmcl.a	`sqllib/lib`
Solaris	libdmcl.so	`sqllib/lib`
Windows	dmcl32.dll	`x:\sqllib\bin`

The next task in this sequence of tasks is setting the Documentum wrapper environment variables.

Related tasks:
- "Setting the Documentum environment variables" on page 172

Setting the Documentum environment variables

Setting the Documentum environment variables is part of the larger task of adding Documentum to a federated server.

Access to Documentum Docbases is controlled through the Documentum client file `dmcl.ini`. The federated database instance must have its environment variables set to the Documentum client file `dmcl.ini` to gain access to a Documentum Docbase.

The valid environment variables for Documentum are:
- DOCUMENTUM
- DMCL_CONFIG

Prerequisites:

A federated server that is properly set up to access your data sources. This includes the installation and configuration of any required software, such as the data source client software.

Restrictions:

See the topic: Restrictions for the db2dj.ini file

Procedure:

To set the environment variables:

1. Edit the `db2dj.ini` file, and set either the `DOCUMENTUM` or `DMCL_CONFIG` environment variable.
   - On federated servers that run AIX and Solaris, the `db2dj.ini` file is located in `$HOME/sqllib/cfg`
   - On federated servers that run Windows, the `db2dj.ini` file is in `x:\sqllib\cfg` where **x:** represents the drive on which the `sqllib` directory is located

   The following examples shows the syntax for these variables on UNIX operating systems:

   DOCUMENTUM=*path*

   or

   DMCL_CONFIG=*path*/dmcl.ini

   | where *path* is the fully qualified directory that contains the `dmcl.ini` file that
   | you want to use. Refer to the documentation that comes with Documentum for
   | more information about path for the `dmcl.ini` file.

   | If both of these variables are set in the `db2dj.ini` file, the value for the
   | DMCL_CONFIG variable is used. If you do not set either of these variables in
   | the `db2dj.ini` file, an error is returned.
2. Ensure that the name of a docbroker, to which all accessible Docbases for the DB2 instance report, is specified in the `dmcl.ini` file as shown in Figure 21.

```
################# DOCUMENTUM CLIENT CONFIGURATION FILE #####################
#
# Copyright Documentum 1994.
# Version 3.1 of the Documentum Server.
#
# A generated client init file for the Documentum Server.
#
# The only REQUIRED information in this file is the
# [DOCBROKER_PRIMARY] section and an entry for host.
# The host value should be the name of host on which
# your network wide DocBroker is running

[DOCBROKER_PRIMARY]
host = server16.comp2.big.com
```

Figure 21. Sample dmcl.ini file with docbroker name specified

3. To ensure that the environment variables are set on the federated server, recycle the DB2 instance. Issue the following commands to recycle the DB2 instance:

   ```
   db2stop
   db2start
   ```

The next task in this sequence of tasks is registering the Documentum wrapper.

Related tasks:
- "Registering the Documentum wrapper" on page 174

Related reference:
- "Restrictions for the db2dj.ini file" on page 59

Registering the Documentum wrapper

Registering the Documentum wrapper is part of the larger task of adding Documentum data sources to a federated server.

You must register a wrapper to access Documentum data sources. Wrappers are used by federated servers to communicate with and retrieve data from data sources. Wrappers are implemented as a set of library files.

Procedure:

To register a wrapper, issue the CREATE WRAPPER statement with the name of the wrapper and the name of the wrapper library file.

For example, to register a wrapper with the name `dctm_wrapper` on the federated server that uses the AIX operating system, issue the following statement:

```
CREATE WRAPPER dctm_wrapper LIBRARY 'libdb2lsdctm.a';
```

The name of the wrapper library file that you specify depends on the operating system of the federated server. See the list of Documentum wrapper library files for the correct name to specify in the CREATE WRAPPER statement.

The next task in this sequence of tasks is registering the server definition for Documentum data sources.

Related reference:
- "Documentum wrapper library files" on page 174
- "CREATE WRAPPER statement" in the *SQL Reference, Volume 2*

Documentum wrapper library files

The following table lists the directory paths and library file names for the Documentum wrapper.

When you install DB2 Information Integrator, 3 library files are added to the directory path listed in the table. For example, if the federated server is running on AIX, the wrapper library files added to the directory path are `libdb2lsdctm.a`, `libdb2lsdctmF.a`, and `libdb2lsdctmU.a`.

When you register a wrapper, specify only the library file name that is listed in the table.

Table 38. Documentum wrapper library locations and file names

Operating system	Directory path	Wrapper library file
AIX	/usr/opt/db2_08_01/lib/	libdb2lsdctm.a

Table 38. Documentum wrapper library locations and file names (continued)

Operating system	Directory path	Wrapper library file
Solaris	/opt/IBM/db2/V8.1/lib	libdb2lsdctm.so
Windows	%DB2PATH%\bin	db2lsdctm.dll

%DB2PATH% is the environment variable that is used to specify the directory path where DB2 Information Integrator is installed on Windows. The default Windows directory path is `C:\Program Files\IBM\SQLLIB`.

Related tasks:
- "Registering the Documentum wrapper" on page 174

Registering the server for Documentum data sources

Registering the server definition for Documentum data sources is part of the larger task of adding Documentum to a federated system.

Restrictions:

All servers running on the same instance of DB2 must share the same configuration parameters in the Documentum `dmcl.ini` file.

Procedure:

You can register a server definition from the DB2 Control Center or the DB2 command line:
- To do this task from the DB2 Control Center, use the Federated Objects wizard or right-click the **Server Definitions** folder and click **Create**. The Discover tool retrieves the node names for the Documentum servers. You must specify the information for the RDBMS_TYPE and OS_TYPE server options to register the server definition.
- To do this task from the DB2 command line, use the CREATE SERVER statement.

  For example, to register the server definition `Dctm_Server1` for a Documentum server that contains a Docbase that runs on AIX and uses Oracle to store data, use this statement:

```
CREATE SERVER Dctm_Server1
 TYPE DCTM
 VERSION 3
 WRAPPER Dctm_Wrapper
 OPTIONS( NODE 'Dctm_Docbase',
    OS_TYPE 'AIX',
    RDBMS_TYPE 'ORACLE');
```

The next task in this sequence of tasks is creating the user mappings.

Related tasks:
- "Registering user mappings for Documentum data sources" on page 176

Related reference:
- "CREATE SERVER statement" in the *SQL Reference, Volume 2*
- "CREATE SERVER statement arguments and options - Documentum wrapper" on page 540

Registering user mappings for Documentum data sources

Creating user mappings for Documentum data sources is part of the larger task of adding Documentum to a federated server.

When you attempt to access a Documentum data source, the federated server establishes a connection to the data source using a user ID and password that are valid for that data source. You must define an association (a user mapping) between the federated server user ID and password and the corresponding data source user ID and password. Create a user mapping for each user ID that will access the federated system to send distributed requests.

Procedure:

To create a user mapping, use the CREATE USER MAPPING statement.

For example, the following CREATE USER MAPPING statement maps user `Chuck` to user `Charles` on the `Dctm_Server1` server.

```
CREATE USER MAPPING FOR Chuck SERVER Dctm_Server1
    OPTIONS(REMOTE_AUTHID 'Charles', REMOTE_PASSWORD 'Charles_pw');
```

You can use the DB2 special register USER to map the authorization ID of the person issuing the CREATE USER MAPPING statement to the data source authorization ID specified in the REMOTE_AUTHID user option. For example:

```
CREATE USER MAPPING FOR USER SERVER Dctm_Server1
    OPTIONS(REMOTE_AUTHID 'Lisa', REMOTE_PASSWORD 'Lisa_pw');
```

The next task in this sequence of tasks is registering nicknames for Documentum data sources.

Related tasks:
- “Registering nicknames for Documentum data sources” on page 176

Related reference:
- “CREATE USER MAPPING statement” in the *SQL Reference, Volume 2*
- “CREATE USER MAPPING statement options - Documentum wrapper” on page 541

Registering nicknames for Documentum data sources

Registering nicknames for Documentum data sources

Registering nicknames for Documentum data sources is part of the larger task of adding Documentum to a federated server.

After you create the user mappings, you must register nicknames. For each Documentum server that you define, register nicknames for each Documentum Docbase object type or registered table that you want to access. You use these nicknames, instead of the names of the Docbase, when you query the Documentum data sources.

Procedure:

To register nicknames, use the CREATE NICKNAME statement.

As part of the CREATE NICKNAME statement, you can define pseudo columns. If you do not define the pseudo columns, the wrapper will create them

The next task in this sequence of tasks is registering custom functions for Documentum data sources.

Related tasks:
- "Registering the custom functions for the Documentum wrapper" on page 182
- "Setting up TurboBlast to work with the BLAST wrapper" on page 113

Related reference:
- "Pseudo columns" on page 177
- "CREATE NICKNAME statement syntax - Documentum wrapper" on page 541
- "CREATE NICKNAME statement - Example for Documentum wrapper" on page 180

Pseudo columns

The CREATE NICKNAME statement that you use with Documentum defines 6 pseudo columns. These columns are used to access object content and other information.

The pseudo-columns and their definitions are listed in Table 39.

Table 39. Pseudo column names and definitions.

Pseudo column name	Definition
GET_FILE	VARCHAR (nnn) [1]
GET_FILE_DEL	VARCHAR (nnn) [1]
GET_RENDITION	VARCHAR (nnn) [1]
GET_RENDITION_DEL	VARCHAR (nnn) [1]
HITS	INTEGER
SCORE	DOUBLE

Note:

1. The length of VARCHAR is determined by the maximum path length of the federated server operating system. On UNIX federated servers, this length is 1024. On Windows federated servers, this length is 260.

Table 40 lists pseudo columns for SELECT clauses.

Table 40. Pseudo columns for SELECT clauses

Pseudo column name	Description

GET_FILE	Retrieves the content file for the current row in addition to the column values.
	
	The extension for the content file is its Documentum format name. If a file of the same name exists, it will be overwritten.
	
	GET_FILE attempts to get the object's base format. Its value in the row is the fully qualified file name of the file or the string ″no_content.″
	
	For example:
	`SELECT object_name, get_file`
	`FROM ...`
	
	The content file is placed in the server directory that is specified by the Server's CONTENT_DIR option. It is also placed in a subdirectory named with your DB2 local name. The subdirectory will be created if it doesn't exist.
	
	It's extension will be its DOS extension defined in the Docbase for the document's format type. For example, ″.doc″, for MS Word documents.

GET_FILE_DEL	This function is the same as GET_FILE except GET_FILE_DEL first deletes the file retrieved for the previous row, if any, in that query. Its value in the row is the fully qualified file name of the file or the string ″no_content.″

GET_RENDITION	Retrieves the content file of that rendition, a copy of the original document in a different format, for the current row in addition to the column values.
	
	The extension for the content file is its Documentum format name. If a file of the same name exists, it will be overwritten.
	
	To specify the rendition format, a predicate of the form DCTM.RENDITION_FORMAT(<format) = 1 must be specified in the WHERE clause.
	
	For example:
	`SELECT object_name, get_rendition`
	`FROM ...`
	`WHERE DCTM.RENDITION_FORMAT('pdf')=1`
	
	GET_RENDITION attempts to get the named rendition of the object. Its value in the row is the fully qualified file name of the file or the string ″no_content.″
	
	The content file is placed in the server directory that is specified by the Server's CONTENT_DIR option. It is also placed in a subdirectory named with your DB2 local name. The subdirectory will be created if it doesn't exist.
	
	It's extension will be its DOS extension defined in the Docbase for the document's format type. For example, ″.doc″, for MS Word documents.

Table 40. Pseudo columns for SELECT clauses (continued)

Pseudo column name	Description
GET_RENDITION_DEL	This function is the same as GET_RENDITION except GET_RENDITION_DEL first deletes the file retrieved for the previous row, if any, in that query. Its value in the row is the fully qualified file name of the file or the string ″no_content.″

Table 41 lists pseudo columns for SELECT clauses in queries that contain search clauses.

Table 41. Pseudo columns for SELECT clauses in queries that contain search clauses

Pseudo column name	Description

HITS	Contains an integer number that represents the number of places in the document in which the search criteria was matched.
	
	For example:
	`SELECT r_object_id, object_name, hits`
	`  FROM std_doc`
	`  WHERE DCTM.SEARCH_WORDS ('''workflow'' OR ''flowchart''')=1`
	
	For each document returned, the number of occurrences of the words ″workflow″ and ″flowchart″ within the document's content are summed and returned as the HITS value.
	
	The HITS pseudo column is appropriate when the documents have only one content file. This is the typical case. This pseudo column can be used in a WHERE clause qualification for a SELECT statement. However, it must also be specified in the SELECT clause.

SCORE	Contains a document's relevance ranking.
	
	Use this pseudo column in conjunction with the Documentum's ACCRUE concept operator. Both return a number that indicates how many of the specified words were found in each returned document.
	
	For example:
	`SELECT object_name, score`
	` FROM std_doc`
	` WHERE`
	`DCTM.SEARCH_TOPIC('<ACCRUE>("document","management","workflow")')=1`
	`  AND SCORE >=75`
	
	The statement returns all documents that have either two or three of the specified words in their content. If a document has only one of the words, it is assigned a score of 50 and therefore fails the WHERE clause criteria and is not returned. If two of the three words are found, a document is assigned a score of 75. If all three words are found, the document's score is 88.
	
	The SCORE pseudo column is used for documents that have one content file. This is the typical case.
	
	SCORE can be in a SELECT clause only if the WHERE contains a SEARCH_WORDS() or SEARCH_TOPIC() function. In a WHERE clause, it is used in conjunction with the ACCRUE concept operator.
	
	For information on the ACCRUE concept operator, see the Documentum documentation.

Related tasks:

- "Registering nicknames for Documentum data sources" on page 176
- "Registering the custom functions for the Documentum wrapper" on page 182

CREATE NICKNAME statement - Example for Documentum wrapper

The following CREATE NICKNAME statement defines the nickname std_doc. Std_doc is associated with a Documentum Docbase with an object type of dm_document. Table 42 maps the Documentum attributes and data types to DB2 relational column names and data types that are then used to construct the CREATE NICKNAME statement.

Table 42. Mapping of Documentum attributes to DB2 columns for the std_doc nickname

Documentum attribute name	Documentum data type	DB2 column name	DB2 data type	Repeats?	Nullable?
object_name	string(255)	object_name	varchar	No	No
r_object_id	ID	object_id	char(16)	No	No
r_object_type	string(32)	object_type	varchar	No	No
title	string(255)	title	varchar	No	No
subject	string(128)	subject	varchar	No	No
authors	string(32)	author	varchar	Yes	Yes
keywords	string(32)	keyword	varchar	Yes	Yes
r_creation_date	time	creation_date	timestamp	No	Yes
r_modify_date	time	modified_date	timestamp	No	Yes
a_status	string(16)	status	varchar	No	No
a_content_type	string(32)	content_type	varchar	No	No
r_content_size	double	content_size	integer	No	No
owner_name	string(32)	owner_name	varchar	No	Yes

Table 43 describes each Documentum attribute used in the nickname.

Table 43. Description of Documentum attributes for the std_doc nickname

Documentum attribute name	Description
object_name	The user-defined name of the object.
r_object_id	The unique object identifier for this object, set at creation time.
r_object_type	The object's type, set when the object is created.
title	The user-defined title of the object.
subject	The user-defined subject of the object.
authors	The user-defined list of the authors for the object.
keywords	The list of user-defined keywords for the object.
r_creation_date	The date and time that the object was created.
r_modify_date	The date and time that the object was last modified.
a_status	Set by server when a router task is forwarded. The value is taken from the values assigned to attached_task_status in the router object.
a_content_type	The file format of the object's content.

Table 43. Description of Documentum attributes for the std_doc nickname (continued)

Documentum attribute name	Description
r_content_size	The number of bytes in the content. For multi-page documents, this attribute records the size of the first content associated with the document.
owner_name	The name of the object's owner (the user who created the object).

Table 42 on page 180 translates into the following CREATE NICKNAME statement.

```
CREATE NICKNAME std_doc (
  object_name varchar(255) not null,
  object_id char(16) not null OPTIONS(REMOTE_NAME 'r_object_id'),
  object_type varchar(32) not null OPTIONS(REMOTE_NAME 'r_object_type'),
  title varchar(255) not null,
  subject varchar(128) not null,
  author varchar(32) OPTIONS(REMOTE_NAME 'authors', IS_REPEATING 'Y'),
  keyword varchar(32) OPTIONS(REMOTE_NAME 'keywords', IS_REPEATING 'Y'),
  creation_date timestamp OPTIONS(REMOTE_NAME 'r_creation_date'),
  modifed_date timestamp OPTIONS(REMOTE_NAME 'r_modify_date'),
  status varchar(16) not null OPTIONS(REMOTE_NAME 'a_status'),
  content_type varchar(32) not null OPTIONS(REMOTE_NAME 'a_content_type'),
  content_size integer not null OPTIONS(REMOTE_NAME 'r_content_size'),
  owner_name varchar(32))
FOR SERVER Dctm_Server2 OPTIONS (REMOTE_OBJECT 'dm_document', IS_REG_TABLE 'N')
```

After you submit the CREATE NICKNAME statement, you can use the nickname *std_doc* to query your federated system. You can also join the *std_doc* nickname with other nicknames and tables in the federated system.

In the catalog, the number of columns for this nickname is 6 more than what is being specified in the CREATE NICKNAME statement due to the pseudo columns.

You can use the CreateNicknameFile utility to automatically map Documentum types to DB2 types and to create an initial CREATE NICKNAME statement.

Related tasks:

- "Registering nicknames for Documentum data sources" on page 176

Related reference:

- "CREATE NICKNAME statement syntax - Documentum wrapper" on page 541

Dual defining repeating attributes (Documentum wrapper)

To maximize the query capabilities of the wrapper, each attribute must be defined as its true equivalent DB2 data type. That is, Documentum integers must be defined as DB2 integers and so forth. However, these definitions prevent the return of multiple values for non-VARCHAR repeating attributes. For such columns, only the last value is returned.

This restriction exists because, whenever possible, the wrapper returns only one row of results per Docbase object. This restriction is an issue only when repeating attributes are selected. However, you can define a second column for the same remote repeating attribute but with a data type of VARCHAR.

This column name would be used in the SELECT list to return all values as a delimiter-separated list of all its values. (Each column's DELIMITER option specifies the delimiter to be used.)

You should standardize the local names of the multi-value columns. You can standardize the local names of each multi-value column by adding a prefix of "m_" to the local name of the column that is defined as its true data type.

For example, suppose you have a nickname column of a Documentum repeating attribute called approval_dates defined with the data type TIMESTAMP. You can create a second nickname column called m_approval_dates and define it as a VARCHAR data type. You can then use m_approval_dates in a SELECT list to return all approval dates in a delimiter-separated list.

You do not need to use dual definitions for repeating attributes whose true data type is VARCHAR.

Related tasks:

- "Registering nicknames for Documentum data sources" on page 176
- "Altering a nickname" on page 523

Queries and custom functions for Documentum data sources

Registering the custom functions for the Documentum wrapper

Registering custom functions for Documentum data sources is part of the larger task of adding Documentum to a federated system. You must use the CREATE FUNCTION statement to register several custom functions. You can use these functions to access some of the unique capabilities of Documentum, such as full-text searching and retrieving document content within queries.

Custom functions for predicates are listed in Table 44 on page 183.

References to the TOPIC function are to Documentum function provided as part of its third-party full-text indexing system from Verity, Inc

Restrictions:

Because DB2 does not support the Boolean type, most of the custom functions (except for USER) used in the WHERE clause must do a check for "=1" because these functions are defined to return an integer.

For example,

```
"... WHERE DCTM.ANY_EQ(authors,'Dave Winters')=1"
```

Procedure:

To register custom functions, use the CREATE FUNCTION statement.

All custom functions must be registered with the schema name DCTM. The fully-qualified name of each function is DCTM.function_name.

The following example registers the ANY_EQ custom function.

```
CREATE FUNCTION DCTM.ANY_EQ (CHAR(), CHAR()) RETURNS INTEGER
 AS TEMPLATE DETERMINISTIC NO EXTERNAL ACTION
```

You must register each custom function one time for each federated database that has the Documentum wrapper installed.

To assist you in registering custom functions, the sample file, `create_function_mappings.ddl`, is provided in the `sqllib/samples/lifesci/dctm` directory. This file contains definitions for each custom function. You can run this DDL file to register the custom functions for each federated database that has the Documentum wrapper installed.

Custom function string argument rules

All arguments passed as strings must adhere to the following rules:

- Each string is enclosed in single quotes.
- Single quotes within strings are expressed by two single quotes.

Using custom functions in queries

The following examples illustrate the use of the custom functions in queries.

To display the object name and author from the std_doc nickname for documents that have one or more authors named 'Dave Winters':

```
SELECT object_name,authors FROM std_doc
WHERE DCTM.ANY_EQ(authors,'Dave Winters')=1
```

To display the object name and author from the std_doc nickname for documents that have one or more authors named 'Dave Winters' or 'Jon Doe':

```
SELECT object_name,authors FROM std_doc
WHERE DCTM.ANY_IN(authors,'Dave Winters','Jon Doe')=1
```

To display the object name and r_object_id, and to retrieve the content file, from the std_doc nickname for documents containing strings like 'Dave Win%' in the authors column:

```
SELECT object_name, r_object_id, get_file FROM std_doc
WHERE DCTM.ANY_LIKE(authors,'Dave Win%')=1
```

Custom function table

Table 44 lists the custom functions for predicates.

Table 44. Custom functions for predicates

Function name	Description
ANY_EQ(arg1, arg2)	Tests a repeating attribute for any value equal to the specified value. Takes two required arguments: **arg1** Specifies the name of a column that represents a repeating attribute. **arg2** Specifies the value to be compared. For example: `... WHERE DCTM.ANY_EQ(authors,'Dave Winters')=1`

Table 44. Custom functions for predicates (continued)

Function name	Description
ANY_NE(arg1, arg2)	Tests a repeating attribute for any value not equal to the specified value. Takes two required arguments: **arg1** Specifies the name of a column that represents a repeating attribute. **arg2** Specifies the value to be compared. For example: `... WHERE DCTM.ANY_NE(authors,'Dave Winters')=1`
ANY_LT(arg1, arg2)	Tests a repeating attribute for any value less than the specified value. Takes two required arguments: **arg1** Specifies the name of a column that represents a repeating attribute. **arg2** Specifies the value to be compared. For example: `... WHERE DCTM.ANY_LT(num_approvers,4)=1`
ANY_GT(arg1, arg2)	Tests a repeating attribute for any value greater than the specified value. Takes two required arguments: **arg1** Specifies the name of a column that represents a repeating attribute. **arg2** Specifies the value to be compared. For example: `... WHERE DCTM.ANY_GT(num_approvers,3)=1`
ANY_LE(arg1, arg2)	Tests a repeating attribute for any value less than or equal to the specified value. Takes two required arguments: **arg1** Specifies the name of a column that represents a repeating attribute. **arg2** Specifies the value to be compared. For example: `... WHERE DCTM.ANY_LE(num_approvers,2)=1`
ANY_GE(arg1, arg2)	Tests a repeating attribute for any value greater than or equal to the specified value. Takes two required arguments: **arg1** Specifies the name of a column that represents a repeating attribute. **arg2** Specifies the value to be compared. For example: `... WHERE DCTM.ANY_GE(num_approvers,1)=1`

Table 44. Custom functions for predicates (continued)

Function name	Description
ANY_IN(arg1, arg2 – arg11)	Tests a repeating attribute for any of ten values in a specified list of values. Takes 3–11 arguments of the same data type: **arg1** Specifies the name of a column that represents a repeating attribute. **arg2–arg11** Specifies a comma-separated list of values to be compared. For example: `... WHERE DCTM.ANY_IN(authors,'Crick','Watson')=1` The maximum number of values in an ANY_IN custom function for repeating attributes is 10 for a single statement. Multiple statements can be OR'd.

ANY_LIKE(arg1, arg2)	Tests a repeating attribute for any value like the specified value. Takes two required arguments:
	arg1 Specifies the name of a column that represents a repeating attribute.
	arg2 Specifies the pattern being compared with sub-strings in single quotes.
	For example:
	`... WHERE DCTM.ANY_LIKE(authors,'Dave Win%')=1`
	`OR DCTM.ANY_LIKE(keywords,'%\_%')=1`
	The escape clause is not supported in ANY_LIKE() predicates.

ANY_NOT_LIKE(arg1, arg2)	Tests a repeating attribute for any value not like the specified value. Takes two required arguments:
	arg1 Specifies the name of a column that represents a repeating attribute.
	arg2 Specifies the pattern being compared with sub-strings in single quotes.
	For example:
	`... WHERE DCTM.ANY_NOT_LIKE(authors,'Dave Win%')=1`
	`OR DCTM.ANY_NOT_LIKE(keywords,'%\_%')=1`
	The escape clause is not supported in ANY_NOT_LIKE() predicates.

ANY_NULL(arg)	Tests a repeating attribute for IS NULL. Takes one required argument that is the name of the repeating attribute or single-valued DATE or TIMESTAMP attribute. For example: `... WHERE DCTM.ANY_NULL(authors)=1`
ANY_NOT_NULL(arg)	Tests a repeating attribute for IS NOT NULL. Takes one required argument that is the name of the repeating attribute. For example: `... WHERE DCTM.ANY_NOT_NULL(authors)=1`

Table 44. Custom functions for predicates (continued)

Function name	Description

ANY_SAME_INDEX(arg1 – arg10)	Tests repeating attributes for values at the same index of each attribute. Takes two to ten of the other ANY_xx() functions.
	
	The following example checks whether a document has at least one author named Ken who is not affiliated with UCD.
	`... WHERE DCTM.ANY_SAME_INDEX(`
	`ANY_EQ(author_name,'Ken'),`
	`DCTM.ANY_NE(author_affiliation,'UCD'))=1`
	
	The maximum number of tests for values at the same index of repeating attributes is 10. The tests must be AND tests that are evaluated left to right.

CABINET(arg) and CABINET_TREE(arg)	Takes one required argument that is the fully-qualified name of a Docbase cabinet.
	
	For example:
	`... WHERE DCTM.CABINET('/Tools')=1`
	`... WHERE DCTM.CABINET_TREE('/MyDocs')=1`
	
	Use multiple instances of CABINET and CABINET_TREE to specify multiple cabinets.
	
	For example:
	`... WHERE DCTM.CABINET('/Tools')=1`
	`OR DCTM.CABINET_TREE('/Parts')=1`

FOLDER(arg) and FOLDER_TREE(arg)	Takes one required argument that is the fully-qualified name of a Docbase folder or cabinet.
	
	For example:
	`... DCTM.FOLDER('/Tools/Drills')=1`
	`... DCTM.FOLDER_TREE('/MyDocs/WhitePapers')=1`
	
	Use multiple instances of FOLDER and FOLDER_TREE to specify multiple folders.
	
	For example:
	`... DCTM.FOLDER('/Tools/Drills')=1`
	`OR DCTM.FOLDER_TREE('/Animals/Horses')=1`

RENDITION_FORMAT (format)	Works with the GET_RENDITION and GET_RENDITION_DEL pseudo columns to establish the format of the rendition to be retrieved. Takes a single character string argument specifying the format.
	
	The following example retrieves a document in PDF format:
	`SELECT get_rendition`
	`FROM ....`
	`WHERE DCTM.RENDITION_FORMAT('pdf')=1`

USER(1)	Compares a value to the Documentum author ID of the current user. Due to a limitation of DB2, the custom function USER is defined with an integer argument that is not used. For example: `... WHERE approver = DCTM.USER(1)` To make the Documentum author ID correspond to the DB2 author ID, use the CREATE USER MAPPING statement.

Table 44. Custom functions for predicates (continued)

Function name	Description

SEARCH_WORDS(arg)	Takes one required string argument that is a list of individual words enclosed in single quotes, separated by AND, OR, or NOT, and using parentheses to control precedence. Words cannot contain white space and must be enclosed in single quotes.
	
	For example:
	`... DCTM.SEARCH_WORDS('''yeast''`
	`AND (''bread'' OR ''cake'')`
	`AND NOT ''wedding''' )=1`

SEARCH_TOPIC(arg)	Takes one required string argument which is a Verity TOPIC query statement that is to be passed to Documentum and Verity verbatim. For example: `... WHERE DCTM.SEARCH_TOPIC('"quick"')=1`

There are no further tasks in this sequence of tasks.
Related reference:
- “CREATE FUNCTION (Sourced or Template) statement” in the *SQL Reference, Volume 2*

Documentum data source – Example queries

After you register the wrapper, you can run SQL queries on the Documentum data source. This section provides several example queries.

To run queries, you use the nickname and the defined nickname columns in your SQL statements in the same manner as you would use a regular table name and table columns.

Connection limitations:

For each connection to a DB2 database made by a DB2 application, the Documentum wrapper can support a maximum of 10 simultaneous Documentum sessions, and each such session can simultaneously manage up to 10 Documentum queries.

A single DB2 application can have several queries in progress simultaneously; the lifetime of a query begins when it is submitted to DB2 and ends when the corresponding cursor over the result set is closed.

At any given time, across the entire set of queries in progress at that time, no more than 10 nicknames from one Documentum server can be referenced. Nicknames mentioned in more than one query, or referenced multiple times in a single query, must be counted once for each time they appear.

The LIKE predicate:

The Documentum server and DB2 process the LIKE predicate differently. When a LIKE predicate is pushed down to the Documentum server, the Documentum semantics apply. In the following example when column `c1` contains a zero-length string, the predicate will be true for Documentum and false for DB2.

```
c1 LIKE '%'
```

Example queries:

The following query displays all of the Docbase documents for documents named 'Test Document':

```
SELECT object_name
FROM std_doc
WHERE object_name='Test Document';
```

The following query uses the custom function ANY_EQ to display all the documents where one of the authors is 'Joe Doe'.

```
SELECT object_name
FROM std_doc
WHERE DCTM.ANY_EQ(author,'Joe Doe')=1
```

The following query uses the FOLDER_TREE function and the SEARCH_WORDS function to find all documents in the Approved cabinet that contain the text "protein".

```
SELECT object_name
FROM std_doc
WHERE DCTM.FOLDER_TREE('/Approved')=1
      AND DCTM.SEARCH_WORDS('protein')=1
```

The following query uses the GET_FILE pseudo column and the FOLDER_TREE and ANY_IN custom functions to retrieve the name of the files, on the DB2 server, into which the content has been placed for all documents in the Approved cabinet that have any of the authors listed.

```
SELECT object_name, object_id, get_file
FROM std_doc
WHERE DCTM.FOLDER_TREE('/Approved')=1
      AND DCTM.ANY_IN(author, 'Mary Black', 'Joe Carson', 'Peter Miller')=1
```

Related tasks:

- "Registering the custom functions for the Documentum wrapper" on page 182

Related reference:

- "Dual defining repeating attributes (Documentum wrapper)" on page 181
- "Access control for the Documentum wrapper" on page 188

Access control for the Documentum wrapper

Queries are subject to your permissions in the Docbase. Only those documents to which you have at least read access are included in query results.

Related reference:

- "File access control model for the table-structured file wrapper" on page 361
- "File access control model for the Excel wrapper" on page 224

Messages for the Documentum wrapper

This section lists and describes messages you might encounter while working with the wrapper for Documentum.

Table 45. Messages issued by the wrapper for Documentum

Error Code	Message	Explanation
SQL0901N	The SQL statement failed because of a non-severe system error. Subsequent SQL statements can be processed. (Reason "sqlno_crule_save_plans [100]:rc (-2144272209) Empty plan list detect".)	The SQL query submitted to DB2 could not be processed by the wrapper. Correct the syntax and resubmit.
SQL0901N	The SQL statement failed because of a non-severe system error. Subsequent SQL statements can be processed. (Reason "dmAPI exec failed: [DM_QUERY_E_BAD_QUAL] error: "The attribute qualifier, A0, for attribute <column_name>, is not a valid qualifier."".)	An incorrect Documentum type or registered table was entered for the REMOTE_OBJECT nickname option. Change the nickname to use the correct Documentum object type or registered table.
SQL0901N	The SQL statement failed because of a non-severe system error. Subsequent SQL statements can be processed. (Reason "Invalid null column specified".)	Internal programming error. Contact IBM Software Support.
SQL0901N	The SQL statement failed because of a non-severe system error. Subsequent SQL statements can be processed. (Reason "Nickname specification is empty".)	Internal programming error. Contact IBM Software Support.
SQL0901N	The SQL statement failed because of a non-severe system error. Subsequent SQL statements can be processed. (Reason "The Output object is empty or incomplete".)	Internal programming error. Contact IBM Software Support.
SQL0901N	The SQL statement failed because of a non-severe system error. Subsequent SQL statements can be processed. (Reason "Unexpected number of columns requested".)	Internal programming error. Contact IBM Software Support.
SQL0901N	The SQL statement failed because of a non-severe system error. Subsequent SQL statements can be processed. (Reason "No column information found".)	Internal programming error. Contact IBM Software Support.
SQL0901N	The SQL statement failed because of a non-severe system error. Subsequent SQL statements can be processed. (Reason "Unsupported column type requested".)	Internal programming error. Contact IBM Software Support.

Table 45. Messages issued by the wrapper for Documentum (continued)

Error Code	Message	Explanation
SQL0901N	The SQL statement failed because of a non-severe system error. Subsequent SQL statements can be processed. (Reason "Incorrect Column definition".)	Internal programming error. Contact IBM Software Support.
SQL0901N	The SQL statement failed because of a non-severe system error. Subsequent SQL statements can be processed. (Reason "Inconsistent type; DB2 request != nickname type".)	Internal programming error. Contact IBM Software Support.
SQL0901N	The SQL statement failed because of a non-severe system error. Subsequent SQL statements can be processed. (Reason "Output parameter is not NULL".)	Internal programming error. Contact IBM Software Support.
SQL0901N	The SQL statement failed because of a non-severe system error. Subsequent SQL statements can be processed. (Reason "Query output variable is not NULL".)	Internal programming error. Contact IBM Software Support.
SQL0901N	The SQL statement failed because of a non-severe system error. Subsequent SQL statements can be processed. (Reason "Invalid timestamp length".)	Internal programming error. Contact IBM Software Support.
SQL0901N	The SQL statement failed because of a non-severe system error. Subsequent SQL statements can be processed. (Reason "Inconsistent number of columns".)	Internal programming error. Contact IBM Software Support.
SQL0901N	The SQL statement failed because of a non-severe system error. Subsequent SQL statements can be processed. (Reason "Could not access data when converting values".)	Internal programming error. Contact IBM Software Support.
SQL0901N	The SQL statement failed because of a non-severe system error. Subsequent SQL statements can be processed. (Reason "Failed to initialize the DMCL client".)	The Documentum client cannot initialize. Contact your system administrator.
SQL0901N	The SQL statement failed because of a non-severe system error. Subsequent SQL statements can be processed. (Reason "Get_User returned NULL".)	Internal programming error. Contact IBM Software Support.

Table 45. Messages issued by the wrapper for Documentum (continued)

Error Code	Message	Explanation
SQL0901N	The SQL statement failed because of a non-severe system error. Subsequent SQL statements can be processed. (Reason "Get_Local_User returned NULL".)	Internal programming error. Contact IBM Software Support.
SQL0901N	The SQL statement failed because of a non-severe system error. Subsequent SQL statements can be processed. (Reason "Begin Transaction failed".)	Documentum reported that begintrans failed. Contact your system administrator.
SQL0901N	The SQL statement failed because of a non-severe system error. Subsequent SQL statements can be processed. (Reason "Input parameter was not NULL".)	Internal programming error. Contact IBM Software Support.
SQL901N	The SQL statement failed because of a non-severe system error. Subsequent SQL statements can be processed. (Reason "Dctm functions must be like DCTM.function(...) =1".)	You did not use =1 as the RHS of the predicate for a Dctm function. Correct the syntax and run the query again.
SQL0901N	The SQL statement failed because of a non-severe system error. Subsequent SQL statements can be processed. (Reason "Invalid column number requested".)	Internal programming error. Contact IBM Software Support.
SQL1881N	"DELIMITER" is not a valid "COLUMN" option for "<column-name>"	The DELIMITER option was specified for column <column-name>, but the IS_REPEATING option was not specified.
SQL1882N	The "SERVER" option "RDBMS_TYPE" cannot be set to "<option-value>" for "<server-name>".	The value specified for the RDBMS_TYPE server option is invalid. It must be one of the following: DB2, INFORMIX, ORACLE, SQLSERVER or SYBASE.
SQL1882N	The "SERVER" option "TRANSACTIONS" cannot be set to "<option-value>" for "<server-name>".	The value specified for the TRANSACTIONS server option is invalid. It must be one of the following: NONE, QUERY, PASSTHRU or ALL.
SQL1882N	The "NICKNAME" option "IS_REG_TABLE" cannot be set to "<option-value>" for "<nickname>".	The value specified for the IS_REG_TABLE nickname option is invalid. It must be one of the following: ’Y’ or ’N’.
SQL1882N	The "NICKNAME" option "ALL_VERSIONS" cannot be set to "<option-value>" for "<nickname>".	The value specified for the ALL_VERSIONS nickname option is invalid. It must be one of the following: ’Y’ or ’N’.
SQL1882N	The "SERVER" option "OS_TYPE" cannot be set to "<option-value>" for "<server-name>"	The value specified for the OS_TYPE server option is invalid. It must be: AIX, HPUX, SOLARIS or WINDOWS.

Table 45. Messages issued by the wrapper for Documentum (continued)

Error Code	Message	Explanation
SQL1882N	The "NICKNAME" option "FOLDERS" cannot be set to "<option-value>" for "<nickname>"	The value specified for the FOLDERS nickname option is invalid. It cannot be specified for a table where IS_REG_TABLE is 'Y'.
SQL1882N	The "NICKNAME" option "VERSIONS" cannot be set to "<option-value>" for "<nickname>"	The value specified for the VERSIONS nickname option is invalid. It must be one of the following: 'Y' or 'N'. The VERSIONS option cannot be set to 'Y' for a table where the IS_REG_TABLE option is set to 'Y'.
SQL30090N	Operation invalid for application execution environment. Reason code = "Invalid column name, IS_REG_TABLE, or IS_REPEATING specified in nickname"	Check the nickname statement for the correct specification of the IS_REG_TABLE, IS_REPEATING, REMOTE_NAME options, and column names.
SQL30090N	Operation invalid for application execution environment. Reason code = "Failed to open log file for debugging"	The log file used for troubleshooting is not accessible. Contact your system administrator.
SQL30090N	Operation invalid for application execution environment. Reason code = "Only one search condition may be specified"	Only one custom search function can be specified per query.
SQL30090N	Operation invalid for application execution environment. Reason code = "Failed to create content directory"	Make sure the destination directory is writable by the DB2 agent.
SQL30090N	Operation invalid for application execution environment. Reason code = "Failed to change permissions on content file"	Make sure the target content directory is writable by the DB2 agent.
SQL5182N	A required environment variable, "DMCL_CONFIG", has not been set.	Neither the DOCUMENTUM nor DMCL_CONFIG environment variable was set. Set them in the db2dj.ini file.

Related concepts:

- "Introduction to messages" in the *Message Reference Volume 1*

Related reference:

- "SQLSTATE messages" in the *Message Reference Volume 2*

Chapter 11. Configuring access to Entrez data sources

This chapter explains how to configure your federated server to access data that is stored in Entrez data sources. You can configure access to Entrez data sources by using the DB2 Control Center or by issuing SQL statements.

This chapter:
- Explains what Entrez is
- Lists the tasks that you need to perform
- Contains examples of the SQL statements that you need
- Lists the error messages associated with the Entrez wrapper

What is Entrez?

Entrez is a query and retrieval system developed by the National Center for Biotechnology Information (NCBI). You can use Entrez to access several linked databases hosted by the NCBI.

These databases include:
- PubMed (biomedical literature)
- Nucleotide (a sequence database also called GenBank)
- OMIM (Online Mendelian Inheritance in Man from John Hopkins University)
- Genome (complete genome assemblies)

You can access all of the Entrez databases through a uniform set of Web-based tools. The Entrez wrapper uses these tools to federate the Entrez databases into the DB2® environment. Although the Entrez interface supports many databases, the Entrez wrapper supports only PubMed and Nucleotide.

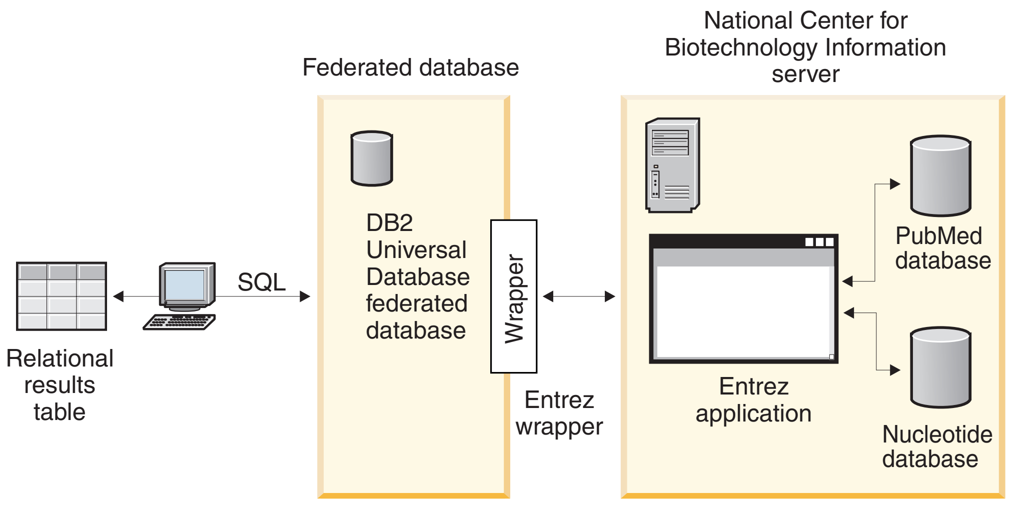


Figure 22. How the Entrez wrapper works

Many elements of the Entrez wrapper are common to all of the databases. These elements include:

- Connectivity with NCBI through the Web and the Entrez ESearch and EFetch utilities
- Mapping of hierarchical XML data into relational tables
- Joins between related tables through the XML wrapper technology

Related tasks:

- "Adding Entrez data sources to a federated server" on page 194

Adding Entrez to a federated server

Adding Entrez data sources to a federated server

To configure the federated server to access Entrez data sources, you must provide the federated server with information about the data sources and objects that you want to access.

| You can configure the federated server to access Entrez data sources by using the DB2 Control Center or the DB2 command line. The DB2 Control Center includes a wizard to guide you through the steps required to configure the federated server.

The Entrez wrapper can access PubMed and Nucleotide data sources in networks that use firewalls with proxies. The proxies that are supported are: HTTP, SOCKS4, and SOCKS5.

Prerequisites:

- DB2 Information Integrator must be installed on a server that will act as the federated server
- A federated database must exist on the federated server

Procedure:

To add Entrez data sources to a federated server:

1. Register custom functions for the Entrez wrapper.
2. Register the wrapper.
3. Register the server definition.
4. Register nicknames for Entrez databases.

Related concepts:

- "DB2 Information Integrator installation process - overview" in the *IBM DB2 Information Integrator Installation Guide for Linux, UNIX, and Windows*

Related tasks:

- "Creating a federated database" on page 51
- "Registering the custom functions for the Entrez wrapper" on page 195

Related reference:

- "Supported operating systems for DB2 Information Integrator (32-bit)" in the *IBM DB2 Information Integrator Installation Guide for Linux, UNIX, and Windows*
- "Supported operating systems for DB2 Information Integrator (64-bit)" in the *IBM DB2 Information Integrator Installation Guide for Linux, UNIX, and Windows*

Registering the custom functions for the Entrez wrapper

Registering custom functions for the Entrez wrapper is part of the larger task of adding Entrez to a federated system. After the custom functions are registered, you must register the wrapper.

Restrictions:

- All of the custom functions for the Entrez wrapper must be registered with the schema name entrez.
- You must register each custom function once for each DB2 database that has the Entrez wrapper installed.

Procedure:

To register custom functions, issue the CREATE FUNCTION statement with the AS TEMPLATE DETERMINISTIC NO EXTERNAL ACTION keywords.

The fully qualified name of each function is entrez.function_name.

The following example registers one version of the CONTAINS function:

```
CREATE FUNCTION entrez.contains (varchar(), varchar())
   RETURNS INTEGER AS TEMPLATE
   DETERMINISTIC NO EXTERNAL ACTION;
```

To register the custom functions, use the sample file create_function_mappings.ddl. The sample file is in the path where DB2 Information Integrator is installed, under the samples/lifesci/entrez directory. The sample file contains definitions for each of the custom function. You can run this DDL file to register the custom functions on each federated database that has the Entrez wrapper installed.

The next task in this sequence of tasks is registering the Entrez wrapper.

Related reference:

- "CREATE FUNCTION (Sourced or Template) statement" in the *SQL Reference, Volume 2*
- "Custom functions and Entrez queries" on page 201
- "Custom function table - Entrez wrapper" on page 195

Custom function table - Entrez wrapper

You use the CREATE FUNCTION statement to register the Entrez custom functions.

The following table list the Entrez custom functions and the data types of the arguments that you specify when you register the functions. The first argument specified in the function is for the column name of a tagged column. The second argument specified in the function is the search term.

Table 46. Custom functions for the Entrez wrapper

Function	Description
entrez.contains (varchar(), varchar()) entrez.contains (integer, varchar()) entrez.contains (smallint, varchar()) entrez.contains (real, varchar()) entrez.contains (double, varchar()) entrez.contains (date, varchar()) entrez.contains (time, varchar()) entrez.contains (char(), varchar()) entrez.contains (timestamp(), varchar())	Searches a tagged column using the term that you specify.
entrez.search_term (char(), varchar())	Passes an Entrez search term directly to the Entrez search engine.

To register the custom functions, use the sample file `create_function_mappings.ddl`. The sample file is installed in the `samples/lifesci/entrez` directory.

Related tasks:

- "Registering the custom functions for the Entrez wrapper" on page 195

Related reference:

- "Custom functions and Entrez queries" on page 201

Registering the Entrez wrapper

Registering the Entrez wrapper is part of the larger task of adding Entrez data sources to a federated server.

You must register a wrapper to access Entrez data sources. Wrappers are used by federated servers to communicate with and retrieve data from data sources. Wrappers are implemented as a set of library files.

Procedure:

To register a wrapper, issue the CREATE WRAPPER statement with the name of the wrapper and the name of the wrapper library file.

For example, to register a wrapper with the name `entrez_wrapper` on the federated server that uses the AIX operating system, issue the following statement:

```
CREATE WRAPPER entrez_wrapper LIBRARY 'libdb2lsentrez.a'
  OPTIONS(EMAIL 'jeff@someplace.com');
```

You must specify an e-mail address when you register an Entrez wrapper. This e-mail address is included with all queries and allows NCBI to contact you if there are problems, such as too many queries overloading the NCBI servers.

The name of the wrapper library file that you specify depends on the operating system of the federated server. See the list of Entrez wrapper library files for the correct name to specify in the CREATE WRAPPER statement.

The next task in this sequence of tasks is registering the server definition for Entrez data sources.

Related reference:

- "Entrez wrapper library files" on page 197
- "CREATE WRAPPER statement" in the *SQL Reference, Volume 2*

Entrez wrapper library files

The following table lists the directory paths and library file names for the Entrez wrapper.

When you install DB2 Information Integrator, 3 library files are added to the directory path listed in the table. For example, if the federated server is running on AIX, the wrapper library files added to the directory path are `libdb2lsentrez.a`, `libdb2lsentrezF.a`, and `libdb2lsentrezU.a`.

When you register a wrapper, specify only the library file name that is listed in the table.

Table 47. Entrez wrapper library locations and file names

Operating system	Directory path	Wrapper library file
AIX	/usr/opt/db2_08_01/lib/	libdb2lsentrez.a
Linux	/opt/IBM/db2/V8.1/lib	libdb2lsentrez.so
Solaris	/opt/IBM/db2/V8.1/lib	libdb2lsentrez.so
Windows	%DB2PATH%\bin	db2lsentrez.dll

%DB2PATH% is the environment variable that is used to specify the directory path where DB2 Information Integrator is installed on Windows. The default Windows directory path is `C:\Program Files\IBM\SQLLIB`.

Related tasks:

- "Registering the Entrez wrapper" on page 196

Registering the server for an Entrez data source

Registering the server definition for an Entrez data source is part of the larger task of adding Entrez to a federated server.

After you register the wrapper, you must register a corresponding server definition

The database, PubMed or Nucleotide, that is represented by a particular data source is identified by the server type value, as expressed on the CREATE SERVER statement. This server type value controls the structure of any nicknames that are created.

Procedure:

To register the Entrez server to the federated system, issue a CREATE SERVER statement.

For example, to register a server named `pubmed_server1` for the `entrez_wrapper` wrapper, issue the following statement:

```
CREATE SERVER pubmed_server1
    TYPE PUBMED
    VERSION 1.0
    WRAPPER entrez_wrapper;
```

Additionally, to register a server named `nucleotid_server1` for the `entrez_wrapper` wrapper, issue the following statement:

```
CREATE SERVER nucleotid_server1
    TYPE NUCLEOTIDE
    VERSION 1.0
    WRAPPER entrez_wrapper;
```

Limiting the number of rows that are retrieved for Entrez queries

The MAX_ROWS server option can be used to limit the number of rows that are returned for a query that uses the Entrez wrapper.

Unlike the FETCH FIRST N ROWS ONLY clause of an SQL statement, which limits the number of rows that are returned to a user or an application, the MAX_ROWS server option enables you to limit the number of rows that can be retrieved from the NCBI Web site.

The MAX_ROWS option value is always used as an upper (maximum) limit to the number of rows that a query can retrieve. If a query attempts to retrieve more rows than what is specified in the MAX_ROWS option, the result set is truncated, and a warning message is issued.

You can set the MAX_ROWS server option when a server is created, or you can use the ALTER SERVER statement to change the option value.

The MAX_ROWS server option is not required. If you do not set the option, a default value is used. The specific default value that is used depends on your operating system. For Microsoft Windows operating systems, the default value is 2000 rows. For UNIX-based operating systems, the default value is 5000 rows.

You can specify only positive numbers and 0 (zero). When you set the option to be 0 (zero), you enable queries to retrieve an unlimited number of rows from the NCBI Web site. However, setting the MAX_ROWS server option to 0 (zero) or to a very high number can possibly impact your query performance.

Accessing Entrez through a proxy server

To access Entrez data sources through a proxy server, you must specify options when you create the server definition. The options that you specify depend on the type of proxy server that you want to access.

Example of registering a server definition for an HTTP proxy server:

To register a server definition and specify an HTTP proxy server, use the following statement:

```
CREATE SERVER pubmed_server_h
    TYPE PUBMED
    VERSION 1.0
    WRAPPER entrez_wrapper
    OPTIONS (PROXY_TYPE 'HTTP', PROXY_SERVER_NAME 'proxy_h',
        PROXY_SERVER_PORT '8080');
```

Example of registering a server definition for a SOCKS4 proxy server:

To register a server definition and specify a SOCKS4 proxy server, use the following statement:

```
CREATE SERVER pubmed_server_s4
    TYPE PUBMED
    VERSION 1.0
    WRAPPER entrez_wrapper
    OPTIONS (PROXY_TYPE 'SOCKS4', PROXY_SERVER_NAME 'proxy_4',
        PROXY_SERVER_PORT '1080');
```

Example of registering a server definition for a SOCKS5 proxy server without authentication information:

To register a server definition and specify a SOCKS5 proxy server without authentication information, use the following statement:

```
CREATE SERVER pubmed_server_s5
    TYPE PUBMED
    VERSION 1.0
    WRAPPER entrez_wrapper
    OPTIONS (PROXY_TYPE 'SOCKS5', PROXY_SERVER_NAME 'proxy_5',
        PROXY_SERVER_PORT '1081');
```

Example of registering a server definition for a SOCKS5 proxy server with authentication information:

To register a server definition and specify a SOCKS5 proxy server with authentication information, use the following statement:

```
CREATE SERVER pubmed_server_s5a
    TYPE PUBMED
    VERSION 1.0
    WRAPPER entrez_wrapper
    OPTIONS (PROXY_TYPE 'SOCKS5', PROXY_SERVER_NAME 'proxy_5',
        PROXY_SERVER_PORT '1081', PROXY_AUTHID 'Khalid',
        PROXY_PASSWORD 'aaa', );
```

The next task in this sequence of tasks is registering nicknames for Entrez data sources.

Related tasks:

- “Registering nicknames for Entrez data sources” on page 199

Related reference:

- “CREATE SERVER statement arguments - Entrez wrapper” on page 552

Registering nicknames for Entrez data sources

Registering nicknames for Entrez data sources is part of the larger task of adding Entrez to a federated system.

Restrictions:

The schema for each Entrez database is fixed by the wrapper and cannot be changed or amended. For each database, there is a fixed set of tables with a fixed list of columns for each table. The tables in a database have a hierarchical relationship. One table, which is the parent of all of the other tables in the database, is called the root (parent) table. All of the other tables in the database have a parent-child relationship that leads back to the root table.

Procedure:

To register nicknames for Entrez data sources, issue a CREATE NICKNAME statement.

Because the list of columns for the nicknames is fixed and is supplied by the wrapper, the basic syntax to create Nucleotide nicknames is simple. For example:

```
CREATE NICKNAME GBSeq FOR SERVER nuc1;
CREATE NICKNAME GBFeatures FOR SERVER nuc1;
CREATE NICKNAME GBIntervals FOR SERVER nuc1;
CREATE NICKNAME GBQualifiers FOR SERVER nuc1;
CREATE NICKNAME GBReference FOR SERVER nuc1;
```

Here is an example of the basic syntax to create PubMed nicknames:

```
CREATE NICKNAME pmarticles FOR SERVER pubmed_server;
CREATE NICKNAME PMACCESSION FOR SERVER pubmed_server;
CREATE NICKNAME PMCHEMICAL FOR SERVER pubmed_server;
CREATE NICKNAME PMMESH FOR SERVER pubmed_server;
CREATE NICKNAME PMCOMMENTS FOR SERVER pubmed_server;
CREATE NICKNAME PMARTICLEID FOR SERVER pubmed_server;
```

The name of the nickname is the name of the underlying table.

Use of this syntax limits you to one family of nicknames for each DB2 schema. You can use other names by using the nickname options REMOTE_OBJECT and PARENT. For a root nickname, only REMOTE_OBJECT is required. For any other nickname, both REMOTE_OBJECT and PARENT must be provided.

The following example shows the same set of Nucleotide nicknames using the renaming capability:

```
CREATE NICKNAME NewSeq FOR SERVER nuc1 OPTIONS (REMOTE_OBJECT 'GBSEQ');
CREATE NICKNAME NewFeatures FOR SERVER nuc1
 OPTIONS (REMOTE_OBJECT 'GBFEATURES', PARENT 'NEWSEQ');
CREATE NICKNAME NewIntervals FOR SERVER nuc1
 OPTIONS (REMOTE_OBJECT 'GBINTERVALS', PARENT 'NEWFEATURES');
CREATE NICKNAME NewQualifiers FOR SERVER nuc1
 OPTIONS (REMOTE_OBJECT 'GBQUALIFIERS', PARENT 'NEWFEATURES');
CREATE NICKNAME NewReference FOR SERVER nuc1
 OPTIONS (REMOTE_OBJECT 'GBREFERENCE', PARENT 'NEWSEQ');
```

This example shows the same set of PubMed nicknames using the renaming capability:

```
CREATE NICKNAME newpmarticles FOR SERVER pubmed_server
 OPTIONS (REMOTE_OBJECT 'PMARTICLES');
CREATE NICKNAME NEWPMACCESSION FOR SERVER pubmed_server
 OPTIONS (REMOTE_OBJECT 'PMACCESSION', PARENT 'NEWPMARTICLES');
CREATE NICKNAME NEWPMCHEMICAL FOR SERVER pubmed_server
 OPTIONS (REMOTE_OBJECT 'PMCHEMICAL' , PARENT 'NEWPMARTICLES');
CREATE NICKNAME NEWPMMESH FOR SERVER pubmed_server
 OPTIONS (REMOTE_OBJECT 'PMMESH' , PARENT 'NEWPMARTICLES');
CREATE NICKNAME NEWPMCOMMENTS FOR SERVER pubmed_server
 OPTIONS (REMOTE_OBJECT 'PMCOMMENTS' , PARENT 'NEWPMARTICLES');
CREATE NICKNAME NEWPMARTICLEID FOR SERVER pubmed_server
 OPTIONS (REMOTE_OBJECT 'PMARTICLEID' , PARENT 'NEWPMARTICLES');
```

The next task in this sequence of tasks is to register custom functions for Entrez data sources.

Related tasks:

- "Adding Entrez data sources to a federated server" on page 194

- “Registering the custom functions for the Entrez wrapper” on page 195
- “Specifying nickname columns for a nonrelational data source” on page 65

Related reference:
- “CREATE NICKNAME statement” in the *SQL Reference, Volume 2*
- “PubMed schema tables” on page 205
- “Nucleotide schema tables” on page 209

Queries and custom functions for Entrez data sources

Custom functions and Entrez queries

The federated environment uses two query engines. For the Entrez wrapper, these query engines are DB2 and Entrez. With one exception, you specify all predicates for the Entrez engine through custom functions. For the DB2 engine, you specify all predicates through the relational operators.

The main custom function is ENTREZ.CONTAINS. The CONTAINS function requires a search term column argument and a query text argument. The following example shows an ENTREZ.CONTAINS statement:

```
ENTREZ.CONTAINS (search_term_column, query_text)
```

A tag in the Q column of the schema tables identifies a search term. The query text must be in the modified Entrez query syntax. This syntax consists of search terms separated by Boolean operators (OR, AND, and NOT) and grouped by using parentheses. The syntax of the CONTAINS query text argument differs from the standard Entrez query syntax in that search term qualifiers, such as `[pd]`, are not allowed.

The custom functions are registered in the Entrez schema, which must be used to refer to the functions. When the custom functions are used, their return value must be compared to the value 1 in an equality predicate.

In some situations, DB2 and Entrez predicates might be mixed in such a way that they cannot be processed. These cases generate the error message SQL0142N (″SQL statement not supported″).

For example, in the following query, you cannot separate the parts of the predicate that are processed by the wrapper (the ENTREZ.CONTAINS invocations) and the parts that must be processed by DB2 (the relational predicate on BaseCountA).

```
WHERE
 ENTREZ.CONTAINS (Organism, 'drosophila') = 1
 OR (BaseCountA > 10 AND ENTREZ.CONTAINS (Keywords, 'glop') = 1)
```

Some search fields do not have corresponding columns in the Entrez schema. For example, in the nucleotide database, the term [ALL] searches all searchable fields, while [WORD] searches all of the free text associated with a record. Pseudo-columns are provided for these search terms. If a pseudo-column is referenced in a select list, a value of NULL is returned.

You can run queries that might not otherwise be possible by issuing the ENTREZ.SEARCH_TERM master function. If you specify the ENTREZ.SEARCH_TERM master function, it must be the only custom function in a query. For each query, there can be only one ENTREZ.SEARCH_TERM master function per Entrez nickname. Also, SEARCH_TERM and CONTAINS functions cannot be mixed for the same nickname in the same query. The first argument, column specification, must be the primary key column for the parent nickname. The second argument, query text, is an Entrez-format search term that includes search field qualifiers. This text is passed unmodified, except for URI escapes as required by the URI syntax, to Entrez.

The following example shows a query with a WHERE clause on a PubMed nickname:

```
WHERE
 ENTREZ.CONTAINS (authors, 'kaufmann OR ito AND NOT rakesh')
 AND
 (ENTREZ.CONTAINS (title, 'drosophila')
   OR
 ENTREZ.CONTAINS(alltext, 'drosophila OR "fruit fly"'))
```

In this example, the individual predicates are `authors`, `title`, and `all text`.

The individual predicates are modified so that the qualifier is added after each search term . Then, the terms are grouped with parentheses to enforce the DB2 Boolean operator precedence. Because of these modifications, the `authors` predicate becomes:

```
((kaufmann[auth] OR ito[auth]) AND (NOT (rakesh[auth])))
```

The `title` predicate becomes:

```
(drosophila[titl])
```

And the `all text` predicate becomes:

```
(drosophila[all] OR "fruit fly"[all])
```

When the individual predicates are combined, parentheses are used to maintain DB2 Boolean operator precedence. Excluding text transformations that are necessary to express the string as part of a URI, the final search term string submitted to Entrez is:

```
((kaufman[auth] OR ito[auth]) AND (NOT (rakesh[auth))) AND
((drosophila[titl]) OR (drosophila[all] OR "fruit fly"[all])
```

Related reference:

- "Custom function table - Entrez wrapper" on page 195

Relational predicates for the Entrez wrapper

The Entrez wrapper supports relational predicates, such as =, BETWEEN, LIKE, and <>, on nickname columns. However, the Entrez search engine processes only a few of these relational predicates. Relational predicates that are not processed by the Entrez search engine are processed by DB2®. The Entrez search engine processes equality (=) and IN predicates on certain ID columns for each schema. These predicates allow the Entrez wrapper to bypass the search phase and execute the fetch phase directly. Examples of valid predicates are:

```
WHERE pmid = '1234567'
```

```
WHERE medlineid IN ('1234567', '9191919')
```

Columns that can be used in this kind of predicate are identified by the F column of the schema tables. The value of this option must be Y.

Related concepts:
- "Invalid WHERE clauses for the Entrez wrapper" on page 203

Related tasks:
- "Entrez data source - Example queries" on page 204
- "Registering the custom functions for the Entrez wrapper" on page 195

Invalid WHERE clauses for the Entrez wrapper

The Entrez wrapper rejects any query that will result in an unqualified scan of the NCBI database. A valid WHERE clause must contain either an equality (or IN) predicate on the primary ID for the schema, or a custom function. Queries that do not meet these criteria are rejected with error code SQL0142N or SQL30090N.

Related concepts:
- "Relational predicates for the Entrez wrapper" on page 202

Related tasks:
- "Entrez data source - Example queries" on page 204
- "Registering the custom functions for the Entrez wrapper" on page 195

Schema data element simplification

Several data elements are converted to a canonical form when they are presented through the SQL schema. These data elements include item lists, names, and dates.

Item lists

Unless otherwise noted, lists of items that are denormalized into a single column have individual items separated by a semicolon and a single space. For example, if an entry contains the keywords dnaA gene, dnaN gene, and orf187, the corresponding Keywords column will contain the value dnaA gene; dnaN gene; orf187.

Names

Names in the NCBI schemas consist of a required last name and one of several optional elements. Some of these optional elements can occur together and others are exclusive of each other. To create a canonical form of a name, assign a precedence to these elements. In order from highest to lowest, these elements are:
- Forename
- First or middle name
- Initials

You can present names with or without affiliations. Without an affiliation, a name is formatted as <last name>, <first>, where <first> is one of the optional elements. If the <first> element is not found, then the comma is not used. An affiliation can be added in the form (<affiliation>).

Separate names in denormalized lists with a semicolon and a space. An example of the correct way to separate names is:

```
Parker, M. J.; Ranjan, K. A.
```

Dates

Dates, especially publication dates, come in a wide variety of formats in the NCBI schemas. To accommodate these formats and allow for date comparisons and date arithmetic where possible, dates in the SQL schema are represented in two forms. First, a date can be a character string. Second, a date can be a column of type DATE.

If only a month is present in a date value without reference to a day, the first day of the month is the default day. If a season is present rather than a month, or a month and day, the first day of the season is used.

Related tasks:

- "Registering nicknames for Entrez data sources" on page 199

Related reference:

- "Custom functions and Entrez queries" on page 201

Entrez data source - Example queries

This topic provides some sample queries to run on Entrez data sources.

Procedure:

To run queries, use the following examples as a guide.

On PubMed nicknames:

The following shows a query with a single fetch key on a PubMed nickname:

```
select PMID, ArticleTitle FROM pmarticles WHERE pmid = '12345';
```

The following shows a query with mixed fetch keys on a PubMed nickname:

```
select PMID, ArticleTitle FROM pmarticles
 WHERE pmid = '12345' OR MedlineID = '12346';
```

The following shows a query with a CONTAINS function on a PubMed nickname:

```
select PMID, ArticleTitle FROM pmarticles
 WHERE entrez.contains (ArticleTitle, 'granulation') = 1
 AND entrez.contains (PubDate, '1992') = 1;
```

The following shows a query that searches for the specified AuthorList and LanguageList on a PubMed nickname:

```
select PMID, ArticleTitle FROM pmarticles
 WHERE entrez.contains (AuthorList, 'Albarrak') = 1
 AND entrez.contains (LanguageList, 'eng')=1;
```

The following shows a query with a complex predicate on a PubMed nickname:

```
select PMID, ArticleTitle FROM pmarticles
 WHERE entrez.contains (PublicationTypeList, 'Journal Article') = 1
 AND entrez.contains (MedlineTA, 'sun')=1
 OR entrez.contains (PersonalNameSubjectList, 'shine')=1;
```

On Nucleotide nicknames:

The following shows a query with multiple fetch keys on a Nucleotide nickname:

```
select PrimaryAccession, LocusName, SeqLength  from gbseq
 WHERE PrimaryAccession in ('NM_000890', 'NC_003106');
```

The following shows a query that searches all of the searchable fields on a Nucleotide nickname:

```
select PrimaryAccession, substr(Definition,1,300), GI from gbseq
 WHERE entrez.contains(AllText, 'abcde')=1;
```

The following shows a query that searches all of the free text on a Nucleotide nickname:

```
select * from gbseq WHERE entrez.contains(FreeText, 'abcde')=1;
```

The following shows a query that searches for a definition on a Nucleotide nickname:

```
select PrimaryAccession, substr(Definition,1,300), version, GI from gbseq
 WHERE entrez.contains(Definition, 'Sulfolobus tokodaii
   AND complete genome') = 1;
```

The following shows a query that searches for a keyword on a Nucleotide nickname:

```
select PrimaryAccession, substr(KeywordList,1,200), Segment from gbseq
 WHERE entrez.contains(KeywordList, 'nkcc1 gene') = 1;
```

Related concepts:

- "Relational predicates for the Entrez wrapper" on page 202
- "Invalid WHERE clauses for the Entrez wrapper" on page 203

Related tasks:

- "Registering the custom functions for the Entrez wrapper" on page 195

PubMed schema tables

The PubMed schema defines the appearance of data from a PubMed type server. The schema consists of several related nicknames:

- PMArticles
- PMAccession
- PMChemical
- PMMeSHHeading
- PMComments
- PMArticleID

The following tables list information about the columns in each nickname. The Tags column contains the valid search tags for the column. For a list of valid search tags, see the following web site and locate the link to Search Field Descriptions and Tags:

`www.ncbi.nlm.nih.gov/entrez/query/static/help/pmhelp.html`

You can override the default data type for a column when you create a nickname. The Entrez wrapper supports the CLOB data type, up to 5 megabytes in length.

You can override the default length for a column when you create a nickname. For example, some columns can return a large amount of data, such as the Abstract column in the PMArticles nickname. The default length for this column is VARCHAR(32000). To return the first 100 bytes of the column, you can define the column with the data type VARCHAR(100). Only the first 100 bytes will be returned.

PMArticles nickname:

The columns in the PMArticles nickname are described in the following table. The F column indicates columns that are designated fetch keys. Using the fetch keys might expedite query processing.

Table 48. PubMed PMArticles nickname

Column name	Data type	Description	Tags	Fetch key
PMID	VARCHAR(10) NOT NULL	Primary key column used to join the PMArticles nickname with child nicknames	UID	Yes
MedlineID	VARCHAR(10)	Medline ID	UID	Yes
Owner	VARCHAR(8) NOT NULL	Owner of the publication entry; values are defined by NCBI and might be NLM, NASA, PIP, KIE, HSR, HMD, SIS, NOTNLM. If not present, then the default is NLM.	none	No
Status	VARCHAR(32) NOT NULL	Publication status as defined by NCBI. Values might include: In-Process, Completed, Out-of-scope, PubMed-not_MEDLINE.	none	No
DateCreated	DATE NOT NULL	Date that the publication entry was created.	none	No
DateCompleted	DATE	Date that the publication entry was completed.	none	No
DateRevised	DATE	Date that the publication entry was revised.	none	No
ArticleTitle	VARCHAR(250) NOT NULL	The title of the article.	TI	No
Pagination	VARCHAR(32)	The full pagination of the article.	none	No
Abstract	VARCHAR(32000)	The abstract for the article.	TIAB	No
Affiliation	VARCHAR(250)	Affiliation and address of first author	AD	No
AuthorList	VARCHAR(3200)	List of authors; canonized	AU	No
LanguageList	VARCHAR(250) NOT NULL	Semicolon-separated list	LA	No
PublicationTypeList	VARCHAR(250) NOT NULL	Semicolon-separated list	PT	No
VernacularTitle	VARCHAR(250)	The vernacular title for the article.	none	No
DateOfElectronic Publication	VARCHAR(32)	The NCBI schema specifies no structure for this column	none	No
Country	VARCHAR(128)	The country or region of publication that is cited in the journal.	none	No

Table 48. PubMed PMArticles nickname (continued)

Column name	Data type	Description	Tags	Fetch key
MedlineTA	VARCHAR(250) NOT NULL	The Medline title abbreviation.	TA	No
NlmUniqueId	VARCHAR(32)	Contains MedlineCode if NlmUniqueID is not present	none	No
GeneSymbolList	VARCHAR(250)	Semicolon-separated list; not used since 1996	none	No
NumberOfReferences	INTEGER	The number of bibliographic references for the review article.	none	No
PersonalNameSubjectList	VARCHAR(250)	Canonized as semicolon-separated list of names	PS	No
KeywordList	VARCHAR(3200)	Semicolon-separated list	none	No
SpaceFlightMissionList	VARCHAR(250)	Semicolon-separated list	none	No
InvestigatorList	VARCHAR(250)	Canonized as semicolon-separated list of names	none	No
PublicationStatus	VARCHAR(32)	The status of the publication.	none	No
ProviderID	VARCHAR(32)	The publication provider ID.	none	No
CitationSubsetList	VARCHAR(250)	Semicolon-separated list	SB	No
AllFields	VARCHAR(1)	Pseudo-column; always returns NULL	ALL	No
TextWords	VARCHAR(1)	Pseudo-column; always returns NULL	TW	No
PubDate	DATE	Includes journal and book publication date + medline date	DP	No
PubDateString	VARCHAR(32)	Includes journal and book publication date + medline date	DP	No
Title	VARCHAR(250)	Book or journal title	TA	No
Journal_ISSN	CHAR(9)	The ISSN for the journal.	TA	No
Journal_Volume	VARCHAR(10)	The volume of the journal.	VI	No
Journal_Issue	VARCHAR(10)	The issue of the journal.	IP	No
Journal_Coden	VARCHAR(32)	The code number (coden) of the journal.	none	No
Journal_ISOAbbreviation	VARCHAR(32)	The ISO abbreviation for the journal.	none	No
Book_Publisher	VARCHAR(128)	The publisher the book.	none	No
Book_Authors	VARCHAR(250)	Canonized as other author lists	none	No
Book_CollectionTitle	VARCHAR(128)	The collection title of the book.	none	No

Table 48. PubMed PMArticles nickname (continued)

Column name	Data type	Description	Tags	Fetch key
Book_Volume	VARCHAR(10)	The volume of the book.	none	No

PMAccession nickname:

The columns in the PMAccession nickname are described in the following table.

Table 49. PubMed PMAccession nickname

Column name	Data type	Description	Tags
PMID	VARCHAR(10) NOT NULL	The key that is used to join a PMAccession child nickname with its parent nickname.	none
DataBankName	VARCHAR(250) NOT NULL	The name of the data bank.	SI
Accession	VARCHAR(32) NOT NULL	The accession number.	SI

PMChemical nickname:

The columns in the PMChemical nickname are described in the following table.

Table 50. PubMed PMChemical nickname

Column name	Data type	Description	Tags
PMID	VARCHAR(10) NOT NULL	The key that is used to join a PMChemical child nickname with its parent nickname.	none
NameOfSubstance	VARCHAR(128) NOT NULL	The name of the substance.	NM
RegistryNumber	VARCHAR(32) NOT NULL	Might be CAS or other registry number	RN
CASRegistry	CHAR	Y or N	none

PMMeSHHeading nickname:

The columns in the PMMeSHHeading nickname are described in the following table.

Table 51. PubMed PMMeSHHeading nickname

Column name	Data type	Description	Tags
PMID	VARCHAR(10) NOT NULL	The key that is used to join a PMMeSHHeading child nickname with its parent nickname.	ID
DescriptorOrName	VARCHAR(128) NOT NULL	The name or descriptor of the MeSH.	MH[1]
DescriptorIsMajor	CHAR NOT NULL	Y if descriptor is major	none
QualifierOrSubhead	VARCHAR(128)	The qualifier or subheading of the MeSH .	SH
QSIsMajor	CHAR	Y if qualifier or subhead is major	none

Table 51. PubMed PMMeSHHeading nickname (continued)

Column name	Data type	Description	Tags

Notes:

1. If the predicate ″DescriptorIsMajor = Y″ is included in the query, then the search term is MAJR.

PMComments nickname:

The columns in the PMComments nickname are described in the following table.

Table 52. PubMed PMComments nickname

Column name	Data type	Description	Tags
PMID	VARCHAR(10) NOT NULL	The key that is used to join a PMComments child nickname with its parent nickname.	none
RefSource	VARCHAR(128) NOT NULL	The source of the reference.	none
Type	VARCHAR(32) NOT NULL	CommentOn, CommentIn, ErratumIn, ErratumFor, RepublishedFrom, RepublishedIn, RetractionOf, RetractionIn, UpdateIn, UpdateOf, SummaryForPatents, OriginalReportIn	none
Note	VARCHAR(3200)	Notes	none

PMArticleID nickname:

The columns in the PMArticleID nickname are described in the following table.

Table 53. PubMed PMArticleID nickname

Column name	Data type	Description	Tags
PMID	VARCHAR(10) NOT NULL	The key that is used to join a PMArticleID child nickname with its parent nickname.	none
ArticleID	VARCHAR(32) NOT NULL	The ID of the article.	none
IdType	VARCHAR(8) NOT NULL	doi, pii, pmcpid, pmpid, sici, pubmed, medline, pmcid	none

Related reference:

- "Schema data element simplification" on page 203
- "Custom functions and Entrez queries" on page 201
- "Nucleotide schema tables" on page 209

Nucleotide schema tables

| The Nucleotide schema defines the appearance of data from a Nucleotide type
| server. The schema consists of several related nicknames:

| • GBSeq
| • GBReference

- GBFeatures
- GBIntervals
- GBQualifiers

The following tables list information about the columns in the nickname. The Qualifier column contains the valid search qualifiers for the column. For a list of valid search tags, see the following web site and locate the link to Search Field Descriptions and Qualifiers:

`www.ncbi.nlm.nih.gov/entrez/query/static/help/Summary_Matrices.html`

You can override the default data type for a column when you create a nickname. For example, the default data type for the Sequence column in the GBSeq nickname VARCHAR(32000). You can change this data type to CLOB(1 MB). The Entrez wrapper supports the CLOB data type, up to 5 megabytes in length.

GBSeq:

The columns in the GBSeq nickname are described in the following table. The F column indicates columns that are designated fetch keys. Using the fetch keys might expedite query processing.

Table 54. Nucleotide GBSeq nickname

Column name	Data type	Description	Qualifier	Fetch key
PrimaryAccession	VARCHAR(16) NOT NULL	Primary accession number	PACC	Yes
SequenceKey	VARCHAR(32) NOT NULL	The primary key column used to join the GBSeq nickname with child nicknames.		No
LocusName	VARCHAR(16) NOT NULL	The name of the locus.	ACCN	No
SeqLength	INTEGER NOT NULL	The length of the sequence.	SLEN	No
Strandedness	VARCHAR(32)	not-set, single-stranded, double-stranded, mixed-stranded	none	No
MoleculeType	VARCHAR(16)	nucleic-acid, dna, rna, trna, rrna, mrna, urna, snrna, snorna, peptide	PROP	No
Topology	VARCHAR(16)	linear, circular	none	No
Division	CHAR(3) NOT NULL	The GenBank division.	PROP	No
UpdateDate	DATE NOT NULL	The date of the most recent update.	MDAT	No
CreateDate	DATE NOT NULL	The date that the record was created.	none	No
Definition	VARCHAR(7000) NOT NULL	The definition line of the sequence.	TITL	No
Version	INTEGER	The version ID of the sequence.	none	No

Table 54. Nucleotide GBSeq nickname (continued)

Column name	Data type	Description	Qualifier	Fetch key
GI	VARCHAR(16)	The GenInfo (GI) sequence ID.	none	No
KeywordList	VARCHAR(7000)	Semicolon separated list	KYWD	No
Segment	VARCHAR(250)	The segment.	none	No
Source	VARCHAR(200) NOT NULL	The source.	ORGN	No
Organism	VARCHAR(7000) NOT NULL	The organism.	ORGN	No
Taxonomy	VARCHAR(7000) NOT NULL	The taxonomy.	none	No
Comment	VARCHAR(7000)	Comments	none	No
Primary	VARCHAR(7000)	The primary.	none	No
SourceDB	VARCHAR(250)	The source database.	none	No
Sequence	VARCHAR(32000)	The sequence.	none	No
AllText	VARCHAR(1)	Pseudo-column, always returns NULL	ALL	No
FreeText	VARCHAR(1)	Pseudo-column, always returns NULL	WORD	No

GBReference:

The columns in the GBReference nickname are described in the following table.

Table 55. Nucleotide GBReference nickname

Column name	Data type	Description	Qualifier
SequenceKey	VARCHAR(32) NOT NULL	The key that is used to join a GBReference child nickname with its parent nickname.	none
ReferenceNum	INTEGER NOT NULL	Parsed from GBReference_reference	none
RangeLow	INTEGER NOT NULL	Low base for reference (parsed from GBReference_reference)	none
RangeHigh	INTEGER NOT NULL	High base for reference (parsed from GBReference_reference)	none
Authors	VARCHAR(3200)	Semicolon-separated list of names in GenBank form	AUTH
Consortium	VARCHAR(250)	The consortium.	none
Title	VARCHAR(250)	The GenBank reference title.	WORD
Journal_Title	VARCHAR(250) NOT NULL	The title of the journal.	JOUR
MedlineID	INTEGER	The Medline ID	none
PubMedID	INTEGER	The PubMed ID	none
Remarks	VARCHAR(3200)	Remarks	none

GBFeatures:

The columns in the GBFeatures nickname are described in the following table.

Table 56. Nucleotide GBFeatures nickname

Column name	Data type	Description	Qualifier
SequenceKey	VARCHAR(32) NOT NULL	The key that is used to join a GBFeatures child nickname with its parent nickname.	none
FeatureJoinKey	VARCHAR(32) NOT NULL	The primary key column used to join the GBFeatures nickname with child nicknames.	none
FeatureKey	VARCHAR(20) NOT NULL		FKEY
FeatureLocation	VARCHAR(200) NOT NULL		none

GBIntervals:

The columns in the GBIntervals nickname are described in the following table.

Table 57. Nucleotide GBIntervals nickname

Column Name	Data type	Description	Qualifier
FeatureJoinKey	VARCHAR(32) NOT NULL	The key that is used to join a GBIntervals child nickname with its parent nickname.	none
IntervalFrom	INTEGER		none
IntervalTo	INTEGER		none
IntervalPoint	INTEGER		none
IntervalAccession	VARCHAR(32) NOT NULL		none

GBQualifiers:

The columns in the GBQualifiers nickname are described in the following table.

Table 58. Nucleotide GBQualifiers nickname

Column name	Data type	Description	Qualifier
FeatureJoinKey	VARCHAR(32) NOT NULL	The key that is used to join a GBQualifiers child nickname with its parent nickname.	none
QualifierName	VARCHAR(50)	Name of the qualifier	none
QualifierValue	VARCHAR(32000)	Value of the qualifier	none

Related reference:

- "Schema data element simplification" on page 203
- "PubMed schema tables" on page 205
- "Custom functions and Entrez queries" on page 201

Messages for the Entrez wrapper

This topic describes messages that you might encounter when working with the wrapper for Entrez. For messages that are not documented in this table, the *Message Reference: Volume 1*, or the *Message Reference: Volume 2*, contact IBM Software support.

Table 59. Messages issued by the wrapper for Entrez

Error Code	Message	Explanation
SQL0142N	The SQL statement is not supported.	An invalid query type was passed to the wrapper. Check to see if the issued SQL statement is supported by this wrapper.
SQL0204N	"<name>" is an undefined name.	The specified name is not valid. Check the CREATE NICKNAME statement.
SQL0405N	The numeric literal "<literal>" is not valid because its value is out of range.	A column in the retrieved XML data or a predicate in an SQL statement contains a value that is out of the possible range for that data type. Check the data type for this column and the column in the data source, or redefine the column to a more appropriate type.
SQL0408N	A value is not compatible with the data type of its assignment target. Target name is "<target_name>".	A column in the XML data contains characters that are not valid for that data type. Check the data type for this column and the column in the data source, or redefine the column to a more appropriate type.
SQL0901N	The SQL statement failed because of a non-severe system error. Subsequent SQL statements can be processed. (Reason "Cannot find database prototype.")	This is an internal error. Contact IBM Software support.
SQL0901N	The SQL statement failed because of a non-severe system error. Subsequent SQL statements can be processed. (Reason "No data to unpack.")	This is an internal error. Contact IBM Software support.
SQL0901N	The SQL statement failed because of a non-severe system error. Subsequent SQL statements can be processed. (Reason "Error creating wrapper object.")	This is an internal error. Contact IBM Software support.

Table 59. Messages issued by the wrapper for Entrez (continued)

Error Code	Message	Explanation
SQL0901N	The SQL statement failed because of a non-severe system error. Subsequent SQL statements can be processed. (Reason "Bad expression type.")	This is an internal error. Contact IBM Software support.
SQL0901N	The SQL statement failed because of a non-severe system error. Subsequent SQL statements can be processed. (Reason "Cannot find nickname.")	This is an internal error. Contact IBM Software support.
SQL0901N	The SQL statement failed because of a non-severe system error. Subsequent SQL statements can be processed. (Reason "Memory allocation error.")	There is not sufficient memory to process the allocation request inside of the wrapper.
SQL1816N	Wrapper "<wrapper_name>" cannot be used to access the "version" of data source ("<server_type>", "<server_version>") that you are trying to define to the federated server.	A value in the VERSION clause of the CREATE SERVER statement is not valid.
SQL1816N	Wrapper "<wrapper_name>" cannot be used to access the "type" of data source ("<server_type>", "<server_version>") that you are trying to define to the federated server.	A value in the TYPE clause of the CREATE SERVER statement is not valid.
SQL1817N	The CREATE SERVER statement does not identify the "type" of the data source that you want to define to the federated database.	The TYPE clause of the CREATE SERVER statement is required but was not specified.
SQL1822N	Unexpected error code "900" received from data source "Entrez Wrapper." Associated text and tokens are "Parent nickname not defined."	This is an internal error. Contact IBM Software support.
SQL1823N	No data type mapping exists for data type "<data_type>" from server "<server_name>."	This is an internal error. Contact IBM Software support.
SQL1881N	"<option_name>" is not a valid "<option_type>" for "<option_name>."	The specified option is not a valid option. Check the CREATE NICKNAME statement.
SQL1882N	The "<option_type>" option "<option_name>" cannot be set to "<option_value>" for "<option_name>."	The specified value is not valid for this option. Check the CREATE NICKNAME statement.

Table 59. Messages issued by the wrapper for Entrez (continued)

Error Code	Message	Explanation
SQL1883N	"<option_name>" is a required "<option_type>" option for "<option_name>."	The specified option is required for the object but was not specified. Check the CREATE NICKNAME statement.
SQL1884N	You specified "FOREIGN_KEY" (a "COLUMN" option) more than once.	This is an internal error. Contact IBM Software support.
SQL1884N	You specified "PRIMARY_KEY" (a "COLUMN" option) more than once.	This is an internal error. Contact IBM Software support.
SQL30090N	Operation invalid for application execution environment. Reason code = "Cannot change server version".	The version of a server cannot be changed by issuing the ALTER SERVER statement. A new server must be created with the new version.
SQL30090N	Operation invalid for application execution environment. Reason code = "Invalid PARENT nickname".	The referenced nickname in a PARENT nickname option is not valid for the current nickname.
SQL30090N	Operation invalid for application execution environment. Reason code = "Invalid column name".	A specified column name in the CREATE NICKNAME statement does not match any of the possible columns for the nickname.
SQL30090N	Operation invalid for application execution environment. Reason code = "Cannot AND fetch keys".	Multiple references to a fetch key, such as the PMID column of the PMArticles nickname, were made in a conjunction. For example, "PMID = 12346 AND PMID = 12348". Fetch key predicates can be associated only using OR.
SQL30090N	Operation invalid for application execution environment. Reason code = "Mixed SEARCH_TERM and CONTAINS functions".	The SEARCH_TERM and CONTAINS functions cannot be mixed in a query. Only one SEARCH_TERM function is allowed per query.
SQL30090N	Operation invalid for application execution environment. Reason code = "Invalid first argument in function".	The first argument to a SEARCH_TERM or CONTAINS function was not valid. This argument must be a reference to a column.

Table 59. Messages issued by the wrapper for Entrez (continued)

Error Code	Message	Explanation
SQL30090N	Operation invalid for application execution environment. Reason code = "Invalid second argument in function".	The second argument to a SEARCH_TERM or CONTAINS function was not valid. This argument must be a string literal, a host variable, or a column reference.
SQL30090N	Operation invalid for application execution environment. Reason code = "Untagged column in CONTAINS function".	The first argument to the CONTAINS function was not valid. This argument must be a reference to a tagged column.
SQL30090N	Operation invalid for application execution environment. Reason code = "Invalid function".	This is an internal error. Contact IBM Software support.

Related concepts:

- "Introduction to messages" in the *Message Reference Volume 1*

Related reference:

- "SQLSTATE messages" in the *Message Reference Volume 2*

Chapter 12. Configuring access to Excel data sources

This chapter explains how to configure your federated server to access data that is stored in Excel data sources. You can configure access to Excel data sources by using the DB2 Control Center or by issuing SQL statements.

This chapter:
- Explains what Excel is
- Lists the tasks that you need to perform
- Contains examples of the SQL statements that you need
- Lists the error messages associated with the Excel wrapper

What is Excel?

An Excel spreadsheet or workbook is a file created using the Microsoft® Excel application and has a file extension of xls. DB2® Information Integrator supports spreadsheets from Excel 97, Excel 2000, and Excel 2002. Figure 23 illustrates how the Excel wrapper connects your spreadsheets to your federated system.

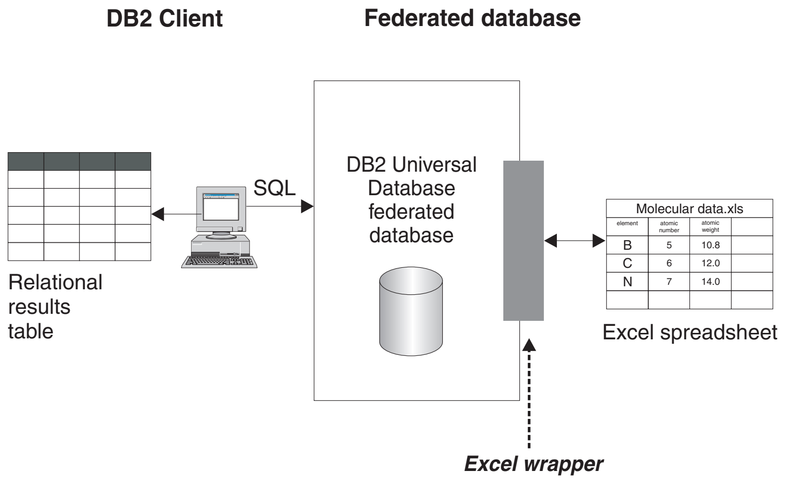


Figure 23. How the Excel wrapper works

The Excel wrapper uses the CREATE NICKNAME statement to map the columns in your Excel spreadsheet to columns in your DB2 UDB federated system. Table 60 shows sample spreadsheet data that is stored in a file called `Compound_Master.xls`.

Table 60. Sample spreadsheet for Compound_Master.xls

	A	B	C	D
1	COMPOUND_NAME	WEIGHT	MOL_COUNT	WAS_TESTED
2	compound_A	1.23	367	tested
3	compound_G		210	
4	compound_F	0.000425536	174	tested
5	compound_Y	1.00256		tested
6	compound_Q		1024	

Table 60. Sample spreadsheet for Compound_Master.xls (continued)

	A	B	C	D
7	compound_B	33.5362		
8	compound_S	0.96723	67	tested
9	compound_O	1.2		tested

This information is usually not available to you through standard SQL commands. When the Excel wrapper is installed and registered, you can access this information as if it were a standard relational data source. For example, if you wanted to know all the compound data where the molecular count is greater than 100, you would run the following SQL query:

```
SELECT * FROM compound_master WHERE mol_count > 100
```

The results of the query are shown in Table 61.

Table 61. Query results

COMPOUND_NAME	WEIGHT	MOL_COUNT	WAS_TESTED
compound_A	1.23	367	tested
compound_G		210	
compound_F	0.000425536	174	tested
compound_Q		1024	

Related concepts:
- "Methods of accessing Excel data" on page 25

Related tasks:
- "Adding Excel data sources to a federated server" on page 218

Adding Excel to a federated server

Adding Excel data sources to a federated server

To configure the federated server to access Excel data sources, you must provide the federated server with information about the data sources and objects that you want to access.

| You can configure the federated server to access Excel data sources by using the
| DB2 Control Center or the DB2 command line. The DB2 Control Center includes a
| wizard to guide you through the steps required to configure the federated server.

Prerequisites:
- DB2 Information Integrator must be installed on a server that will act as the federated server
- A federated database must exist on the federated server
- Excel worksheets that are structured properly so that the wrapper can access the data

Procedure:

To add the Excel data sources to a federated server:

1. Register the wrapper.
2. Register the server definition.
3. Register nicknames for the Excel worksheets.

Related concepts:

- "DB2 Information Integrator installation process - overview" in the *IBM DB2 Information Integrator Installation Guide for Linux, UNIX, and Windows*
- "Methods of accessing Excel data" on page 25
- "What is Excel?" on page 217
- "Fast track to configuring your data sources" on page 55

Related tasks:

- "Creating a federated database" on page 51
- "Registering the Excel wrapper" on page 219
- "Registering the server for an Excel data source" on page 220
- "Registering nicknames for Excel data sources" on page 221
- "Checking the setup of the federated server" on page 37

Related reference:

- "Supported operating systems for DB2 Information Integrator (32-bit)" in the *IBM DB2 Information Integrator Installation Guide for Linux, UNIX, and Windows*
- "Supported operating systems for DB2 Information Integrator (64-bit)" in the *IBM DB2 Information Integrator Installation Guide for Linux, UNIX, and Windows*
- "Choose the correct wrapper" on page 24

Registering the Excel wrapper

Registering the Excel wrapper is part of the larger task of adding Excel data sources to a federated server.

You must register a wrapper to access Excel data sources. Wrappers are used by federated servers to communicate with and retrieve data from data sources. Wrappers are implemented as a set of library files.

Restrictions:

- The Excel wrapper is available only for Microsoft Windows operating systems that support DB2 Universal Database Enterprise Server Edition.
- The Excel application must be installed on the server where DB2 Information Integrator is installed before the Excel wrapper can be used.
- Pass-through sessions are not allowed.

Procedure:

To register a wrapper, issue the CREATE WRAPPER statement with the name of the wrapper and the name of the wrapper library file.

| For example, to register a wrapper with the name `excel_wrapper`, issue the
| following statement:

```
CREATE WRAPPER excel_wrapper LIBRARY 'db2lsxls.dll';
```

You must specify the wrapper library file `db2lsxls.dll` in the CREATE WRAPPER statement.

The next task in this sequence of tasks is registering the server definition for the Excel wrapper.

Related tasks:
- "Registering the server for an Excel data source" on page 220

Related reference:
- "CREATE WRAPPER statement" in the *SQL Reference, Volume 2*

Excel wrapper library files

The following table lists the directory paths and library file names for the Excel wrapper.

When you install DB2 Information Integrator, this library file is added to the directory path listed in the table.

When you register a wrapper, specify only the library file name that is listed in the table.

Table 62. Excel wrapper library location and file name

Operating system	Directory path	Wrapper library file
Windows	%DB2PATH%\bin	db2lsxls.dll

%DB2PATH% is the environment variable that is used to specify the directory path where DB2 Information Integrator is installed on Windows. The default Windows directory path is `C:\Program Files\IBM\SQLLIB`.

Related tasks:
- "Registering the Excel wrapper" on page 219

Registering the server for an Excel data source

Registering the server for an Excel data source is part of the larger task of adding Excel to a federated system. After the wrapper is registered, you must register a corresponding server.

For Excel, a server definition is created because the hierarchy of federated objects requires that data source files (identified by nicknames) are associated with a specific server object.

Procedure:

To register the Excel server to the federated system, use the CREATE SERVER statement.

Suppose that you want to create a server object called `biochem_lab` for a workbook that contains biochemical data. The server object must be associated with the Excel wrapper that you registered using the CREATE WRAPPER statement. The CREATE SERVER statement to register this server object is:

```
CREATE SERVER biochem_lab WRAPPER Excel_2000_Wrapper;
```

The next task in this sequence of tasks is registering nicknames for Excel data sources.

Related tasks:
- "Registering nicknames for Excel data sources" on page 221

Related reference:
- "CREATE SERVER statement" in the *SQL Reference, Volume 2*
- "CREATE SERVER statement arguments - Excel wrapper" on page 544

Registering nicknames for Excel data sources

Registering nicknames for Excel data sources is part of the larger task of adding Excel to a federated system. After you register a server, you must register a corresponding nickname. Nicknames are used when you refer to an Excel data source in a query.

Procedure:

To map the Excel data source to relational tables, create a nickname using the CREATE NICKNAME statement.

The statement in the following example creates the Compounds nickname from the Excel spreadsheet file named `CompoundMaster.xls`. The file contains three columns of data that are being defined to the federated system as `Compound_ID`, `CompoundName`, and `MolWeight`.

```
CREATE NICKNAME Compounds (
         Compound_ID INTEGER,
         CompoundName VARCHAR(50),
         MolWeight FLOAT)
     FOR SERVER biochem_lab
     OPTIONS (FILE_PATH 'C:\My Documents\CompoundMaster.xls',
         RANGE 'B2:D5');
```

There are no further tasks in this sequence of tasks.

Related tasks:
- "Adding Excel data sources to a federated server" on page 218
- "Specifying nickname columns for a nonrelational data source" on page 65

Related reference:
- "Excel data source – Example queries" on page 221
- "CREATE NICKNAME statement syntax - Excel wrapper" on page 544

Excel data source – Example queries

This topic lists several sample Excel spreadsheet queries using the example nickname `Compounds`.

To run queries, you use the nickname and the defined nickname columns in your SQL statements in the same manner as you would use a regular table name and table columns.

The following query displays all compound_ID's where the molecular weight is greater than 2000:

```
SELECT compound_ID
FROM Compounds
WHERE MolWeight > 200;
```

The following query displays all records where the compound name or molecular weight is null:

```
SELECT *
FROM Compounds
WHERE CompoundName IS NULL
OR MolWeight IS NULL;
```

The following query displays all records where the compound name contains the string ase and the molecular weight is greater than or equal to 300:

```
SELECT *
FROM Compounds
WHERE CompoundName LIKE '%ase%
AND MolWeight >=300;
```

Related reference:

- "Documentum data source – Example queries" on page 187
- "Excel data source – Sample scenario" on page 222

Excel data source – Sample scenario

This section demonstrates a sample implementation of the Excel_2000 wrapper accessing an Excel 2000 worksheet located in the C:\Data directory. The scenario registers the wrapper, registers a server and registers one nickname, that will be used to access the worksheet. The statements shown in the scenario are entered using the DB2 command line. After the wrapper is registered, you can run queries on the worksheet.

The scenario starts with a compound worksheet, called Compund_Master.xls, with 4 columns and 9 rows. The fully-qualified path name to the file is C:\Data\Compound_Master.xls. The contents are show in Table 63.

Table 63. Sample worksheet Compound_Master.xls

	A	B	C	D
1	COMPOUND_NAME	WEIGHT	MOL_COUNT	WAS_TESTED
2	compound_A	1.23	367	tested
3	compound_G		210	
4	compound_F	0.000425536	174	tested
5	compound_Y	1.00256		tested
6	compound_Q		1024	
7	compound_B	33.5362		
8	compound_S	0.96723	67	tested
9	compound_O	1.2		tested

Procedure:

To access an Excel 2000 worksheet using the Excel wrapper:

1. Register the Excel_2000 wrapper:

   ```
   db2 => CREATE WRAPPER Excel_2000 LIBRARY 'db2lsxls.dll'
   ```
2. Register the server:

   ```
   db2 => CREATE SERVER biochem_lab WRAPPER Excel_2000
   ```
3. Register a nickname that refers to the Excel worksheet:

   ```
   db2 => CREATE NICKNAME Compound_Master (compound_name VARCHAR(40),
       weight FLOAT, mol_count INTEGER, was_tested VARCHAR(20))
   FOR biochem_lab
   OPTIONS ( FILE_PATH 'C:\Data\Compound_Master.xls')
   ```

The registration process is complete. The Excel data source is now part of the federated system, and can be used in SQL queries.

The following examples show sample SQL queries and results obtained using the Excel data source.

- Sample SQL query: "Give me all the compound data where `mol_count` is greater than 100."

  ```
  SELECT * FROM compound_master WHERE mol_count > 100
  ```
  Result: All fields for rows 2, 3, 4, 6, and 8.
- Sample SQL query: "Give me the `compound_name` and `mol_count` for all compounds where the `mol_count` has not yet been determined."

  ```
  SELECT compound_name, mol_count FROM compound_master
  WHERE mol_count IS NULL
  ```
  Result: Fields `compound_name` and `mol_count` of rows 5, 7, and 10 from the worksheet.
- Sample SQL query: "Count the number of compounds that have not been tested and the weight is greater than 1."

  ```
  SELECT count(*) FROM compound_master
  WHERE was_tested IS NULL AND weight > 1
  ```
  Result: The record count of 1 which represents the single row 7 from the worksheet that meets the criteria.
- Sample SQL query: "Give me the `compound_name` and `mol_count` for all compounds where the `mol_count` has been determined and is less than the average `mol_count`."

  ```
  SELECT compound_name, mol_count
  FROM compound_master
  WHERE mol_count IS NOT NULL
  AND mol_count < (SELECT AVG(mol_count) FROM compound_master
                  WHERE mol_count IS NOT NULL AND was_tested IS NOT NULL)
  ```
  The sub-query returns the average 368 to the main query which then returns Table 64:

Table 64. Query results

COMPOUND_NAME	MOL_COUNT
compound_A	367
compound_G	210
compound_F	174
compound_S	67

Related tasks:

- "Adding Excel data sources to a federated server" on page 218

Related reference:

- "Excel data source – Example queries" on page 221

File access control model for the Excel wrapper

The database management system accesses Excel files with the authority of the LOG ON AS property of the DB2 database service. This setting can be viewed in the LOG ON properties page for the DB2 instance. The properties page is accessed through the Windows NT Services control panel.

Related reference:

- "File access control model for the table-structured file wrapper" on page 361
- "Access control for the Documentum wrapper" on page 188

Messages for the Excel wrapper

This section lists and describes messages you might encounter while working with the wrapper for Excel.

Table 65. Messages issued by the wrapper for Excel

Error Code	Message	Explanation
SQL1817N	The CREATE SERVER statement does not identify the "VERSION" of data source that you want defined to the federated database.	The VERSION parameter was not specified during the CREATE SERVER statement. Correct the SQL statement and run it again.
SQL1822N	Unexpected error code "-1000.<internal program code>" received from data source "Excel Wrapper". Associated text and tokens are "Memory allocation error"	Contact IBM Software Support.
SQL1822N	Unexpected error code "-1001.<internal program code>" received from data source "Excel Wrapper". Associated text and tokens are "Unknown option".	The option specified in the DDL statement is not supported. Correct the SQL statement and run it again.
SQL1822N	Unexpected error code "-1002.<internal program code>" received from data source "Excel Wrapper". Associated text and tokens are "Creation of DELTA object failed".	An internal program error has occurred. Contact IBM Software Support.
SQL1822N	Unexpected error code "-1100.<internal program code>" received from data source "Excel Wrapper". Associated text and tokens are "Wrapper options are not supported"	Wrapper OPTIONS are not supported by this wrapper. Correct the SQL statement and run it again.
SQL1822N	Unexpected error code "-1200.<internal program code>" received from data source "Excel Wrapper". Associated text and tokens are "<option> is an unsupported Server option".	The specified option is not supported by this wrapper. Correct the SQL statement and run it again.

Table 65. Messages issued by the wrapper for Excel (continued)

Error Code	Message	Explanation
SQL1822N	Unexpected error code "-1201.<internal program code>" received from data source "Excel Wrapper". Associated text and tokens are "Error obtaining server name"	An internal program error has occurred. Contact IBM Software Support.
SQL1822N	Unexpected error code "-1209. <internal program code>" received from data source "Excel Wrapper". Associated text and tokens are "Error converting VARCHAR data"	An internal program error has occurred. Contact IBM Software Support.
SQL1822N	Unexpected error code "-1211.<internal program code>" received from data source "Excel Wrapper". Associated text and tokens are "Error converting INTEGER data"	An internal program error has occurred. Contact IBM Software Support.
SQL1822N	Unexpected error code "-1212.<internal program code>" received from data source "Excel Wrapper". Associated text and tokens are "Error converting FLOAT data"	An internal program error has occurred. Contact IBM Software Support.
SQL1822N	Unexpected error code "-1400.<internal program code>" received from data source "Excel Wrapper". Associated text and tokens are "<option> is an unsupported User option"	The specified option is not supported by this wrapper. Correct the SQL statement and run it again.
SQL1822N	Unexpected error code "-1401.<internal program code>" received from data source "Excel Wrapper". Associated text and tokens are "Creation of USER Delta object failed"	An internal program error has occurred. Contact IBM Software Support.
SQL1822N	Unexpected error code "-1500.<internal program code>" received from data source "Excel Wrapper". Associated text and tokens are "<option> is an unsupported Nickname option"	The specified option is not supported by this wrapper. Correct the SQL statement and run it again.
SQL1822N	Unexpected error code "-1501.<internal program code>" received from data source "Excel Wrapper". Associated text and tokens are "Required option PATH not specified"	The PATH option is required to register the NICKNAME. Correct the SQL statement and run it again.
SQL1822N	Unexpected error code "-1502.<internal program code>" received from data source "Excel Wrapper". Associated text and tokens are "Creation of NICKNAME Delta object failed"	An internal program error has occurred. Contact IBM Software Support.

Table 65. Messages issued by the wrapper for Excel (continued)

Error Code	Message	Explanation
SQL1822N	Unexpected error code "-1503.<internal program code>" received from data source "Excel Wrapper". Associated text and tokens are "Error obtaining Nickname column type"	An internal program error has occurred. Contact IBM Software Support.
SQL1822N	Unexpected error code "-1504.<internal program code>" received from data source "Excel Wrapper". Associated text and tokens are "Error obtaining Nickname column type name"	An internal program error has occurred. Contact IBM Software Support.
SQL1822N	Unexpected error code "-1505.<internal program code>" received from data source "Excel Wrapper". Associated text and tokens are received from data source "Excel Wrapper".	The specified <data type> is not supported by this wrapper. Correct the SQL statement and run it again.
SQL1822N	Unexpected error code "-1506.<internal program code>" received from data source "Excel Wrapper". Associated text and tokens are "Error obtaining Nickname column info"	An internal program error has occurred. Contact IBM Software Support.
SQL1822N	Unexpected error code "-1507.<internal program code>" received from data source "Excel Wrapper". Associated text and tokens are "<option> option cannot be dropped"	The specified option cannot be dropped because it is a required option.
SQL1822N	Unexpected error code "-1508.VANI" received from data source "Excel Wrapper". Associated text and tokens are "Column names cannot be altered"	The altering of column names is not permitted by the Excel wrapper.
SQL1822N	Unexpected error code. "-1509.VCTS" received from data source "Excel Wrapper". Associated text and tokens are "No column info found".	The column information is not found.
SQL1822N	Unexpected error code "-1701.<internal program code>" received from data source "Excel Wrapper". Associated text and tokens are "Error parsing SQL"	An internal program error has occurred. Contact IBM Software Support.
SQL1822N	Unexpected error code "-1702.<internal program code>" received from data source "Excel Wrapper". Associated text and tokens are "Error accessing NICKNAME object"	An internal program error has occurred. Contact IBM Software Support.

Table 65. Messages issued by the wrapper for Excel (continued)

Error Code	**Message**	**Explanation**
SQL1822N	Unexpected error code "-1703.<internal program code>" received from data source "Excel Wrapper". Associated text and tokens are "Error building data storage area"	An internal program error has occurred. Contact IBM Software Support.
SQL1822N	Unexpected error code "-1704.<internal program code>" received from data source "Excel Wrapper". Associated text and tokens are "Error linking SQL to Nickname Data"	An internal program error has occurred. Contact IBM Software Support.
SQL1822N	Unexpected error code "-1705.<internal program code>" received from data source "Excel Wrapper". Associated text and tokens are "Excel application startup failed"	The startup of the Excel application failed. Confirm that Excel is installed on the system and has been registered with the correct version of the wrapper. Check the LOG ON AS property for the DB2 instance in the Windows NT Services control panel. The Excel application will be accessed using this authority. Confirm that this user has appropriate rights or change this property to an authorized account, then restart DB2 and run the SQL query again.
SQL1822N	Unexpected error code "-1706.<internal program code>" received from data source "Excel Wrapper". Associated text and tokens are "Error opening source spreadsheet"	A problem occurred while opening the spreadsheet referenced by the nickname in the SQL query. Ensure that the file exists in the PATH specified during the CREATE NICKNAME statement during registration.
SQL1822N	Unexpected error code "-1707.<internal program code>" received from data source "Excel Wrapper". Associated text and tokens are "Error accessing DL output storage area"	An internal program error occurred. Contact IBM Software Support.
SQL1822N	Unexpected error code "-1708.<internal program code>" received from data source "Excel Wrapper". Associated text and tokens are "Excel application end failed"	An internal program error occurred. If this error persists after repeated queries, contact IBM Software Support.
SQL1822N	Unexpected error code "-1711.<internal program code>" received from data source "Excel Wrapper". Associated text and tokens are "Error during fetch, possible data/col type mismatch"	The data fetched during the SQL query was of a different data type than the data type specified during the registration of the nickname. Correct the data in the source spreadsheet or correct the registered data type in the nickname. If this does not correct the problem, contact IBM Software Support.

Table 65. Messages issued by the wrapper for Excel (continued)

Error Code	Message	Explanation
SQL1822N	Unexpected error code "-1900.<internal program code>" received from data source "Excel Wrapper". Associated text and tokens are "Memory allocation error"	An internal program error has occurred. Contact IBM Software Support.

Related concepts:

- "Introduction to messages" in the *Message Reference Volume 1*

Related reference:

- "SQLSTATE messages" in the *Message Reference Volume 2*

Chapter 13. Configuring access to Extended Search data sources

This chapter explains how to configure your federated server to access data that is stored in IBM Lotus Extended Search data sources. You can configure access to Extended Search data sources by using the DB2 Control Center or by issuing SQL statements.

This chapter:
- Explains what Extended Search is
- Lists the tasks that you need to perform
- Contains examples of the SQL statements that you need
- Lists the error messages associated with the Extended Search wrapper

What is Extended Search?

The Extended Search product is a multi-tiered client/server system that provides extensive search and retrieval capabilities. With Extended Search, you can enter a single request and search potentially thousands of data repositories and the Internet at the same time. These repositories, which can be of varied content and structure, might be geographically dispersed throughout the world.

Extended Search supports distributed, heterogeneous searching of structured and unstructured data through a single point of access. It leverages your current data management investment and completely handles the logistics required to access many diverse sources simultaneously.

Extended Search uses its generalized query language (GQL) as a common search syntax and internally translates each search request into the native search languages of the data sources that you want to search. It also uses methods that are native to those sources to find and retrieve information without regard for where a source is located.

See the Extended Search product documentation for information about installing an Extended Search server, configuring the search domain, and using GQL. The following documents are available on the Resources page of the IBM® Lotus® Extended Search Web site:

`http://www.lotus.com/products/des.nsf/wdocuments/resources`

Extended Search General Information
: Describes the components in an Extended Search system and how they interact with each other and the backend data systems.

Extended Search Installation
: Defines the system prerequisites and provides instructions for installing the product and verifying the installation process.

Extended Search Administration
: Provides instructions for adding data sources to the search domain, configuring searchable fields, and using sample search applications to query Extended Search sources.

Extended Search Programming
Discusses the application development tools that you can use to extend search support to data sources that are not supported in the default configuration of the product. Includes a description of the Extended Search generalized query language.

Extended Search data sources

With Extended Search, you can search the following types of data sources:

- Many popular Web search sites and news sites. If you need to search your intranet's search site, or other internal or external search sites, you can easily add support for doing so.
- Mail systems, such as those that you manage with Lotus Notes® and Microsoft® Exchange Server.
- Document management systems, such as DB2® Information Integrator for Content databases.
- Relational databases, such as IBM DB2, Oracle, Microsoft SQL Server, Microsoft Access, and other databases that comply with Open Database Connectivity (ODBC) standards.
- Full text indexes, such as those that you create with IBM WebSphere® Portal, Domino™ Domain Index, Microsoft Index Server, and Microsoft Site Server.
- Lotus repositories, including Notes databases, Domino.Doc® libraries and cabinets, Lotus QuickPlace places, and Lotus Discovery Server™ knowledge maps (K-maps).
- Instant messaging systems, such as Lotus Sametime®. This feature enables you to direct queries to knowledgeable persons, not just searchable data repositories.
- Lightweight Directory Access Protocol (LDAP) directories, such as those that you manage with IBM SecureWay®, Domino LDAP Server, and Exchange LDAP Server.
- File systems. You can search text files that are stored locally or on network drives. You cannot search compressed or encrypted files.

With the Extended Search C++ and Java™ application programming interfaces (APIs), you can extend support to other types of sources, such as proprietary databases that are not mentioned here.

How the Extended Search wrapper works

In a structured relational database model, columns are named and represented in a consistent format. This feature allows you to perform precise computational operations and join data from different tables by comparing specific column values. You can also do other types of analysis, such as listing objects in one table that are missing from another table.

In contrast, unstructured data is often stored in a free text form. Typically, there is little or no metadata that enables you to query for information by column name. A search of unstructured data depends more on finding data that matches user-specified keywords than on computational criteria.

The Extended Search wrapper combines these two search techniques. With the wrapper, you can use structured query language to search unstructured content in an Extended Search domain. You can then perform analytical or relational operations on the search results.

You issue queries by entering SQL statements that refer to a special purpose DB2 table (a nickname table). Extended Search performs the search according to the SQL criteria and populates the nickname table with the result data. Because the search results persist in a table, the data is available for operations with other database tables, including other nickname tables.

When you submit a search request with the wrapper, you can retrieve data from any Extended Search source that is mapped to a nickname table. You can integrate this data with other data sources in your federated system without moving the data out of the native data source. Search results appear as a single result set regardless of how many sources provide responses to the query.

The following figure shows how the Extended Search wrapper connects the diverse data sources in an Extended Search domain to a federated database system. The wrapper accesses and retrieves data from one or more remote Extended Search servers. If the wrapper contacts an Extended Search server that is connected to other Extended Search servers, search results can be returned from multiple servers.

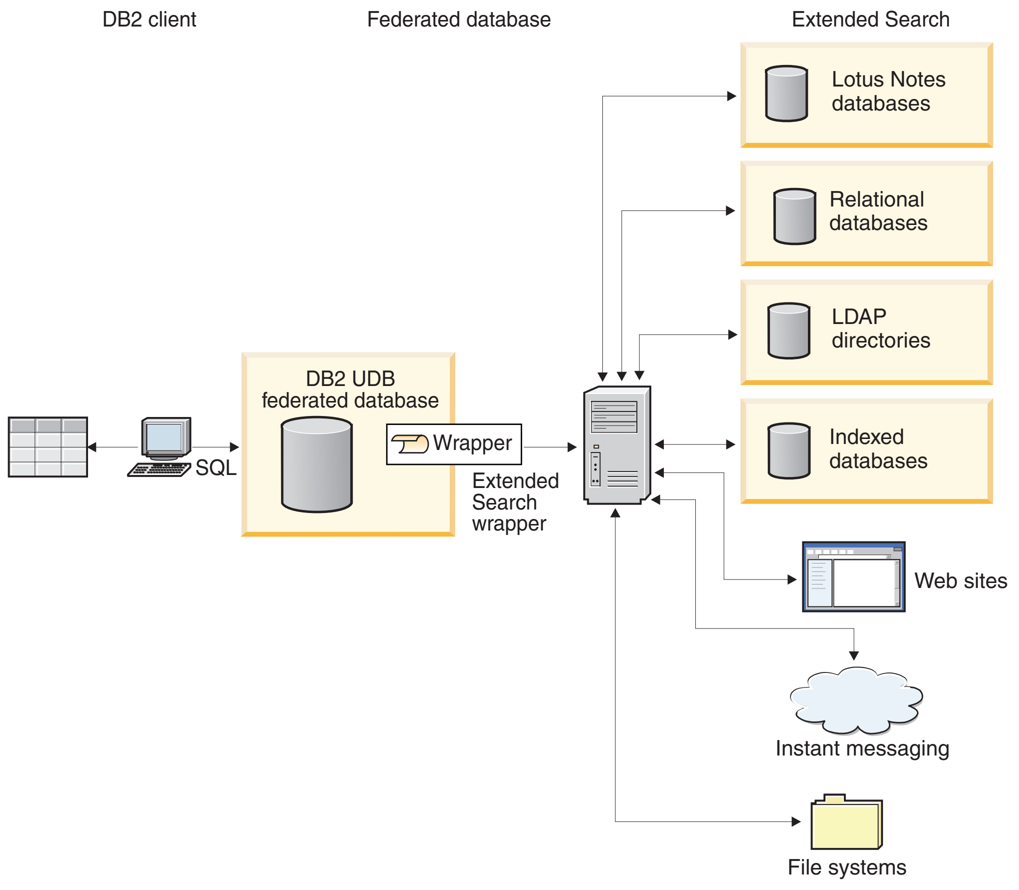


Figure 24. How the Extended Search wrapper works

Related tasks:

- “Adding Extended Search data sources to a federated server” on page 235

Extended Search nicknames

In the Extended Search data model, one or more fields constitute a document. A collection of documents constitutes a data source. You can combine any number of data sources into a category, which enables you to search them and administer them as a group.

To ensure that users access only the data sources for which they have a need, a category must belong to at least one application. Think of applications as a way of grouping users for purposes of controlling access and search capabilities. For example, a personnel application might include the same data sources as a financial application, but the users of each application would not necessarily need access to the same fields in those data sources.

When you register nicknames, you identify the applications, categories, data sources, and data source fields that you want to search. These entities must exist in the Extended Search configuration database. To search an Extended Search data source with the Extended Search wrapper, you must create a nickname for the source.

The contents of the nickname table reflect the state of the Extended Search configuration database at the time that you register the nickname. If an Extended Search administrator updates the configuration (for example, by adding or deleting sources or fields), those changes are not reflected in the nickname table. If a nickname table refers to changed data, and you want to stay current with the Extended Search configuration database, you must alter the nickname or drop it and create a new nickname.

If you do not alter or recreate the nickname, you might receive errors and reports of zero results when you attempt to search items that no longer exist in the Extended Search domain.

Although a single nickname table can contain information about all the sources that are configured in Extended Search, creating several nickname tables might be more useful. To use the full power of DB2®, create a separate nickname for each type of data source that you plan to search with the Extended Search wrapper.

For example, you might have one nickname for Web sources, one for Notes databases, one for file systems, and so on. By having separate nickname tables, you are better able to perform joins on the data that is returned to the wrapper, relate diverse sources based on field values, and integrate the result data with other data in your federated system.

Related concepts:

- "Extended Search vertical tables" on page 233

Related tasks:

- "Registering nicknames for Extended Search data sources" on page 239

Related reference:

- "Extended Search wrapper - Example queries" on page 242
- "CREATE NICKNAME statement syntax - Extended Search wrapper" on page 548

Extended Search vertical tables

An Extended Search application can consist of many categories which, in turn, can contain many data sources. Because each data source uses its own conventions for field names, an intersection of fields might result in an empty set. When you map data source fields to user-defined columns in nickname tables, and present search results as a horizontal table, the table might contain an unmanageable number of columns. If many rows contain only a few columns with data, the table will appear sparsely populated. For example:

Column_1	Column_2	Column_3
Value_11		
	Value_22	
Value_31		Value_33

Within Extended Search, you can control the presentation of results by defining mapped fields. Mapped fields provide a way for you to combine content that has a common purpose but that is named differently in different sources. For example, you might create a mapped field named EmployeeNumber to represent result data from fields that are named EmpNum, EmpNo, and EmpID in various sources. Without this mapping feature, you would need to define a nickname column for each unique field name as opposed to a single column for the mapped field.

Mapping fields is useful when you know the names of the fields that you need to relate. Some applications, however, need to relate a large number of fields from many data sources. The relationships between the fields, particularly for unstructured data, might not be known ahead of time. Thus, it becomes difficult to define and structure meaningful nickname tables. To support this type of application, the Extended Search wrapper allows you to create a vertical nickname table.

When you create a nickname table for Extended Search, you can enable the VERTICAL_TABLE option. This option returns all the fields that are configured to be returnable in a data source, as defined in the Extended Search configuration database. Use this option when you are not sure which columns will be relevant in your search or which columns will be relevant when you perform post-processing queries or joins on the result sets.

Each row in the vertical table contains information about a field that was returned in the result set. For each row, Extended Search returns the name of the source that the field came from, the field name, its value, and its data type (date, integer, and so on). Unlike results that are scattered across columns in a horizontal table, the vertical table is densely populated and contains many rows of data. For example:

Field_Name	Field_Value	Field_Datatype
Column_1	Value_11	VARCHAR
Column_2	Value_22	DATE
Column_1	Value_31	VARCHAR
Column_3	Value_33	VARCHAR

You can perform SQL operations on this data when you query the table, and you can query all column labels. For example:

```
Field_Value LIKE '%IBM%'
```

Because the VERTICAL_TABLE option returns information about all returnable fields in a data source, you might not need to query specific user-defined columns. If you enable this option and then issue a SELECT statement to search user-defined columns, you might receive duplicate information in the search results. However, if you define user-defined columns, you can use those columns in joins with other tables in your federated system.

The following two tables summarizes the system-provided columns that Extended Search returns for each row in a vertical nickname table.

The wrapper always returns the following three fixed columns for each nickname.

Column Name	Data Type	Description
DOC_ID	VARCHAR(512)	The document identifier, unique to each item in a set of search results.
DOC_RANK	INTEGER	The relevance ranking of the document.
CLIENT_LOCALE	VARCHAR(5)	The client locale of the search request. If the SQL query does not provide the client locale, the query will use enUS as the default client locale.

The wrapper creates the following fixed columns only if the VERTICAL_TABLE option is enabled.

Column Name	Data Type	Description
DATASOURCE_NAME	VARCHAR(128)	The name of the data source that produced the search result.
FIELD_NAME	VARCHAR(128)	The name of a field that was returned in the search result.
FIELD_VALUE	VARCHAR(4096)	The value of a field that was returned in a result set. If the field value is longer than the maximum length of the nickname column (the VARCHAR value), the field value is truncated. The token ES_TRUNCATE at the end of the column indicates that the value is incomplete.
FIELD_DATATYPE	SMALLINT	An integer value that represents the actual data type of the field value: 384 DATE 448 VARCHAR 496 INTEGER

A vertical table, which stores result data as VARCHAR values, can be difficult to query. For more precise searching, create mapped fields in the Extended Search configuration database and then define them in the nickname table. With mapped fields, you can create a concise horizontal table of search results. You also optimize your ability to perform relational operations on the results and combine them in queries that involve other tables in your federated database system.

For information about defining mapped fields in Extended Search, see *Extended Search Administration*, which is available on the Resources page of the IBM® Lotus® Extended Search Web site:

`http://www.lotus.com/products/des.nsf/wdocuments/resources`

Related concepts:
- "Extended Search nicknames" on page 232

Related tasks:
- "Registering nicknames for Extended Search data sources" on page 239

Related reference:
- "Extended Search wrapper - Example queries" on page 242
- "CREATE NICKNAME statement syntax - Extended Search wrapper" on page 548

Adding Extended Search to a federated server

Adding Extended Search data sources to a federated server

To configure the federated server to access Extended Search data sources, you must provide the federated server with information about the data sources and objects that you want to access.

| You can configure the federated server to access Extended Search data sources by
| using the DB2 Control Center or the DB2 command line. The DB2 Control Center
| includes a wizard to guide you through the steps required to configure the
| federated server.

Prerequisites:
- DB2 Information Integrator must be installed on a server that will act as the federated server
- A federated database must exist on the federated server
- Before you use the Extended Search wrapper, ensure that the sources that you plan to search are configured in the Extended Search configuration database. Submit a few queries through the Extended Search client to verify your ability to search the sources before you attempt to search the sources with the Extended Search wrapper.

Procedure:

To add Extended Search data sources to a federated server:
1. Register the wrapper.
2. Register the server definition.
3. Optional: Create the user mappings.
4. Register nicknames for the Extended Search data sources.
5. Optional: Register custom function for the Extended Search wrapper.

Related concepts:
- "DB2 Information Integrator installation process - overview" in the *IBM DB2 Information Integrator Installation Guide for Linux, UNIX, and Windows*

Related tasks:
- "Creating a federated database" on page 51
- "Registering the Extended Search wrapper" on page 236
- "Registering the server for Extended Search data sources" on page 237
- "Registering nicknames for Extended Search data sources" on page 239
- "Registering user mappings for Extended Search data sources" on page 238
- "Registering the custom functions for the Extended Search wrapper" on page 240

Related reference:
- "Supported operating systems for DB2 Information Integrator (32-bit)" in the *IBM DB2 Information Integrator Installation Guide for Linux, UNIX, and Windows*
- "Supported operating systems for DB2 Information Integrator (64-bit)" in the *IBM DB2 Information Integrator Installation Guide for Linux, UNIX, and Windows*

Registering the Extended Search wrapper

Registering the Extended Search wrapper is part of the larger task of adding Extended Search data sources to a federated server.

You must register a wrapper to access Extended Search data sources. Wrappers are used by federated servers to communicate with and retrieve data from data sources. Wrappers are implemented as a set of library files.

Procedure:

To register a wrapper, issue the CREATE WRAPPER statement with the name of the wrapper and the name of the wrapper library file.

For example, to register a wrapper with the name `NotesDBwrapper` on the federated server that uses the Windows operating system, issue the following statement:

```
CREATE WRAPPER NotesDBwrapper LIBRARY 'db2uies.dll'
```

The name of the wrapper library file that you specify depends on the operating system of the federated server. See the list of Extended Search wrapper library files for the correct name to specify in the CREATE WRAPPER statement.

The next task in this sequence of tasks is registering the server definitions for Extended Search wrapper.

Related tasks:
- "Registering the server for Extended Search data sources" on page 237

Related reference:
- "Extended Search wrapper library files" on page 236
- "CREATE WRAPPER statement syntax - Extended Search wrapper" on page 546

Extended Search wrapper library files

The following table lists the directory paths and library file names for the Extended Search wrapper.

When you install DB2 Information Integrator, 3 library files are added to the directory path listed in the table. For example, if the federated server is running on AIX, the wrapper library files added to the directory path are `libdb2uies.a`, `libdb2uiesF.a`, and `libdb2uiesU.a`.

When you register a wrapper, specify only the library file name that is listed in the table.

Table 66. Extended Search wrapper library locations and file names

Operating system	Directory path	Wrapper library file
AIX	/usr/opt/db2_08_01/lib/	libdb2uies.a
Linux	/opt/IBM/db2/V8.1/lib	libdb2uies.so
Solaris	/opt/IBM/db2/V8.1/lib	libdb2uies.so
Windows	%DB2PATH%\bin	db2uies.dll

%DB2PATH% is the environment variable that is used to specify the directory path where DB2 Information Integrator is installed on Windows. The default Windows directory path is `C:\Program Files\IBM\SQLLIB`.

Related tasks:
- "Registering the Extended Search wrapper" on page 236

Registering the server for Extended Search data sources

This task is part of the main task for adding Extended Search data sources to a federated system. After you register a wrapper, you must create a corresponding server definition to identify the remote Extended Search server that you are integrating with your federated system. This definition enables the wrapper to connect to the Extended Search server.

Procedure:

To register the Extended Search server, issue a CREATE SERVER statement from the DB2 Command Line Processor.

For example, to register a server named es1 for a wrapper named myESwrapper, issue the following statement. The Extended Search server uses the default port value.

```
CREATE SERVER es1 WRAPPER myESwrapper OPTIONS (ES_HOST 'my.server.com')
```

To create this same server, enable tracing for all message levels (critical, noncritical, warning, and information), and write the trace messages to a file named `es1wrapper.log` in the `wrapper` directory, issue the following statement:

```
CREATE SERVER es1 WRAPPER myESwrapper OPTIONS (ES_HOST 'my.server.com',
  ES_TRACING 'ON', ES_TRACELEVEL 'CNWI',
  ESTRACEFILENAME '/wrapper/es1wrapper.log')
```

The next task in this sequence of tasks is registering nicknames for Extended Search data sources.

Related tasks:
- "Registering nicknames for Extended Search data sources" on page 239

Related reference:
- "CREATE SERVER statement syntax - Extended Search wrapper" on page 547

Registering user mappings for Extended Search data sources

This task is an optional step in the main task for adding Extended Search data sources to a federated system.

User mappings provide a way to authenticate the access of users who query an Extended Search source with the Extended Search wrapper. If a user submits an SQL query to a registered Extended Search nickname, and no user mappings are defined for that user, the Extended Search wrapper will use a default user ID and password in an attempt to retrieve data from the remote Extended Search server. If a data source that is being queried requires authentication, an empty result set might be returned.

To ensure that the correct user ID and password get passed to the Extended Search server, create user mappings in your federated system for users who are authorized to search Extended Search sources. When you create a user mapping, the password is stored in an encrypted format in a DB2 catalog table. The password remains in a secure format as it is passed from DB2 through Extended Search to the sources that are being searched.

Security settings in the Extended Search configuration database determine whether the user ID and password are authorized to access the sources that are being searched and whether any additional mapping of the user ID will be performed.

Procedure:

To register Extended Search user mappings, issue a CREATE USER MAPPING statement from the DB2 Command Line Processor.

The statement must identify the DB2 user ID that needs to be mapped, the Extended Search server that hosts the target data sources, and the user ID and password that enable the user to access those data sources.

For example, the following statement registers the user1 user ID so that it can use the es1 Extended Search server to search remote databases.

```
CREATE USER MAPPING FOR user1 SERVER es1 OPTIONS
  (REMOTE_AUTHID 'ESUserId', REMOTE_PASSWORD 'abc123def')
```

The next task in this sequence of tasks is registering the Extended Search custom function template.

Related tasks:
- "Registering the custom functions for the Extended Search wrapper" on page 240

Related reference:
- "CREATE USER MAPPING statement syntax - Extended Search wrapper" on page 548

Registering nicknames for Extended Search data sources

This task is part of the main task for adding Extended Search data sources to a federated system. After you register a server, you must register at least one nickname. A nickname table is a virtual DB2 table that identifies one or more searchable sources in an Extended Search domain. When you submit a query, you specify the nickname for the sources that you want to search.

Prerequisites:

Make sure that the Extended Search server for which you are creating nicknames is running. When you create a nickname, the system verifies that information about the sources and fields that you plan to search exists in the Extended Search configuration database.

Procedure:

To register an Extended Search nickname, issue a CREATE NICKNAME statement from the DB2 Command Line Processor.

For example, issue the following statement to create a nickname table for searching all data sources that belong to the Web category in the Demo application that is hosted by the es1 Extended Search server. Return the WebTitle and WebDescription fields and use the default search processing options.

```
CREATE NICKNAME allweb (WebTitle VARCHAR(255), WebDescription VARCHAR(1000))
  FOR SERVER es1 OPTIONS(APPLICATIONID 'Demo', CATEGORY 'Web')
```

Issue the following statement to create a nickname table for searching several data sources in the Science application. Present the search results as a vertical list of column names, set the timeout value to 60 seconds, allow each source to return up to 100 result documents, expand the size of the result set to 1000 entries, and sort the results by author name.

```
CREATE NICKNAME stars (Title VARCHAR(80), Author VARCHAR(40),
        Abstract VARCHAR(200))
  FOR SERVER es1 OPTIONS (APPLICATIONID 'Science',
  DATASOURCES 'Astronomy;NASA Library;Astrophysics', VERTICAL_TABLE 'yes',
  TIMEOUT '60', MAXHITS '100', TOTALMAXHITS '1000', SORTFIELD 'Author')
```

The next task in this sequence of tasks is registering user mappings for the Extended Search wrapper.

Related concepts:

- "Extended Search nicknames" on page 232
- "Extended Search vertical tables" on page 233

Related tasks:

- "Adding Extended Search data sources to a federated server" on page 235
- "Registering user mappings for Extended Search data sources" on page 238
- "Specifying nickname columns for a nonrelational data source" on page 65

Related reference:

- "Extended Search wrapper - Example queries" on page 242
- "CREATE NICKNAME statement syntax - Extended Search wrapper" on page 548

Registering the custom functions for the Extended Search wrapper

This task is an optional step in the main task for adding Extended Search data sources to a federated system.

Custom functions contain no executable code. After you register a function, you can refer to it in queries to alter default search behavior. The custom function for the Extended Search wrapper enables you to specify precise search expressions and search content that is not defined as a column in the nickname table.

Restrictions:
- You can call the Extended Search function only with a WHERE clause.
- The WHERE clause must contain at least one predicate that serves as a search predicate, either the `Extended Search` function or a predicate of type ″column-name operator constant.″
- The Extended Search function is a scalar function template. It must use the EQUAL (=) operator and the comparison value must be one (1).
- The first parameter in the Extended Search function serves as an anchor value for identifying the nickname to which the function should be applied, such as the document’s rank (DOC_RANK) in the search results. You must specify an INTEGER field for this parameter. This parameter, which does not get evaluated, is particularly important if the SQL query contains more than one nickname or a combination of nicknames and tables. For example:

  ```
  SELECT * FROM es_nickname1, es_nickname2
  WHERE eswrapper.es_search(es_nickname1.DOC_RANK, '"IBM"') = 1 AND
        eswrapper.es_search(es_nickname2.DOC_RANK, '"IBM"') = 1
  ```

Procedure:

To register the Extended Search custom function, issue the following CREATE FUNCTION statement:

```
| CREATE FUNCTION eswrapper.es_search(integer, varchar(1024))
|    RETURNS INTEGER AS TEMPLATE
|    DETERMINISTIC NO EXTERNAL ACTION;
```

Related reference:
- “Extended Search wrapper - Example queries” on page 242
- “Extended Search wrapper - Generalized query language” on page 244
- “CREATE FUNCTION statement syntax - Extended Search wrapper” on page 551
- “Extended Search wrapper - Query guidelines” on page 240

Querying Extended Search data sources

Extended Search wrapper - Query guidelines

The Extended Search wrapper expects queries to be in a specific format and does not support queries that do not meet precise language criteria. This topic provides guidelines for creating queries and gives examples of correct and incorrect query syntax.

Querying Web sources in multiple languages

The third-party software that Extended Search uses to link to Web sources supports languages that use the ISO–8859–1 code page (such as English, French, German, Portuguese, and Swedish). Therefore, when you search Web sources, you cannot search double-byte character set languages such as Korean, bi-directional languages such as Hebrew, or other non-ISO–8859–1 languages. The parser that processes search results fails when it detects what it regards as illegal character codes.

Specifying the CLIENT_LOCALE value

If you include the CLIENT_LOCALE column in a WHERE clause to set the value of the client locale, you must use an AND predicate to specify the search criteria. You cannot use an OR predicate with the CLIENT_LOCALE column.

Examples — correct syntax

The following examples show the correct way to include the CLIENT_LOCALE column in a WHERE clause:

```
WHERE CLIENT_LOCALE = 'enUS' AND
ESWRAPPER.ES_SEARCH(DOC_RANK, '"IBM"')=1

WHERE ESWRAPPER.ES_SEARCH(DOC_RANK, '"IBM"')=1
AND CLIENT_LOCALE = 'enUS'
```

Examples — incorrect syntax

The following examples are incorrect because they attempt to use an OR predicate with the CLIENT_LOCALE column:

```
WHERE CLIENT_LOCALE = 'enUS' OR
ESWRAPPER.ES_SEARCH(DOC_RANK, '"IBM"')=1

WHERE ESWRAPPER.ES_SEARCH(DOC_RANK, '"IBM"')=1
OR CLIENT_LOCALE = 'enUS'
```

Specifying predicates on Extended Search fixed columns

An SQL statement that contains an Extended Search nickname must specify a predicate for the nickname in the WHERE clause. However, a predicate on an Extended Search fixed column does not count as a predicate.

Examples — incorrect syntax

The following example shows a query that is incorrect because it does not contain a predicate:

```
SELECT * FROM ES_NICKNAME
```

The following example shows a query that is incorrect because the only predicate is on a fixed column:

```
SELECT * FROM ES_NICKNAME WHERE DOC_RANK < 20
```

Specifying unbound predicates

A predicate on a user-defined column will be handled by the Extended Search wrapper only if the predicate value is a constant. If the predicate value is unbound, the predicate will be handled by the DB2 engine. If an unbound predicate is the only predicate in an SQL statement, an error will result. An Extended Search nickname requires a predicate that can be handled by the Extended Search wrapper.

Examples — correct syntax

The WHERE statement in the following example shows a predicate that will be handled by the Extended Search wrapper:

```
SELECT *
FROM   ES_NICKNAME
WHERE  Author = 'Ernest Hemingway'
```

Examples — incorrect syntax

The WHERE statement in the following example shows a predicate that will be handled by DB2:

```
SELECT *
FROM   ES_NICKNAME_1, ES_NICKNAME_2
WHERE  ES_NICKNAME_1.Author = ES_NICKNAME_2.Author
```

Joining queries with an OR predicate

The Extended Search wrapper cannot search different nickname tables, or nickname tables and database tables, that are joined by a simple OR predicate. You can use an OR predicate only within the same nickname.

Examples — incorrect syntax

```
SELECT *
FROM   ES_Nickname as N1, TABLE as T1
WHERE  N1.Column1 = 'abc' OR T1.Column1 = 'abc'

SELECT *
FROM  ES_Nickname_1 as N1, ES_Nickname_2 as N2
WHERE N1.USerdefCol = 'abc' OR N2.USerdefCol = 'cdf'

SELECT *
FROM  ES_Nickname_1 as N1, ES_Nickname_2 as N2
WHERE ESWRAPPER.ES_SEARCH(N1.DOC_RANK, '"IBM"')=1 OR
      ESWRAPPER.ES_SEARCH(N2.DOC_RANK, '"LOTUS"')=1
```

Related tasks:

- "Registering the custom functions for the Extended Search wrapper" on page 240

Related reference:

- "Extended Search wrapper - Example queries" on page 242
- "Extended Search wrapper - Generalized query language" on page 244

Extended Search wrapper - Example queries

To run queries with the Extended Search wrapper, you specify a registered nickname and nickname columns in your SQL statements the same way that you specify a typical DB2 table name and table columns.

In this sample search scenario, a hospital team needs to search and compare the latest medical research. To search a wide variety of sources, the hospital uses an Extended Search server. The Extended Search domain includes an application named MedResearch and several categories that are configured to search document-based databases, mail servers, and the Web.

In addition to searching, the team needs to compare the results from various searches. For example, they need to identify people who published articles within a certain time frame, recently purchased herbs and vitamins, discussed alternative medicine with colleagues through e-mail, and applied to renew a medical license. The Extended Search wrapper, with its ability to integrate unstructured Extended Search data into DB2 for structured retrieval, provides the solution.

The hospital team decides to create the following three nicknames, one for searching document repositories, one for searching e-mail systems, and one for searching specific Web sources. The Owner and Date fields are defined as mapped fields in the Extended Search configuration database, which enables you to use them in joins regardless of how the fields are named in the native data sources.

Document nickname:

```
CREATE NICKNAME MedDocs ( Owner     VARCHAR(80),
                          Date      DATE,
                          Title     VARCHAR(80),
                          Abstract  VARCHAR(200) )
FOR SERVER esServer OPTIONS ( APPLICATIONID 'MedResearch',
                          CATEGORY 'AMA Library;Medical Records;Pharmacy',
                          VERTICAL_TABLE 'YES',
                          TIMEOUT '60', MAXHITS '100',
               TOTALMAXHITS '1000' )
```

E-mail nickname:

```
CREATE NICKNAME MedMail ( Owner    VARCHAR(80),
                          To       VARCHAR(80),
                          Date     DATE,
                          Subject  VARCHAR(80) )
FOR SERVER esServer OPTIONS ( APPLICATIONID 'MedResearch',
                              CATEGORY 'Exchange Server;Lotus Notes',
                              VERTICAL_TABLE 'YES', )
                              TIMEOUT '60', MAXHITS '100',
                              TOTALMAXHITS '1000' )
```

Web nickname:

```
CREATE NICKNAME MedWeb ( WebTitle VARCHAR(255),
                         WebDescription VARCHAR(1000) )
FOR SERVER esServer OPTIONS ( APPLICATIONID 'MedResearch',
                              DATASOURCES 'Google!;Alta Vista;CNN',
                              TOTALMAXHITS '500' )
```

The following query searches for documents that contain the phrase Artificial Liver in the title and the abbreviation MARS in the document content. The result set should exclude any documents that were published before the year 2001.

```
SELECT OWNER, DOC_CONTENT
FROM   MedDocs
WHERE  ESWRAPPER.ES_Search(DOC_RANK, '( ( TOKEN:EXACT "MARS") AND
                          ( ("TITLE" IN "Artificial Liver") AND
                          ("DATE" >= "01/01/2001") ) ) ') = 1
```

The following query searches for e-mail that was written during the past few months that discussed alternative medicine:

```
SELECT *
FROM   MedMail
WHERE  ESWRAPPER.ES_Search(DOC_RANK, '(
                          ("SUBJECT" IN "alternative medicine") AND
                          ("DATE" BETWEENI "03/01/2002" AND
                           "09/30/2002") ) ') = 1
```

The following query searches Web sources that refer to complementary and alternative medicine (CAM) therapy and its acceptance by the American public:

```
SELECT WebTitle, WebDescription
FROM   MedWeb
WHERE  ESWRAPPER.ES_Search(DOC_RANK, '(
                           TOKEN:EXACT "CAM therapy" ) AND
                           ( TOKEN:FUZZY "United States" ) ' ) = 1
```

The following query searches for recently licensed doctors who purchased large quantities of herbs or vitamins from the hospital pharmacy. The query then matches up the names of those doctors with persons who wrote e-mail about alternative medicine.

```
SELECT  N2.OWNER, N2.DATE
FROM    MedDocs as N1,
MedMail as N2
WHERE   ESWRAPPER.ES_SEARCH(N1.DOC_RANK, ' (
                           ("LICENSE_DATE" >= "01/01/2002")  AND
                  ( ( ( "PRODUCT" = "HERB") OR ("PRODUCT" = "VITAMIN") ) AND
                  ("QUANTITY" > "1000") ) ) ' ) = 1
AND ESWRAPPER.ES_SEARCH(N2.DOC_RANK, ' ("SUBJECT" IN
                          "alternative medicine") ') = 1
AND N1.OWNER = N2.OWNER
```

Related concepts:

- "Extended Search nicknames" on page 232
- "Extended Search vertical tables" on page 233

Related tasks:

- "Registering nicknames for Extended Search data sources" on page 239
- "Registering the custom functions for the Extended Search wrapper" on page 240

Related reference:

- "Extended Search wrapper - Generalized query language" on page 244
- "CREATE FUNCTION statement syntax - Extended Search wrapper" on page 551
- "CREATE NICKNAME statement syntax - Extended Search wrapper" on page 548
- "Extended Search wrapper - Query guidelines" on page 240

Extended Search wrapper - Generalized query language

Queries that you pass to an Extended Search server through the Extended Search wrapper can contain search expressions in generalized query language (GQL), the query language of Extended Search.

For example, assume a user wants to find all employees whose names start with JO in a relational database that contains a table with employee information. You might issue the following query in GQL:

```
(LIKE "EMPLOYEE_NAME" "JO")
```

You might issue the same query in SQL as follows:

```
SELECT * FROM EMP.TABLE WHERE EMPLOYEE_NAME LIKE JO%
```

Like SQL, the wrapper supports infix notation, a syntax that requires operators to be between the field name and a comparison value. The native Extended Search GQL grammar uses prefix notation, a syntax that requires operators to precede the fields and values that you want to evaluate.

Compare the following query expressions that search for documents that contain the word IBM in the TITLE field:

Infix GQL
```
("TITLE" IN "IBM")
```

Prefix GQL
```
(IN "TITLE" "IBM")
```

When you submit a query with the Extended Search wrapper, the API converts the infix SQL statements to prefix GQL for processing by Extended Search.

The following syntax description shows the Backus-Naur Form specification for the Extended Search grammar that you can use in queries.

```
expr:           pattern_expr
              | bool_expr
              | field_expr
              | prox_expr

pattern_expr:   STRING
              | token_expr

token_expr:     ( TOKEN [:CASE] [:STEM] [:EXACT] [:WEIGHT "x"]
                        [:WILD] [:FUZZY] STRING )

bool_expr:      (expr_list bool_operator [:WEIGHT "x"] expr )

bool_text_expr: (text_expr_list  bool_operator [:WEIGHT "x"] text_expr )

text_expr:      pattern_expr
              | bool_text_expr
              | prox_expr

text_expr_list: text_expr
              | text_expr_list text_expr

expr_list:      expr
              | expr_list expr

field_expr:    ( field_name operator_1 [:WEIGHT "x"] text_expr )
              | ( field_name operator_2 [:WEIGHT "x"] value )
              | ( field_name operator_3 [:WEIGHT "x"] value_1 AND value_2 )
              | ( field_name operator_4 value )

prox_expr:    ( prox_op [:COUNT "x"][:ORDER][:MATH "y"][:WEIGHT "x"]
                expr_list expr )

prox_op:        DOCUMENT
              | PARAGRAPH
              | SENTENCE
              | WORD
              | CHARACTER

operator1:      START
              | END
              | IN
              | =
operator_2:     =
              | >
              | >=
              | <
              | <=
              | EQ
              | GT
              | GTE
              | LT
              | LTE

operator_3:     BETWEENI
              | BETWEENE
              | LIKE

bool_operator:  AND
              | OR
              | NOT
```

For complete information about the GQL grammar, see *Extended Search Programming*, which is available on the Resources page of the IBM Lotus Extended Search Web site:

`http://www.lotus.com/products/des.nsf/wdocuments/resources`

Related tasks:

- "Registering the custom functions for the Extended Search wrapper" on page 240

Related reference:

- "Extended Search wrapper - Example queries" on page 242
- "CREATE FUNCTION statement syntax - Extended Search wrapper" on page 551
- "Extended Search wrapper - Query guidelines" on page 240

Messages for the Extended Search wrapper

This topic describes messages that you might encounter while you work with the Extended Search wrapper.

Table 67. Messages issued by the wrapper for Extended Search

Error Code	Message	Explanation
SQL0901N	The SQL statement failed because of a non-severe system error. Subsequent SQL statements can be processed. (Reason: INTERNAL Extended Search WRAPPER ERROR - RC: *xxx*.)	Record the reason code (a number from 901 to 999) and contact IBM Software Support.

Table 67. Messages issued by the wrapper for Extended Search (continued)

Error Code	Message	Explanation
SQL0973N	Not enough storage is available in the Application heap to process the statement.	The Extended Search wrapper was not able to allocate memory in the Application heap. To resolve the problem, increase the Application heap size and try the statement again. For example: `db2 update db cfg for db-name using heap-name heap-size` If the error continues after you increase this value, contact IBM Software Support.
SQL1822N	Unexpected error code ″<error_code>″ received from data source ″Extended Search wrapper″. Associated text and tokens are ″<tokens>″.	The remote Extended Search server returned an error while processing a search request. The error also returned a token that indicates what caused the error on the remote server. If tracing is enabled for the Extended Search server, review the trace log file for diagnostic help.
SQL1823N	No data type mapping exists for data type ″<data_type>″ from server ″<server_name>″.	A column in a CREATE NICKNAME statement or ALTER NICKNAME statement uses a data type that is not supported by the Extended Search system. This error can also occur during query processing. To solve the problem if it occurs while the query is being processed, drop the nickname table and create a new nickname.
SQL1825N	This SQL statement cannot be handled in a federated environment.	The current SQL statement cannot be handled by the Extended Search wrapper. To solve the problem, see the Extended Search wrapper documentation, change the SQL statement as needed, and submit the request again.
SQL1833N	Connection to remote Extended Search server ″<host_name>″ on port ″<port_number>″ could not be established or was terminated.	The Extended Search wrapper tried to connect to the remote Extended Search server at the specified port but the connection could not be established or was terminated by the remote server. Verify the host name and port number of the remote Extended Search server, make sure that the Extended Search server is running, and try again.
SQL1834N	User-defined column ″<column_name>″ is identical to a fixed column for wrapper ″<wrapper_name>″ but uses a different data type.	A CREATE NICKNAME statement or ALTER NICKNAME statement contains a user-defined column that has the same name as a fixed column for the specified Extended Search wrapper but uses a different data type. You do not need to specify fixed columns in the column definition of a CREATE NICKNAME statement. If you do, make sure that the fixed column name, data type, and data type length match the fixed column definition. You cannot ALTER a fixed column name or data type.

Table 67. Messages issued by the wrapper for Extended Search (continued)

Error Code	Message	Explanation
SQL1835N	Extended Search object ″<object_name>″ of type ″<object_type>″ could not be found on the remote Extended Search server ″<host_name>″.	The specified Extended Search object could not be found on the specified remote Extended Search server. Verify that the object name is defined on this Extended Search server and that it is of the specified object type. Also verify that the spelling of this object is correct.
SQL1836N	No column mapping exists between user-defined column ″<column_name>″ and a field name on the remote Extended Search server ″<host_name>″.	None of the data sources that are included in a DATASOURCE or CATEGORY option contain a field name the matches the specified user-defined column name. Verify that the column name is a field in at least one of the data sources in the DATASOURCE option, or in at least one of the data sources that belongs to a category in CATEGORY option, and submit the statement again.
SQL1837N	The required option ″<option_name>″ of type ″<object_type>″ on wrapper ″<wrapper_name>″ cannot be dropped.	You cannot drop a required option. Change the ALTER statement to use SET instead of DROP. Correct the search statement and submit the request again. Consult the DB2 SQL Reference for information about creating valid SQL search statements. If the search statement includes the ES_SEARCH function, consult the Extended Search wrapper documentation for information about using Extended Search generalized query language (GQL).
SQL1838N	The search statement ″<option_name>″ is not a valid Extended Search query.	The Extended Search wrapper tried to process the specified search statement but the query failed because the statement does not use proper query syntax. Consult the *DB2 SQL Reference* for information about creating valid SQL search statements. If the search statement includes the ES_SEARCH function, consult the Extended Search wrapper documentation for information about using Extended Search generalized query language (GQL).
SQL1839N	One or more search parameters are not valid.	The Extended Search wrapper tried to use the specified search parameters, but they are not valid for Extended Search. Consult the Extended Search wrapper documentation, correct the invalid parameters, and submit the request again.
SQL1881N	″<option_name>″ is not a valid ″<option_type>″ option for ″<object_name>″.	The specified option is not valid for the specified object (wrapper, server, nickname, column, or user mapping). See the Extended Search wrapper documentation, remove or change the invalid option, and submit the statement again.
SQL1882N	The ″<option_type>″ option ″<option_name>″ cannot be set to ″<option_value>″ for ″<object_name>″.	The specified option value is not valid for the specified object (wrapper, server, nickname, column, or user mapping). See the Extended Search wrapper documentation, change the invalid option value, and submit the statement again.

Table 67. Messages issued by the wrapper for Extended Search *(continued)*

Error Code	Message	Explanation
SQL1883N	″<option_name>″ is a required ″<option_type>″ option for ″<object_name>″.	A required option for the Extended Search wrapper was missing from the statement to create, alter, or initialize the specified object (wrapper, server, nickname or user mapping). See the Extended Search wrapper documentation, add the required option, and submit the statement again.

For more information about messages, see the *DB2 Message Reference*. You might also want to consult the Extended Search product messages in *Extended Search Administration*. If you receive errors about improper GQL query syntax, see *Extended Search Programming*. The Extended Search documents are available on the Resources page of the IBM Lotus Extended Search Web site:

`http://www.lotus.com/products/des.nsf/wdocuments/resources`

Related concepts:

- “Introduction to messages” in the *Message Reference Volume 1*

Related reference:

- “SQLSTATE messages” in the *Message Reference Volume 2*
- “sql0900” in the *Message Reference Volume 2*
- “sql1800” in the *Message Reference Volume 2*

Chapter 14. Configuring access to HMMER data sources

This chapter explains how to configure your federated server to access data that is stored in HMMER data sources. You can configure access to HMMER data sources by using the DB2 Control Center or by issuing SQL statements.

This chapter:
- Explains what HMMER is
- Lists the tasks that you need to perform
- Contains examples of the SQL statements that you need
- Lists the error messages associated with the HMMER wrapper

What is HMMER?

HMMER is a application package that you can use to search gene sequence databases that use statistical models or profile hidden Markov models (HMMs). You can download the HMMER application package at no charge from http://hmmer.wustl.edu/. You can install the HMMER application package on a separate HMMER server or on the federated server.

An HMM is a statistical model of the primary structure consensus of a gene sequence family. An HMM is based upon probability models. You can train an HMM to recognize patterns from unaligned gene sequences if a trusted alignment is not yet known. You need less skill and manual intervention to train and use a successful HMM than to carefully construct a profile. You can use a trained HMM to access libraries of hundreds of profile HMMs and apply them on a very large scale to whole genome or Expressed Sequence Tag (EST) analyses.

PFAM (Protein Families Database of Alignments and HMMs) is a database of protein domain models. The HMMER application package is tightly tied to the construction and use of the PFAM database.

The HMMER application package contains the 9 programs, but only two of these programs are supported by the DB2® Information Integrator, the hmmpfam and the hmmsearch programs.

Table 68. The HMMER programs supported by the HMMER wrapper

HMMER program	Description
hmmpfam	Uses a specific gene sequence to search an HMM database and determine the family that the test gene sequence might belong to. Calculates how well each model matches a specified sequence and a database of models. The match is expressed in terms of statistical significance.
hmmsearch	Uses a specific HMM profile to search a sequence database for significantly similar sequence matches.

Users or applications issue SQL query statements with HMMER-specific predicates to the federated server. The predicates in these statements map to command-line options in the hmmpfam or hmmsearch programs.

The HMMER wrapper transforms the query statements into a format that the HMMER application package can interpret and starts the hmmpfam program or the hmmsearch program to run the query.

A special daemon program runs on the server where the HMMER application package is installed. This daemon receives the query request from the federated server and sends it to the HMMER application package. The HMMER application package runs the query on a profile database, such as PFAM.

Figure 25 shows how HMMER works with your federated system.

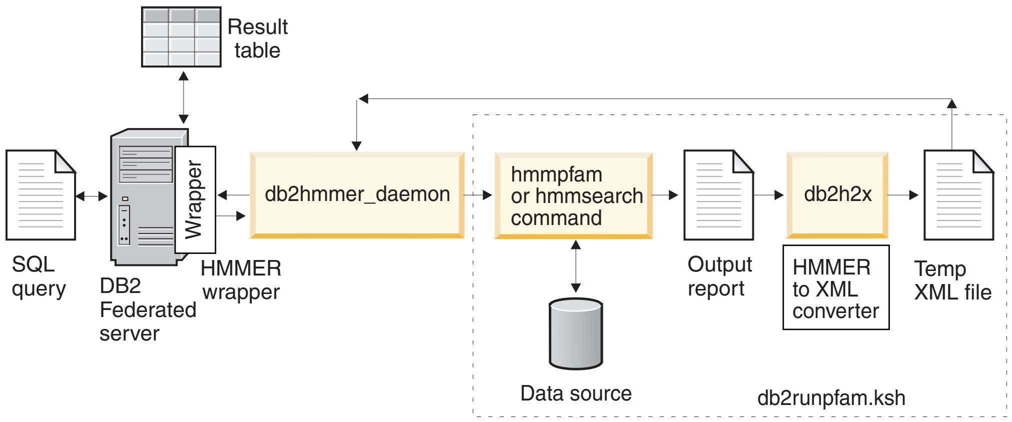


Figure 25. How the HMMER wrapper works

The daemon returns the results to the HMMER wrapper. The wrapper transforms the data into a relational table, and returns this table to the user or application.

The following example shows how information is extracted from profile databases, which are constructed by HMMER programs, and displayed as a relational table. The HMMER User's Guide http://hmmer.wustl.edu/ provides examples of creating profile databases and a HMMER tutorial.

Figure 26 on page 253 shows a sample query that uses the 7LES_DROME gene sequence. You specify sequences in the WHERE clause of the query.

```
SELECT Model, ModelScore, DomainNumber, DomainScore
FROM myhmms
WHERE HmmQSeq = 'MTMFWQQNVDHQSDEQDKQAKGAAPTKRLNISFNVKIAVNVNTKMTTTH
INQQAPGTSSSSSNSQNASPSKIVVRQQSSSFDLRQQLARLGRQLASGQDGHGGISTILIINLLLL
ILLSICCDVCRSHNYTVHQSPEPVSKDQMRLLRPKLDSDVVEKVAIWHKHAAAAPPSIVEGIAISS
RPQSTMAHHPDDRDRDRDPSEEQHGVDERMVLERVTRDCVQRCIVEEDLFLDEFGIQCEKADNGEK
CYKTRCTKGCAQWYRALKELESCQEACLSLQFYPYDMPCIGACEMAQRDYWHLQRLAISHLVERTQ
PQLERAPRADGQSTPLTIRWAMHFPEHYLASRPFNIQYQFVDHHGEELDLEQEDQDASGETGSSAW
FNLADYDCDEYYMCEILEALIPYTQYRFRFELPFGENRDEVLYSPATPAYQTPPEGAPISAPVIEH
LMGLDDSHLAVHWHPGRFTNGPIEGYRLRLSSSEGNATSEQLVPAGRGSYIFSQLQAGTNYTLALS
MINKQGEGPVAKGFVQTHSARNEKPAKDLTESVLLVGRRAVMWQSLEPAGENSMIYQSQEELADIA
WSKREQQLWLLNVHGELRSLKFESGQMVSPAQQLKLDLGNISSGRWVPRRLSFDWLHHRLYFAMES
PERNQSSFQIISTDLLGESAQKVGESFDLPVEQLEVDALNGWIFWRNEESLWRQDLHGRMIHRLLR
IRQPGWFLVQPQHFIIHLMLPQEGKFLEISYDGGFKHPLPLPPPSNGAGNGPASSHWQSFALLGRS
LLLPDSGQLILVEQQGQAASPSASWPLKNLPDCWAVILLVPESQPLTSAGGKPHSLKALLGAQAAK
ISWKEPERNPYQSADAARSWSYELEVLDVASQSAFSIRNIRGPIFGLQRLQPDNLYQLRVRAINVD
GEPGEWTEPLAARTWPLGPHRLRWASRQGSVIHTNELGEGLEVQQEQLERLPGPMTMVNESVGYYV
TGDGLLHCINLVHSQWGCPISEPLQHVGSVTYDWRGGRVYWTDLARNCVVRMDPWSGSRELLPVFE
ANFLALDPRQGHLYYATSSQLSRHGSTPDEAVTYYRVNGLEGSIASFVLDTQQDQLFWLVKGSGAL
RLYRAPLTAGGDSLQMIQQIKGVFQAVPDSLQLLRPLGALLWLERSGRRARLVRLAAPLDVMELPT
PDQASPASALQLLDPQPLPPRDEGVIPMTVLPDSVRLDDGHWDDFHVRWQPSTSGGNHSVSYRLLL
EFGQRLQTLDLSTPFARLTQLPQAQLQLKISITPRTAWRSGDTTRVQLTTPPVAPSQPRRLRVFVE
RLATALQEANVSAVLRWDAPEQGQEAPMQALEYHISCWVGSELHEELRLNQSALEARVEHLQPDQT
YHFQVEARVAATGAAAGAASHALHVAPEVQAVPRVLYANAEFIGELDLDTRNRRRLVHTASPVEHL
VGIEGEQRLLWVNEHVELLTHVPGSAPAKLARMRAEVLALAVDWIQRIVYWAELDATAPQAAIIYR
LDLCNFEGKILQGERVWSTPRGRLLKDLVALPQAQSLIWLEYEQGSPRNGSLRGRNLTDGSELEWA
TVQPLIRLHAGSLEPGSETLNLVDNQGKLCVYDVARQLCTASALRAQLNLLGEDSIAGQLAQDSGY
LYAVKNWSIRAYGRRRQQLEYTVELEPEEVRLLQAHNYQAYPPKNCLLLPSSGGSLLKATDCEEQR
CLLNLPMITASEDCPLPIPGVRYQLNLTLARGPGSEEHDHGVEPLGQWLLGAGESLNLTDLLPFTR
YRVSGILSSFYQKKLALPTLVLAPLELLTASATPSPPRNFSVRVLSPRELEVSWLPPEQLRSESVY
YTLHWQQELDGENVQDRREWEAHERRLETAGTHRLTGIKPGSGYSLWVQAHATPTKSNSSERLHVR
SFAELPELQLLELGPYSLSLTWAGTPDPLGSLQLECRSSAEQLRRNVAGNHTKMVVEPLQPRTRYQ
CRLLLGYAATPGAPLYHGTAEVYETLGDAPSQPGKPQLEHIAEEVFRVTWTAARGNGAPIALYNLE
ALQARSDIRRRRRRRRRNSGGSLEQLPWAEEPVVVEDQWLDFCNTTELSCIVKSLHSSRLLLFRVR
ARSLEHGWGPYSEESERVAEPFVSPEKRGSLVLAIIAPAAIVSSCVLALVLVRKVQKRRLRAKKLL
QQSRPSIWSNLSTLQTQQQLMAVRNRAFSTTLSDADIALLPQINWSQLKLLRFLGSGAFGEVYEGQ
LKTEDSEEPQRVAIKSLRKGASEFAELLQEAQLMSNFKHENIVRLVGICFDTESISLIMEHMEAGD
LLSYLRAARATSTQEPQPTAGLSLSELLAMCIDVANGCSYLEDMHFVHRDLACRNCLVTESTGSTD
RRRTVKIGDFGLARDIYKSDYYRKEGEGLLPVRWMSPESLVDGLFTTQSDVWAFGVLCWEILTLGQ
QPYAARNNFEVLAHVKEGGRLQQPPMCTEKLYSLLLLCWRTDPWERPSFRRCYNTLHAISTDLRRT
QMASATADTVVSCSRPEFKVRFDGQPLEEHREHNERPEDENLTLREVPLKDKQLYANEGVSRL'
```

Figure 26. Sample query run on 7LES_DROME data

The HMMER wrapper transforms the results from the query into the relational table shown in Table 69.

Table 69. The HMMER results are transformed into a relational table

Model	ModelScore	DomainNumber	DomainScore
pkinase	+3.04100000000000E+002	1	+3.04100000000000E+002
fn3	+1.76300000000000E+002	1	+4.90000000000000E+001
fn3	+1.76300000000000E+002	2	+1.36000000000000E+001
fn3	+1.76300000000000E+002	3	+1.62000000000000E+001
fn3	+1.76300000000000E+002	4	+6.35000000000000E+001
fn3	+1.76300000000000E+002	5	+1.46000000000000E+001
fn3	+1.76300000000000E+002	6	+1.94000000000000E+001
rrm	-4.45000000000000E+001	1	-4.45000000000000E+001

The data is now in a relational format and can be joined with data from other data sources.

Related tasks:

- "Adding HMMER data sources to a federated server" on page 254

Adding HMMER to a federated server

Adding HMMER data sources to a federated server

To configure the federated server to access HMMER data sources, you must provide the federated server with information about the data sources and objects that you want to access.

| You can configure the federated server to access HMMER data sources by using
| the DB2 Control Center or the DB2 command line. The DB2 Control Center
| includes a wizard to guide you through the steps required to configure the
| federated server.

Prerequisites:

- DB2 Information Integrator must be installed on a server that will act as the federated server
- A federated database must exist on the federated server

Procedure:

To add HMMER data sources to a federated server:

1. Verify that the correct version of the HMMER program executable files are installed.
2. Configure the HMMER daemon.
3. Start the HMMER daemon.
4. Register the wrapper.
5. Register the server definitions.
6. Register the nicknames.

Related concepts:

- "DB2 Information Integrator installation process - overview" in the *IBM DB2 Information Integrator Installation Guide for Linux, UNIX, and Windows*
- "What is HMMER?" on page 251

Related tasks:

- "Creating a federated database" on page 51
- "Verifying the version of the HMMER program executable" on page 255
- "Configuring the HMMER daemon" on page 255
- "Registering the HMMER wrapper" on page 262
- "Registering the server definition for a HMMER data source" on page 263
- "Registering nicknames for HMMER data sources" on page 265
- "Checking the setup of the federated server" on page 37

Related reference:

- “Supported operating systems for DB2 Information Integrator (32-bit)” in the *IBM DB2 Information Integrator Installation Guide for Linux, UNIX, and Windows*
- “Supported operating systems for DB2 Information Integrator (64-bit)” in the *IBM DB2 Information Integrator Installation Guide for Linux, UNIX, and Windows*
- “HMMER data source – complete example” on page 270

Verifying the version of the HMMER program executable

Verifying that the required version of the HMMER program executable is installed is part of the larger task of adding HMMER data sources to a federated server.

You must have a supported version of the hmmpfam and hmmsearch executable files installed on the server where the HMMER application program is installed. The HMMER application program can be installed on the federated server or a separate HMMER server.

Procedure:

To check the version level of the executable file:

1. Issue a command that returns the version number:
   - For the hmmpfam program, the command is:
     ```
     hmmpfam -h
     ```
   - For the hmmsearch program, the command is:
     ```
     hmmsearch -h
     ```
2. In the output file, check the version of the executable files. You must have HMMER version 2.2g (or later).
3. If you do not have the correct version, download the files from http://hmmer.wustl.edu/.

The next task in this sequence of tasks is configuring the HMMER daemon.

Related tasks:

- “Configuring the HMMER daemon” on page 255

Configuring the HMMER daemon

Configuring the HMMER daemon is part of the larger task of adding HMMER to a federated server.

The HMMER wrapper requires a HMMER daemon. The HMMER daemon must be running on a server that you can access through TCP/IP from your DB2 Universal Database federated system. This can be the same server that operates as the federated server, or a separate HMMER server.

The daemon runs separately from the wrapper and DB2 Universal Database. The daemon listens for HMMER job requests from the wrapper.

Prerequisites:

The HMMER daemon must have:

- Execute access to the `hmmpfam` and `hmmsearch` executable files so that it can run HMMER searches.

- Write access to a directory in which it can write temporary files.
- Read access to at least one profile database on which you can run HMMER searches.

Restrictions:

The HMMER daemon might not run properly if the executable file or the database path contains spaces. For example, you should not install the HMMER executable file in `C:\Program Files` on Windows servers.

Procedure:

To configure the HMMER daemon:

1. Ensure that the HMMER daemon executable files are on the proper server. During the installation of DB2 Information Integrator, the daemon executable files are installed in a directory on the federated server:

   On UNIX
   The daemon executable files are installed in the `$DB2PATH/bin` directory.

   On Windows
   The daemon executable files are installed in the `%DB2PATH%\bin` directory.

   If you use a separate HMMER server, you must copy the daemon executable files from the directory on the federated server to a directory on the HMMER server. The daemon executable files can run in any directory on the HMMER server that does not contain spaces in the names in the directory path.
2. Ensure that the configuration file and other required files are on the server where HMMER is installed. Some of the required files are installed with DB2 Information Integrator in a directory on the federated server. You must provide the other required files.

   On UNIX
   The files that must be on the server where HMMER is installed are:
   - The daemon executable file, `%DB2PATH%/bin/db2hmmer_daemon`
   - The HMMER daemon configuration file, `%DB2PATH%/samples/lifesci/HMMER_DAEMON.config`
   - The conversion utility, `%DB2PATH%/bin/db2h2x`
   - The shell script, `%DB2PATH%/bin/db2runpfam.ksh`
   - The HMMER executable files (not supplied by IBM), `hmmpfam` and `hmmsearch`
   - The HMMER database files (not supplied by IBM)

   `%DB2PATH%` is the path where DB2 Information Integrator is installed.

   On Windows
   The files that must be on the server where HMMER is installed are:
   - The daemon executable files, `%DB2PATH%\bin\db2hmmer_daemon.exe` and `%DB2PATH%\bin\db2hmmer_daemon_svc.exe`
   - The HMMER daemon configuration file, `%DB2PATH%\samples\lifesci\HMMER_DAEMON.config`
   - The conversion utility, `%DB2PATH%\bin\db2h2x.exe`
   - The HMMER executable files (not supplied by IBM), `hmmpfam.exe` and `hmmsearch.exe`
   - The HMMER database files (not supplied by IBM)

`%DB2PATH%` is the path where DB2 Information Integrator is installed.

By default, the daemon expects to find the configuration file in the working directory from which the daemon is started. You can copy the configuration file to another location. If you use a HMMER server, you must copy the daemon configuration file from the directory on the federated server to a directory on the HMMER server. You can copy the daemon configuration file to any directory on the HMMER server that the daemon can access.

3. On UNIX, ensure that the HMMER daemon executable file, conversion utility, and the shell script are executable. To make the files executable, run the following command:

```
chmod a+x db2hmmer_daemon db2h2x db2runpfam.ksh
```

4. Edit the daemon configuration file to work with your data source. You can also rename the configuration file.
   - The first line in the configuration file must be an equal sign. If the equal sign is missing, the daemon will not start. An error message will indicate that the DAEMON_PORT was not specified.
   - The last line in the configuration file must end with a new line. The sample configuration file that is provided with DB2 Information Integrator ends with a new line. When you edit the file, you must ensure that the last line in the file ends in a new line. If the last line does not end with a new line, you will receive an error message when you attempt to run your first HMMER query using the data source listed on the last line.
   - Specify the following options in the configuration file. For options that require paths, you can specify relative paths. Relative paths are relative to the directory from which the daemon process was started.

DAEMON_PORT
This is the network port on which the daemon listens for HMMER job requests submitted by the wrapper.

MAX_PENDING_REQUESTS
This is the maximum number of HMMER job requests that can be blocking on the daemon at any one time. This number does not represent the number of HMMER jobs that run concurrently, only the number of job requests that can block at one time. It is recommended that you set this to a number greater than five. The HMMER daemon does not restrict the number of HMMER jobs that can run concurrently.

DAEMON_LOGFILE_DIR
This is the directory in which the daemon creates its log file. This file contains useful status and error information generated by the HMMER daemon.

Q_SEQ_DIR_PATH
This is the directory in which a temporary query sequence data file is created by the daemon. This temporary file is cleaned up once the HMMER job completes.

HMMER_OUT_DIR_PATH
This is the directory in which the daemon creates the temporary file to store the HMMER output data. Data is read from this file and passed back to the wrapper through the network connection. After the data is passed to the wrapper, the daemon cleans up the temporary file.

RUNPFAM_PATH
: This is the fully-qualified name of the `db2runpfam.ksh` shell script provided with DB2 Information Integrator. This option is ignored if it is specified on Windows.

HMMERPFAM_PATH
: This is the fully-qualified name of the HMMER executable file on the computer that is running the daemon. On UNIX, the name of the file is `hmmpfam`. On Windows, the name of the file is `hmmpfam.exe`.

HMMSEARCH_PATH
: This is the fully-qualified name of the HMMER executable file on the computer that is running the daemon. On UNIX, the name of the file is `hmmsearch`. On Windows, the name of the file is `hmmsearch.exe`.

H2X_PATH
: This is the fully-qualified name of the conversion program (HMMER to XML) provided with the daemon. On UNIX, the name of the program is `db2h2x`. On Windows, the name of the program is `db2h2x.exe`.

database specification entry
: Specifies the location of a profile database or sequence file. Make note of the database *data_source_name* that you specify in the configuration file. For the daemon to function properly, you must specify the database *data_source_name* when you create the nickname for the data source. The name is case-sensitive. The database *data_source_name* is specified in:

- The DATASOURCE option of the CREATE NICKNAME statement (for hmmpfam)
- The MODEL predicate of the CREATE NICKNAME statement (for hmmsearch)

The configuration file must contain at least one database specification entry in the following form:

```
data_source_name=fully_qualified_name_of_profile_or_sequence_database
```

On UNIX
: For example, to specify the MYHMMS profile database you would add the following line to the daemon configuration file :

```
myhmms=/home/user_ID/myhmms
```

On Windows
: For example, to specify MYHMMS profile database you would add the following line to the daemon configuration file:

```
myhmms=c:\hmmer\tutorial\myhmms
```

The next task in this sequence of tasks is starting the HMMER daemon.

Related tasks:
- "Starting the HMMER daemon" on page 259

Related reference:
- "HMMER daemon configuration file - examples" on page 258

HMMER daemon configuration file - examples

The following examples show the contents of a sample configuration file for PFAM and SEARCH.

Example – HMMER_DAEMON.config file for UNIX:

This example shows the required options and profile database specification for UNIX.

```
=
DAEMON_PORT=4098
MAX_PENDING_REQUESTS=10
DAEMON_LOGFILE_DIR=./
Q_SEQ_DIR_PATH=./
HMMER_OUT_DIR_PATH=./
RUNPFAM_PATH=./db2runpfam.ksh
HMMPFAM_PATH=/home/user_id/hmmer/bin/hmmpfam
HMMSEARCH_PATH=/home/user_id/hmmer/bin/hmmsearch
H2X_PATH=/home/user_id/sqllib/bin/db2h2x
myhmms=/home/user_id/hmmer/tutorial/myhmms
globin=/home/user_id/hmmer/tutorial/globin.hmm
pfamls=/home/user_id/hmmer/pfam/Pfam_ls
```

| **Example – HMMER_DAEMON.config file for Windows:**

| This example shows the required options and profile database specification for
| Windows.

```
| =
| DAEMON_PORT=4098
| MAX_PENDING_REQUESTS=10
| DAEMON_LOGFILE_DIR=.\
| Q_SEQ_DIR_PATH=.\
| HMMER_OUT_DIR_PATH=.\
| HMMPFAM_PATH=c:\hmmer\bin\hmmpfam.exe
| HMMSEARCH_PATH=c:\hmmer\bin\hmmsearch.exe
| H2X_PATH=.\db2h2x.exe
| myhmms=c:\hmmer\tutorial\myhmms
| globin=c:\hmmer\tutorial\globin.hmm
| pfamseq=c:\hmmer\pfam\pfamseq
```

| **Related tasks:**

- "Configuring the HMMER daemon" on page 255

Starting the HMMER daemon

Starting the HMMER daemon is part of the larger task of adding HMMER data sources to a federated server. Before you can access HMMER data sources, you must start the HMMER daemon.

Prerequisites:

Before you start the HMMER daemon, you must have write access to all paths listed under the `DAEMON_LOGFILE_DIR`, `HMMER_OUT_DIR_PATH`, and `Q_SEQ_DIR_PATH` entries in the configuration file.

Procedure:

To start the HMMER daemon on a UNIX server:

1. Open the directory where the daemon executable file is located.
2. Issue the `db2hmmer_daemon` command:
   - If you did not change the name of the daemon configuration file and the configuration file is in the same directory as the daemon executable file, type the following command at the command line:

```
db2hmmer_daemon
```

- If you changed the name of the daemon configuration file or if the daemon configuration file is not in the same directory as the daemon executable file, you must use the -c option on the wrapper daemon command to point the daemon executable to the new name or location.

  For example, the following command causes the wrapper daemon to look for the daemon configuration information in a file called `HMMER_D.config` in the subdirectory `cfg`.

  ```
  db2hmmer_daemon -c cfg/HMMER_D.config
  ```

The executable file starts a new process in which the HMMER daemon runs.

To stop the daemon on a UNIX server:

1. List the process ID of the `db2hmmer_daemon` by using the following UNIX command:

   ```
   ps -ef | grep db2hmmer
   ```

2. Use the process ID to stop the daemon. Use this command:

   ```
   kill nnnn
   ```

   where `nnnn` is the process ID of the `db2hmmer_daemon`.

To start the HMMER daemon on a Windows server:

1. Open the directory where the daemon executable file is located.
2. Issue the `db2hmmer_daemon` command with the parameters that you need. For example, to install the daemon service with debugging turned on and start the daemon issue these commands:

   ```
   db2hmmer_daemon -a install -d 2
   ```

   ```
   db2hmmer_daemon -a start
   ```

To stop the daemon, use the following Windows command:

```
db2hmmer_daemon -a stop
```

The next task in this sequence of tasks is registering the HMMER wrapper.

Related tasks:

- "Configuring the HMMER daemon" on page 255
- "Registering the HMMER wrapper" on page 262
- "Adding HMMER data sources to a federated server" on page 254

Related reference:

- "db2hmmer_daemon command - syntax and examples" on page 260

db2hmmer_daemon command - syntax and examples

The `db2hmmer_daemon` command can be used on UNIX and Windows servers. Some of the arguments listed in the syntax can be used only on Windows servers.

The syntax for the `db2hmmer_daemon` command is:

```
db2hmmer_daemon -a action -c config_file -d debug_level
    -u user_id -p password
```

-a *action*
: Performs the specified activity. Valid actions are *status*, *install*, *start*, *stop*, and *remove*.

You can specify this argument only on Windows servers.

-c *config_file*
Instructs the daemon service to use the specified configuration file instead of the default configuration file. If you do not specify the configuration file, the daemon searches for the `HMMER_DAEMON.config` file in the directory where the daemon executable files are installed. You can use this option with the *install* and *start* actions.

You can specify this argument on UNIX and Windows servers.

-d *debug_level*
Sets the daemon service debug level to the specified value. The valid values are 1, 2, or 3. You can use this option with the *install* and *start* actions.

You can specify this argument on UNIX and Windows servers.

-u *user_id*
Sets the service to run under the specified user ID. You can use this option with the *install* action.

You can specify this argument only on Windows servers.

-p *password*
Specifies the password for the specified user ID. The password is valid and required only when you specify the -u option. If the -p option is not specified when you set the -u option, the program prompts you for the password. You can use this option with the *install* action.

You can specify this argument only on Windows servers.

The options that are specified with the *start* action affect only the current run of the daemon, and override the values that are specified with the *install* action.

Examples:

The following examples show daemon actions on Windows. These examples assume that the `HMMER_DAEMON.config` file is in the same directory as `db2hmmer_daemon.exe`.

- To check the status of the daemon:
```
db2hmmer_daemon -a status
```
- To install the daemon service with debugging turned on:
```
db2hmmer_daemon -a install -d 2
```
- To start the daemon:
```
db2hmmer_daemon -a start
```
- To stop the daemon:
```
db2hmmer_daemon -a stop
```
- To remove or uninstall the daemon service:
```
db2hmmer_daemon -a remove
```

Related tasks:
- "Starting the HMMER daemon" on page 259

Registering the HMMER wrapper

Registering the HMMER wrapper is part of the larger task of adding HMMER data sources to a federated server.

You must register a wrapper to access HMMER data sources. Wrappers are used by federated servers to communicate with and retrieve data from data sources. Wrappers are implemented as a set of library files.

Procedure:

To register a wrapper, issue the CREATE WRAPPER statement with the name of the wrapper and the name of the wrapper library file.

For example, to register a wrapper with the name `hmmer_wrapper` on the federated server that uses the AIX operating system, issue the following statement:

```
CREATE WRAPPER hmmer_wrapper LIBRARY 'libdb2lshmmer.a';
```

The name of the wrapper library file that you specify depends on the operating system of the federated server. See the list of HMMER wrapper library files for the correct name to specify in the CREATE WRAPPER statement.

The next task in this sequence of tasks is registering the server definitions for the HMMER wrapper.

Related tasks:
- "Registering the server definition for a HMMER data source" on page 263

Related reference:
- "HMMER wrapper library files" on page 262
- "CREATE WRAPPER statement" in the *SQL Reference, Volume 2*

HMMER wrapper library files

The following table lists the directory paths and library file names for the HMMER wrapper.

When you install DB2 Information Integrator, 3 library files are added to the directory path listed in the table. For example, if the federated server is running on AIX, the wrapper library files added to the directory path are `libdb2lshmmer.a`, `libdb2lshmmerF.a`, and `libdb2lshmmerU.a`.

When you register a wrapper, specify only the library file name that is listed in the table.

Table 70. HMMER wrapper library locations and file names

Operating system	Directory path	Wrapper library file
AIX	/usr/opt/db2_08_01/lib/	libdb2lshmmer.a
Linux	/opt/IBM/db2/V8.1/lib	libdb2lshmmer.so
Solaris	/opt/IBM/db2/V8.1/lib	libdb2lshmmer.so
Windows	%DB2PATH%\bin	db2lshmmer.dll

%DB2PATH% is the environment variable that is used to specify the directory path where DB2 Information Integrator is installed on Windows. The default Windows directory path is `C:\Program Files\IBM\SQLLIB`.

Related tasks:

- "Registering the HMMER wrapper" on page 262

Registering the server definition for a HMMER data source

Registering server definitions for HMMER data sources is part of the larger task of adding HMMER data source to a federated server.

After you register the wrapper, you must register a corresponding server definition.

Procedure:

To register the HMMER server definition, issue the CREATE SERVER statement.

For example:

```
CREATE SERVER server_name TYPE HMMER_search_type
   VERSION version WRAPPER wrapper_name
   OPTIONS (NODE 'node_name', DAEMON_PORT 'port_number')
```

You must register a definition for each server that you want to run a HMMER search on.

The next task in this sequence of tasks is registering nicknames for HMMER data sources.

Related tasks:

- "Registering nicknames for HMMER data sources" on page 265

Related reference:

- Appendix D, "Server options for federated systems," on page 575
- "CREATE SERVER statement - examples for HMMER wrapper" on page 263

CREATE SERVER statement - examples for HMMER wrapper

This topic provides an example that shows you how to use the CREATE SERVER statement to register server definitions for the HMMER wrapper.

To register the server definition `hmmpfam_server` for searches that use the `hmmpfam` program, issue the following statement:

```
CREATE SERVER hmmpfam_server
 TYPE pfam
   VERSION 2.2
   WRAPPER hmmer_wrapper
   OPTIONS (NODE 'someserver.someschool.edu', DAEMON_PORT '4098')
```

hmmpfam_server
: A name that you assign to the HMMER server definition. This name must be unique.

TYPE *pfam*
: The type of search that the server definition supports. The types that you can specify are PFAM (for hmmpfam) or SEARCH (for hmmsearch).

VERSION *2.2*
: The version of the `hmmpfam` or `hmmsearch` executable file that you are using. The supported versions are HMMER 2.2g (or later).

WRAPPER *hmmer_wrapper*
: The name that you specified in the CREATE WRAPPER statement.

NODE *'someserver.someschool.edu'*
: The host name or the IP address of the server on which the HMMER daemon process runs.

  Although the node name is specified as an option in the CREATE SERVER statement, it is required for HMMER data sources.

DAEMON_PORT *'4098'*
: The port number on which the daemon listens for HMMER job requests. The port number must be the same number specified in the DAEMON_PORT option of the daemon configuration file. The default is 4098.

Additional server options:

When you create the server definition, you can specify additional server options in the CREATE SERVER statement.

PROCESSORS
: The number of processors that the HMMER program uses. This option is equivalent to the --cpu option of the **hmmpfam** and **hmmsearch** commands. Example: `PROCESSORS '2'`.

HMMPFAM_OPTIONS
: Use this server option to pass options to the **hmmpfam** command that cannot be specified in a predicate. For example: `HMMPFAM_OPTIONS '--null2 --pvm'`. In this example, `hmmpfam` will always run with the two additional options `--null2` and `--pvm` whenever a query is run against this server. The HMMPFAM_OPTIONS option is only valid with servers specified as type PFAM.

HMMSEARCH_OPTIONS
: Use this server option to pass options to the **hmmsearch** command that cannot be specified in a predicate. For example: `HMMSEARCH_OPTIONS '--null2 --pvm'`. In this example, `hmmsearch` will always run with the two additional options `--null2` and `--pvm` whenever a query is run against this server. The HMMSEARCH_OPTIONS option is only valid with servers specified as type SEARCH.

Related tasks:
- "Registering the server definition for a HMMER data source" on page 263

Related reference:
- "CREATE SERVER statement" in the *SQL Reference, Volume 2*
- Appendix D, "Server options for federated systems," on page 575

Registering nicknames for HMMER data sources

Registering nicknames for HMMER data sources is part of the larger task of adding HMMER data source to a federated server.

After you register a server definition, you must register a corresponding nickname. When you refer to a HMMER data source in a query, you use nicknames.

Procedure:

To register a HMMER nickname, issue the CREATE NICKNAME statement. For example:

```
CREATE NICKNAME nickname
   FOR SERVER server_name
   OPTIONS(DATASOURCE 'data_source')
```

You must define a separate nickname for each profile database that you want to query. The `data_source` name must match an existing `data_source_name` in the `HMMER_DAEMON.config` file on the HMMER server.

| When you create a nickname for a HMMER database, a set of input and output
| fixed columns for the profile database are registered in the federated database
| system catalog.

There are no further tasks in this sequence of tasks.

Related tasks:
- "Specifying nickname columns for a nonrelational data source" on page 65
- "Configuring the HMMER daemon" on page 255
- "Construct new HMMER queries with samples" on page 271
- "Adding HMMER data sources to a federated server" on page 254

Related reference:
- "Fixed columns for HMMER nicknames" on page 265
- "HMMER data source – complete example" on page 270
- "CREATE NICKNAME statement - Example for HMMER wrapper" on page 269
- "Fixed columns for HMMER nicknames" on page 265

Fixed columns for HMMER nicknames

| When you issue the CREATE NICKNAME statement for a HMMER data source, a
| set of fixed input column and fixed output columns are automatically created with
| the nickname. If you want to change the default data type that is assigned to a
| fixed column, you can specify the column name and data type in the CREATE
| NICKNAME statement. For example, to limit the AlignmentConsensus column
| output to no more than the first 100 characters, you issue the following statement:

```
| CREATE NICKNAME nucleo1 (AlignmentConsensus VARCHAR(100))
|    FOR SERVER searchtest
|    OPTIONS(DATASOURCE 'nucleo1', TIMEOUT '1');
```

You can reference the fixed columns in SQL queries as part of the nickname definition. There are two types of fixed columns, input and output.

Fixed input columns for HMMER nicknames

The fixed input columns are specified in the WHERE clause. Input columns are used as parameter-passing predicates in SQL queries. They pass standard HMMER switches to either hmmpfam or hmmsearch. HMMER then runs on the specified data source using these switches. Fixed input columns can also be referenced in the query SELECT list and are returned as part of the results table.

Input fixed columns for servers of type PFAM:

The following table lists the fixed columns that you can use in the WHERE clause.

Table 71. Fixed input columns for servers of type PFAM

Name	Data type	Description	Operator	Switches	Returned Value
HmmQSeq	varchar (32000)	Input gene sequence that is used to search	=		Same as the input value that you specify. This column is required.
ModelEValue	double	Estimated e-value	<	-E *n*	See output.
ModelScore	double	Raw score	>	-T *n*	See output.
DBSize	integer	Calculate e-values as if the database had 'n' gene sequences	=	-Z *n*	Same as the input value that you specify. Uses hmmpfam default if not specified.
CutMode	char(2)	Cutoff mode; can be ga, tc or nc (case sensitive)	=	--cut_ga --cut_tc --cut_nc	Same as the input value that you specify. NULL if not specified.
DomainScore	double	Domain score	>	--domT *n*	See output.
DomainEValue	double	Domain e-value	<	--domE *n*	See output.
ForwardAlgorithm	char	Use Forward algorithm rather than Viterbi; value can be 'Y' or 'N'	=	--forward	Same as the input value that you specify. 'N' is the default.

Input fixed columns for servers of type SEARCH:

The following table lists the fixed columns that you can use in the WHERE clause.

Table 72. Fixed input columns for servers of type SEARCH

Name	Data type	Description	Operator	Options	Returned Value
Model	varchar (32000)	Name of the HMM profile file used in the search. The name must be one of the data source names listed in the database specification entry in the `HMMER_DAEMON.config` file.	=		Same as the input value that you specify. This column is required.
SequenceEValue	double	Estimated e-value	<	-E *n*	See output.
SequenceScore	double	Raw score	>	-T *n*	See output.
DBSize	integer	Calculate e-values as if the database had 'n' gene sequences	=	-Z *n*	Same as the input value that you specify. Uses hmmpfam default if not specified.
CutMode	char(2)	Cutoff mode; can be ga, tc or nc (case sensitive)	=	--cut_ga --cut_tc --cut_nc	Same as the input value that you specify. NULL if not specified.
DomainScore	double	Domain score	>	--domT *n*	See output.
DomainEValue	double	Domain e-value	<	--domE *n*	See output.
ForwardAlgorithm	char	Use Forward algorithm rather than Viterbi; value can be 'Y' or 'N'	=	--forward	Same as the input value that you specify. 'N' is the default.

Fixed output columns for HMMER nicknames

You can specify any of the fixed output columns in the SELECT list. You can also specify fixed output columns in the WHERE clause (as predicates).

Fixed output columns for PFAM:

The following table lists the fixed columns that are returned as output for PFAM.

Table 73. Fixed output columns for PFAM

Name	Data type	Description
Model	varchar(32)	Name of model.
ModelDescription	varchar(64)	Text description of model.

Table 73. Fixed output columns for PFAM (continued)

Name	Data type	Description
ModelScore	double	Raw score (″bit score″).
ModelEValue	double	Estimated e-value.
ModelHits	integer	Number of domains hit within the model.
DomainNumber	integer	Specific domain (within one model).
SequenceFrom	integer	Starting point of gene sequence.
SequenceFromGlobal	char	′Y′ if the alignment starts at the beginning of the gene sequence.
HmmFrom	integer	Starting point of consensus model.
HmmFromGlobal	char	′Y′ if the alignment starts at the beginning of the consensus model.
HmmTo	integer	Ending point in consensus model.
HmmToGlobal	char	′Y′ if the alignment ends at the end of the consensus model.
DomainScore	double	Raw score (″bit score″) for the isolated domain.
DomainEValue	double	Expected value for the isolated domain.
AlignmentConsensus	varchar(32000)	The HMM consensus. The amino acid shown for the consensus is the highest probability amino acid at that position according to the HMM, not necessarily the highest scoring amino acid.
AlignmentExactMatch	varchar(32000)	Matches the highest probability residue in the HMM.
AlignmentSubSequence	varchar(32000)	Shows the gene sequence itself.

Fixed output columns for SEARCH:

The following table lists the fixed columns that are returned as output for SEARCH.

Table 74. Fixed output columns for SEARCH

Name	Data type	Description
Sequence	varchar(32)	The sequence identifier.
SequenceDescription	varchar(64)	Text description of the sequence.
SequenceScore	double	Raw score (″bit score″).
SequenceEValue	double	Estimated e-value.
SequenceHits	integer	Number of domains hit within the sequence.
DomainNumber	integer	Specific domain (within one sequence).
SequenceFrom	integer	Starting point of gene sequence.
SequenceFromGlobal	char	′Y′ if the alignment starts at the beginning of the gene sequence.
HmmFrom	integer	Starting point of consensus model.
HmmFromGlobal	char	′Y′ if the alignment starts at the beginning of the consensus model.

Table 74. Fixed output columns for SEARCH (continued)

Name	Data type	Description
HmmTo	integer	Ending point in consensus model.
HmmToGlobal	char	'Y' if the alignment ends at the end of the consensus model.
DomainScore	double	Raw score ("bit score") for the isolated domain.
DomainEValue	double	Expected value for the isolated domain.
AlignmentConsensus	varchar(32000)	The HMM consensus. The amino acid shown for the consensus is the highest probability amino acid at that position according to the HMM, not necessarily the highest scoring amino acid.
AlignmentExactMatch	varchar(32000)	Matches the highest probability residue in the HMM.
AlignmentSubSequence	varchar(32000)	Shows the gene sequence itself.

Related tasks:

- "Registering nicknames for HMMER data sources" on page 265

Related reference:

- "HMMER data source – complete example" on page 270
- "CREATE NICKNAME statement - Example for HMMER wrapper" on page 269

CREATE NICKNAME statement - Example for HMMER wrapper

This topic provides an example that shows you how to use the CREATE NICKNAME statement to register a nickname for HMMER data sources.

To register the nickname `hmmpfam_nickname` that uses the server definition `hmmpfam_server`, you issue the following statement:

```
CREATE NICKNAME hmmpfam_nickname
   FOR SERVER hmmpfam_server
   OPTIONS(DATASOURCE 'myhmms',TIMEOUT '30')
```

hmmpfam_nickname
A name that you assign to the nickname. This name must be unique.

SERVER *hmmpfam_server*
The name of the server definition that you want this nickname to be associated with.

DATASOURCE *'myhmms'*
The name of the database that you will run HMMER searches on. This database must be listed in the HMMER daemon configuration file.

Although the data source is specified as an option in the CREATE NICKNAME statement, it is required for HMMER data sources.

TIMEOUT *'30'*
The maximum time, in minutes, that the wrapper waits for results from the daemon. The default is 60 minutes.

Related tasks:

- “Registering nicknames for HMMER data sources” on page 265

Related reference:

- Appendix F, “Nickname options for federated systems,” on page 593

HMMER data source – complete example

This topic provides an example of all the SQL statements that you need to issue to add HMMER data sources to a federated server. This example also shows a query that is run using the nickname created in the SQL statements.

SQL statements for HMMER data sources must contain special input predicates that are used to pass standard HMMER options to the program executable file. To be valid, every query passed to the HMMER wrapper must contain at least the HmmQSeq input predicate (for TYPE PFAM) or model predicate (for TYPE SEARCH). All other predicates are optional.

To construct a HMMER query against a nickname, specify input columns in the WHERE clause and output columns in the SELECT list.

Example for the hmmpfam program:

This example creates a wrapper, server definition, and nickname on an AIX federated server for the hmmpfam program. This is example also runs a query that uses a string literal for the search sequence.

```
CREATE WRAPPER hmmer_wrapper
   LIBRARY 'libdb2lshmmer.a';

CREATE SERVER hmmpfam_server
   TYPE pfam VERSION 2.2
   WRAPPER hmmer_wrapper
   OPTIONS(NODE 'HMMERserv.MyCompany.com');

CREATE NICKNAME hmmpfam_nickname
   FOR SERVER hmmpfam_server
   OPTIONS(DATASOURCE 'myhmms', TIMEOUT '1');

-- Run the 7LES_DROME gene sequence on the hmmpfam_nickname
SELECT Model, substr(ModelDescription,1,50) as ModelDescription,
      ModelScore, ModelEValue, ModelHits, DomainNumber,
      SequenceFrom, SequenceTo, SequenceFromGlobal, SequenceToGlobal,
      HmmFrom, HmmTo, HmmFromGlobal, HmmToGlobal,
      DomainScore, DomainEValue,
      length(HmmQSeq)           as "length(HmmQSeq)",
      length(AlignmentConsensus) as "length(AConsensus)",
      length(AlignmentMatch)     as "length(AMatch)",
      length(AlignmentSubSeq)    as "length(ASubSeq)",
      substr(HmmQSeq,1,64)            as HmmQSeq,
      substr(AlignmentConsensus,1,64) as AlignmentConsensus,
      substr(AlignmentMatch,    1,64) as AlignmentMatch,
      substr(AlignmentSubSeq,   1,64) as AlignmentSubSeq
FROM hmmpfam_nickname
WHERE HmmQSeq =
     'MTMFWQQNVDHQSDEQDKQAKGAAPTKRLNISFNVKIAVNVNTKMTTTHINQQAPGTSS...';
```

Example for the hmmsearch program:

This example creates a wrapper, server definition, and nickname on a Windows federated server for the hmmsearch program.

```
CREATE WRAPPER hmmer_wrapper
   LIBRARY 'db2lshmmer.dll'
   OPTIONS(DB2_FENCED 'Y');

CREATE SERVER hmmsearch_serv
   TYPE search VERSION 2.2
   WRAPPER hmmer_wrapper
   OPTIONS(NODE 'localhost');

CREATE NICKNAME artemia
   FOR SERVER hmmsearch_server
   OPTIONS(DATASOURCE 'artemia', TIMEOUT '1');

SELECT Model, Sequence, substr(SequenceDescription,1,50)
   as SequenceDescription, SequenceScore, SequenceEValue,
   SequenceHits, DomainNumber, SequenceFrom,
   SequenceTo, SequenceFromGlobal, SequenceToGlobal,
   HmmFrom, HmmTo, HmmFromGlobal, HmmToGlobal, DomainScore,
   DomainEValue,
   length(AlignmentConsensus)  as "length(AConsensus)",
   length(AlignmentMatch)      as "length(AMatch)",
   length(AlignmentSubSeq)     as "length(ASubSeq)",
   substr(AlignmentConsensus,1,200) as AlignmentConsensus,
   substr(AlignmentMatch, 1,200)    as AlignmentMatch,
   substr(AlignmentSubSeq, 1,200)   as AlignmentSubSeq
FROM artemia
WHERE Model = 'globin' and DomainScore > 50;
```

Related tasks:

- "Registering nicknames for HMMER data sources" on page 265
- "Construct new HMMER queries with samples" on page 271
- "Adding HMMER data sources to a federated server" on page 254

Construct new HMMER queries with samples

The following sample HMMER queries show how to construct queries for HMMER data sources.

Procedure:

To run queries, use the following examples as a guide.

In these queries, the nickname is a name that describes the type of HMMER search and the data source. Some examples also show how to use the HMMER wrapper with other data sources.

Query 1.

```
SELECT Model, ModelScore, ModelEValue, DomainNumber, DomainScore, DomainEvalue
   FROM hmmpfam_nickname
   WHERE HmmQSeq = 'MTMFWQQNVDHQSDEQDKQAKGAAPTKRLNISFNVKIAVNVNTKMTTTHINQ...'
```

When this SQL statement runs, the wrapper uses the indicated sequence and the HMM database defined by the nickname to run the hmmpfam program. The wrapper returns the columns that are listed in the SELECT statement.

Query 2.

```
SELECT Model, ModelScore, ModelEValue
   FROM hmmpfam_nickname
   WHERE HmmQSeq = 'MTMFWQQNVDHQSDEQDKQAKGAAPTKRLNISFNVKIAVNVNTKMTTTHINQ...'
   AND ModelScore > 0
```

When this SQL statement runs, the wrapper performs an hmmpfam search of hmmpfam_nickname that uses the indicated gene sequence. In addition, the wrapper passes the -T 0 option to the hmmpfam command. This option is from the list of fixed columns for HMMER nicknames. The wrapper returns the three columns that are listed after SELECT.

Query 3.

```
SELECT Model, DomainNumber, DomainScore, DomainEValue
   FROM hmmpfam_nickname
   WHERE HmmQSeq = 'MTMFWQQNVDHQSDEQDKQAKGAAPTKRLNISFNVKIAVNVNTKMTTTHINQ...'
   AND ModelEValue < 1
   ORDER BY DomainScore DESC
```

When this SQL statement runs, the wrapper performs an hmmpfam search of hmmpfam_nickname that uses the indicated gene sequence. In addition, the wrapper passes the -E 1 option to the hmmpfam command. This option is from the list of fixed columns for HMMER nicknames. The wrapper returns the four columns that are listed after SELECT and sorts the result from highest to lowest by the DomainScore.

Query 4.

```
CREATE WRAPPER hmmer_wrapper
   LIBRARY 'db2lshmmer.dll';

CREATE SERVER hmmsearch_server
   TYPE search VERSION 2.2
   WRAPPER hmmer_wrapper
   OPTIONS(NODE 'HMMERserv.MyCompany.com');

CREATE NICKNAME artemia_nickname
   FOR SERVER hmmsearch_server
   OPTIONS(DATASOURCE 'artemia', TIMEOUT '1');

SELECT Model, Sequence, substr(SequenceDescription,1,50)
   as SequenceDescription, SequenceScore, SequenceEValue,
   SequenceHits, DomainNumber, SequenceFrom,
   SequenceTo, SequenceFromGlobal, SequenceToGlobal,
   HmmFrom, HmmTo, HmmFromGlobal, HmmToGlobal, DomainScore,
   DomainEValue,
   length(AlignmentConsensus)  as "length(AConsensus)",
   length(AlignmentMatch)      as "length(AMatch)",
   length(AlignmentSubSeq)     as "length(ASubSeq)",
   substr(AlignmentConsensus,1,200) as AlignmentConsensus,
   substr(AlignmentMatch, 1,200)    as AlignmentMatch,
   substr(AlignmentSubSeq, 1,200)   as AlignmentSubSeq
FROM artemia_nickname
WHERE Model = 'globin' and DomainScore > 50;
```

When this SQL statement runs, the wrapper runs hmmsearch against the sequence file artemia, using the HMM specified by globin. The rows with a DomainScore greater than 50 are returned, because the wrapper passes the --domT 50 option to the **hmmsearch** command. The wrapper returns the columns specified after SELECT. Column values that are longer than 200 characters are truncated. Only the first 200 characters in these columns are returned.

Related tasks:

- "Adding HMMER data sources to a federated server" on page 254

Related reference:

- "Fixed columns for HMMER nicknames" on page 265

- “HMMER data source – complete example” on page 270
- “Messages for the HMMER wrapper” on page 273

Messages for the HMMER wrapper

For the HMMER wrapper to work, you must specify a query that contains a predicate on the HmmQSeq column. When you query a fragment that lacks a predicate on the HmmQSeq column, you get an error.

This section lists and describes messages that you might encounter when you work with the HMMER wrapper.

Table 75. HMMER wrapper messages

Error Code	Message	Explanation
SQL0142N	The SQL statement is not supported.	The SQL query submitted to DB2 could not be processed by the wrapper. Add the required predicate and resubmit. Verify that the operator used in a predicate is valid for that column. See the fixed columns for HMMER nicknames.
SQL1822N	Unexpected error code ″Unspecified Error″ received from data source ″Hmmer wrapper″. Associated text and tokens are ″Unable to resolve NODE host name″.	The TCP/IP NODE name specified in CREATE SERVER is invalid.
SQL1822N	Unexpected error code ″Unspecified Error″ received from data source ″Hmmer wrapper″. Associated text and tokens are ″Unable to connect to daemon″.	Either the hmmer_daemon program is not currently running on the target node, or the DAEMON_PORT specified in the CREATE SERVER command does not match the DAEMON_PORT value specified in daemon configuration file HMMER_DAEMON.config.
SQL1822N	Unexpected error code ″Unspecified Error″ received from data source ″Hmmer wrapper″. Associated text and tokens are ″Unknown error from the hmmer daemon″.	The DATASOURCE name specified in the CREATE NICKNAME statement cannot match any of the profile database names listed in the daemon configuration file HMMER_DAEMON.config.
SQL1822N	Unexpected error code ″Unspecified Error″ received from data source ″Hmmer wrapper″. Associated text and tokens are ″FATAL: No such option ″--cut_TC″.	The CutMode predicate must be specified in lowercase. Example: WHERE CutMode = ′tc′

Related concepts:
- “Introduction to messages” in the *Message Reference Volume 1*

Related reference:
- “SQLSTATE messages” in the *Message Reference Volume 2*

Chapter 15. Configuring access to Informix data sources

This chapter explains how to configure your federated server to access data that is stored in Informix data sources. You can configure access to Informix data sources by using the DB2 Control Center or by issuing SQL statements.

This chapter lists the tasks that you need to perform when you configure the federated server, contains examples of the SQL statements that you need, and provides tuning and troubleshooting information for configuring the federated server.

Adding Informix to a federated server

Adding Informix data sources to federated servers

To configure the federated server to access Informix data sources, you must provide the federated server with information about the data sources and objects that you want to access.

| You can configure the federated server to access Informix data sources by using the
| DB2 Control Center or the DB2 command line. The DB2 Control Center includes a
| wizard to guide you through the steps required to configure the federated server.

Prerequisites:

- A DB2 server that is configured for federation.
- A federated database must exist on the federated server
- The Informix Client SDK software installed and configured on the federated server.
- On AIX federated servers, the AIX Base Application Development Math Library. You can determine if the Library is installed by issuing the AIX command **lslpp -l bos.adt.libm**.

Procedure:

To add Informix data sources to a federated server:

1. Set up and test the Informix client configuration file.
2. Set the Informix environment variables.
3. Register the wrapper.
4. Register the server definition.
5. Create the user mappings.
6. Test the connection to the Informix server.
7. Register nicknames for Informix tables, views, and synonyms.

Related concepts:

- "Fast track to configuring your data sources" on page 55

Related tasks:

- "Checking the FEDERATED parameter" on page 42

- "Creating a federated database" on page 51
- "Setting up and testing the Informix client configuration file" on page 276
- "Registering the Informix wrapper" on page 280
- "Registering the server definitions for an Informix data source" on page 282
- "Creating the user mapping for an Informix data source" on page 284
- "Testing the connection to the Informix server" on page 286
- "Registering nicknames for Informix tables, views, and synonyms" on page 286
- "Tuning and troubleshooting the configuration to Informix data sources" on page 288
- "Setting the Informix environment variables" on page 277
- "Checking the setup of the federated server" on page 37

Related reference:

- "Supported operating systems for DB2 Information Integrator (32-bit)" in the *IBM DB2 Information Integrator Installation Guide for Linux, UNIX, and Windows*
- "Supported operating systems for DB2 Information Integrator (64-bit)" in the *IBM DB2 Information Integrator Installation Guide for Linux, UNIX, and Windows*

Setting up and testing the Informix client configuration file

Setting up and testing the Informix client configuration file is part of the larger task of adding Informix data sources to federated servers.

The client configuration file is used to connect to Informix, using the client libraries that are installed on the federated server. This file specifies the location of each Informix database server and type of connection (protocol) for the database server.

The default location of the client configuration file depends on the operating system that is used by the federated server.

- On UNIX operating systems, the default location and name of the file is `$INFORMIXDIR/etc/sqlhosts`. The `sqlhosts` file is installed with the Informix client SDK.
- On Windows operating systems, the default location of the `sqlhosts` registry is the local computer.

The format of `sqlhosts` is described in the *Administrator's Guide for Informix Dynamic Server*.

Procedure:

To set up and test the Informix client configuration file:

1. Configure the Informix Client SDK.
   - On UNIX, you can configure the Informix Client SDK by editing the `sqlhosts` file. You can also copy the `sqlhosts` file from another system that has Informix Connect or Informix Client SDK installed.
   - On Windows, you can configure the Informix Client SDK with the Informix Setnet32 utility. The Setnet32 utility sets up the `sqlhosts` registry.
2. Verify the location of the `sqlhosts` file or registry.
   - On UNIX operating systems, the `sqlhosts` file is located in the `$INFORMIXDIR/etc/` directory.

- On Windows operating systems, the sqlhosts information is kept in the following key in the Windows registry:

  ```
  HKEY_LOCAL_MACHINE\SOFTWARE\INFORMIX\SQLHOSTS
  ```
3. If the `sqlhosts` file or registry is not in the default location, set the environment variable INFORMIXSQLHOSTS.
   - On UNIX operating systems, set the environment variable INFORMIXSQLHOSTS to the fully-qualified name of the `sqlhosts` file.
   - On Windows operating systems, set the environment variable INFORMIXSQLHOSTS to the name of the Windows computer that stores the registry.
4. Test the connection to ensure that the client software is able to connect to the Informix server. If the Informix **dbaccess** tool is on the federated server, use this tool to test the connection. Otherwise, run the Informix demo program to test the client setup.

The next task in this sequence of tasks is setting the Informix environment variables.

Related tasks:
- "Registering the Informix wrapper" on page 280
- "Tuning and troubleshooting the configuration to Informix data sources" on page 288

Setting the Informix environment variables

Setting the Informix environment variables is part of the larger task of adding Informix to a federated server.

When you install DB2 Information Integrator, the installation process attempts to set the Informix environment variables in the `db2dj.ini` file.

The environment variables will not be set in the `db2dj.ini` file if you:
- Install the Informix client software after the DB2 federated server is set up.
- Have not installed the Informix client software.

The valid environment variables for Informix are:
- INFORMIXDIR
- INFORMIXSERVER
- INFORMIXSQLHOSTS (optional)
- CLIENT_LOCALE (optional)
- DB_LOCALE (optional)
- DBNLS (optional)

The optional environment variables must be set manually.

Prerequisites:

A federated server that is properly set up to access your data sources. This includes the installation and configuration of any required software, such as the data source client software.

Restrictions:

See the topic: Restrictions for the db2dj.ini file

Procedure:

To set the required environment variables automatically:

1. Install and configure the client software on the DB2 federated server (if necessary).
2. Set the required environment variables. You can set the environment variables automatically by running the DB2 server installation again and specify the Typical installation option.

To manually set the environment variables:

1. Edit the `db2dj.ini` file.
   - On federated servers running Windows, this file is located in the `%DB2PATH%\cfg` directory.
   - On federated servers running UNIX, this file is located in the `sqllib/cfg` directory.

   The `db2dj.ini` file contains configuration information about the Informix client software installed on your federated server. If the file does not exist, you can create a new file with the name `db2dj.ini` using any text editor. In the `db2dj.ini` file you must specify the fully qualified path for the variables, otherwise you will encounter errors.
2. Set the following environment variables as necessary:

   INFORMIXDIR

   Set the INFORMIXDIR environment variable to the directory path where the Informix Client SDK software is installed. For example:

   On federated servers running Windows, set the path to:

   ```
   INFORMIXDIR=C:\informix\csdk
   ```

   On federated servers running UNIX, set the path to:

   ```
   INFORMIXDIR=/informix/csdk
   ```

   INFORMIXSERVER

   This variable identifies the name of the default Informix server. This setting must be a valid entry in the `sqlhosts` file (UNIX) or the SQLHOSTS registry key (Windows). To get a value for INFORMIXSERVER, read the `sqlhosts` file. Select one of the *dbservername* values. The *dbservername* is the first value in each entry in the `sqlhosts`. For example:

   ```
   INFORMIXSERVER=inf93
   ```

   Requirement: Although the Informix wrapper does not use the value of this variable, the Informix client requires that this variable be set. The wrapper uses the value of the **NODE** server option, which specifies the Informix database server that you want to access.

   INFORMIXSQLHOSTS

   If you are using the default path for the Informix `sqlhosts` file, you do not need to set this variable. However, if you are using some other path for the Informix `sqlhosts` file, then you need to set this variable to the full path name where the Informix `sqlhosts` file resides.

- On federated servers running UNIX, the default path is `$INFORMIXDIR/etc`.
- On federated servers running Windows, if the SQLHOSTS registry key does not reside on the local computer, then the INFORMIXSQLHOSTS is the name of the Windows computer that stores the registry.

A UNIX example of setting this variable to another path is:

```
INFORMIXSQLHOSTS=/informix/csdk/etc/my_sqlhosts
```

3. To ensure that the environment variables are set on the federated server, recycle the DB2 instance. Issue the following commands to recycle the DB2 instance:

```
db2stop
db2start
```

Setting up Informix code page conversion:

For Informix code page conversion, you can set the following optional environment variables:
- CLIENT_LOCALE
- DB_LOCALE
- DBNLS

Each time that the Informix wrapper connects to an Informix data source, the wrapper determines which code page value to use for that connection. If the Informix environment variable CLIENT_LOCALE is set in the `db2dj.ini` file on the federated server, then the wrapper uses the value in the `db2dj.ini` file.

You can obtain the list of valid Informix locales by issuing the **glfiles** command on the Informix server. Refer to the *Informix Guide to GLS Functionality* for more information about code page conversions.

The Informix code page environment variables are:

CLIENT_LOCALE

Set the CLIENT_LOCALE environment variable to the Informix locale that you want to use. If CLIENT_LOCALE is not set, the wrapper determines the code page and territory of the federated database. The wrapper sets the CLIENT_LOCALE variable to the closest matching Informix locale. If there is no matching Informix locale, the wrapper sets the CLIENT_LOCALE variable to the en_us.8859-1 locale for UNIX systems and to the en_us.CP1252 locale for Windows systems. You can see a list of locale names by using the Informix **glfiles** command.

```
CLIENT_LOCALE=Informix_client_locale_value
```

DB_LOCALE

Set this environment variable if the Informix database uses a different code page than your client locale, and you want Informix to perform conversions between the two code pages. Set Informix environment variable DB_LOCALE to the name of the Informix database locale, for example:

```
DB_LOCALE=Informix_db_locale_value
```

DBNLS

To have Informix verify that the DB_LOCALE setting matches the actual locale of the Informix database, set this Informix environment variable to 1.

```
DBNLS=1
```

Code page environment variables-example:

Suppose that the Informix database uses a different code page than your client locale and you want Informix to perform conversions between the two code pages. You need to set Informix environment variable DB_LOCALE to the name of the Informix database locale. You set this variable in the db2dj.ini file on the federated server.

If you want Informix to verify that the DB_LOCALE setting matches the actual locale of the Informix database, then you need to set the Informix environment variable DBNLS to 1. You set this variable in the db2dj.ini file on the federated server.

If you access a data source that contains data that uses the Chinese code page GB 18030, your federated database must use the UTF-8 code page. The Informix wrapper sets Informix environment variables to:

```
CLIENT_LOCALE=zh_cn.UTF8
GL_USEGLU=1
```

You must add the following setting to your `db2dj.ini` file so that the Informix client correctly translates the GB 18030 data to Unicode:

```
DB_LOCALE=zh_cn.GB18030-2000
```

The next task in this sequence of tasks is registering the Informix wrapper.

Related tasks:

- "Registering nicknames for Informix tables, views, and synonyms" on page 286

Related reference:

- "Restrictions for the db2dj.ini file" on page 59
- "Restrictions for the db2dj.ini file" on page 59

Registering the Informix wrapper

Registering the Informix wrapper is part of the larger task of adding Informix data sources to a federated server.

You must register a wrapper to access Informix data sources. Wrappers are used by federated servers to communicate with and retrieve data from data sources. Wrappers are implemented as a set of library files.

Procedure:

To register a wrapper, issue the CREATE WRAPPER statement and specify the default name for the wrapper.

For example:

```
CREATE WRAPPER INFORMIX
```

Recommendation: Use the default wrapper name called INFORMIX. When you register the wrapper using the default name, the federated server automatically takes the default library name that is associated with that wrapper name.

If the wrapper name conflicts with an existing wrapper name in the federated database, you can substitute the default wrapper name with a name that you choose. If you use a name that is different from one of the default names, you must include the LIBRARY parameter in the CREATE WRAPPER statement.

For example, to register a wrapper with the name `inf_wrapper` on the federated server that uses the AIX operating system, issue the following statement:

```
CREATE WRAPPER inf_wrapper LIBRARY 'libdb2informix.a'
```

The name of the wrapper library file that you specify depends on the operating system of the federated server. See the list of Informix wrapper library files for the correct name to specify in the CREATE WRAPPER statement.

The next task in this sequence of tasks is registering the server definitions for the Informix wrapper.

Related tasks:
- "Registering the server definitions for an Informix data source" on page 282

Related reference:
- "Informix wrapper library files" on page 281
- "CREATE WRAPPER statement" in the *SQL Reference, Volume 2*

Informix wrapper library files

The following table lists the directory paths and library file names for the Informix wrapper.

When you install DB2 Information Integrator, 3 library files are added to the directory path listed in the table. For example, if the federated server is running on AIX, the wrapper library files added to the directory path are `libdb2informix.a`, `libdb2informixF.a`, and `libdb2informixU.a`.

When you register a wrapper, specify only the library file name that is listed in the table.

Table 76. Informix wrapper library locations and file names

Operating system	Directory path	Library file name
AIX	/usr/opt/db2_08_01/lib/	libdb2informix.a
HP-UX	/opt/IBM/db2/V8.1/lib	libdb2informix.sl
Linux	/opt/IBM/db2/V8.1/lib	libdb2informix.so
Solaris	/opt/IBM/db2/V8.1/lib	libdb2informix.so
Windows	%DB2PATH%\bin	db2informix.dll

%DB2PATH% is the environment variable that is used to specify the directory path where DB2 Information Integrator is installed on Windows. The default Windows directory path is `C:\Program Files\IBM\SQLLIB`.

Related tasks:
- "Registering the Informix wrapper" on page 280

Registering the server definitions for an Informix data source

Registering the server definitions for an Informix data source is part of the larger task of adding Informix data sources to federated servers.

In the federated database, you must define each Informix server that you want to access.

Procedure:

You can register a server definition from the DB2 Control Center or the DB2 command line:

- To do this task from the DB2 Control Center, use the Federated Objects wizard or right-click the **Server Definitions** folder and click **Create**. The Discover tool retrieves the node names for the Informix servers. You must specify the information for the DBNAME server option to register the server definition.
- To do this task from the DB2 command line, use the CREATE SERVER statement:
  1. Locate the node name in the Informix `sqlhosts` file or registry.

     Sample sqlhosts file:

     ```
     inf724 onsoctcp anaconda inmx724
     inf731 onsoctcp boa ifmx731
     inf92 onsoctcp python ifmx92
     ```

     The first value in each line is the *node_name*, such as `inf724`.

     The second value in each line is the *nettype*, or type of connection. In this example `onsoctcp` indicates this is a TCP/IP connection.

     The third value in each line is the host name, such as anaconda, boa, and python.

     The fourth value in each line is the service name, such as `inmx724`. The service name field depends on the *nettype* listed in the second value.

     Although the *node_name* is specified as an option in the CREATE SERVER SQL statement, it is required for Informix data sources.

     For more information about the format of this file and the meaning of these fields, see the Informix manual *Administrators Guide for Informix Dynamic Server*.
  2. Issue the CREATE SERVER statement.

     For example:

     ```
     CREATE SERVER server_name TYPE informix
         VERSION 9 WRAPPER INFORMIX
         OPTIONS (NODE 'node_name', DBNAME 'db_name');
     ```

After the server definition is registered, use the ALTER SERVER statement to add or drop server options.

The next task in this sequence of tasks is creating the user mappings for an Informix data source.

Related tasks:

- "Creating the user mapping for an Informix data source" on page 284

Related reference:

- "CREATE SERVER statement" in the *SQL Reference, Volume 2*

- Appendix D, “Server options for federated systems,” on page 575
- “CREATE SERVER statement - Examples for Informix wrapper” on page 283

CREATE SERVER statement - Examples for Informix wrapper

This topic provides several examples that show you how to use the CREATE SERVER statement to register server definitions for the Informix wrapper. This topic includes a complete example, which shows how to register a server definition with the required parameters, and an example with additional server options.

Complete example:

The following example shows you how to register a server definition for an Informix wrapper by using the CREATE SERVER statement:

```
CREATE SERVER asia TYPE informix VERSION 9 WRAPPER INFORMIX
      OPTIONS (NODE 'abc', DBNAME 'sales', IUD_APP_SVPT_ENFORCE 'N')
```

asia A name you assign to the Informix database server. This name must be unique. Duplicate server names are not allowed.

TYPE *informix*
Specifies the type of data source server to which you are configuring access. For the Informix wrapper, the server type must be `informix`.

VERSION *9*
The Informix database server version that you want to access. The supported Informix versions are 7, 8, and 9.

WRAPPER *INFORMIX*
The name you specified in the CREATE WRAPPER statement.

NODE *'abc'*
The name of the node where Informix database server resides. Obtain the node name from the `sqlhosts` file. This value is case sensitive.

Although the node name is specified as an option in the CREATE SERVER statement, it is required for Informix data sources.

DBNAME *'sales'*
The name of the Informix database that you want to access. This value is case sensitive.

Although the database name is specified as an option in the CREATE SERVER statement, it is required for Informix data sources.

IUD_APP_SVPT_ENFORCE *'N'*
Specifies whether the DB2 federated system should enforce detecting or building of application savepoint statements. Informix does not support application savepoint statements. When set to 'N', the federated server will not roll back transactions when an error is encountered. Your application must handle the error recovery.

The IUD_APP_SVPT_ENFORCE server option must be set to 'N' to enable replication to or from Informix data sources.

Server options example:

When you create the server definition, you can specify additional server options in the CREATE SERVER statement. The server options can be general server options and Informix-specific server options.

The following example shows an Informix server definition with additional server options:

```
CREATE SERVER asia TYPE informix VERSION 9 WRAPPER INFORMIX
      OPTIONS (NODE 'abc', DBNAME 'sales', FOLD_ID 'N', FOLD_PW 'N')
```

When the federated server connects to a data source, the federated server tries to connect using all possible combinations of uppercase and lowercase for the user ID and password, as well as the current case. The federated server might make up to nine connect attempts before successfully connecting to the data source server. These attempts can slow down connect times and might result in the user ID being locked out. You can prevent lock outs by specifying values for the FOLD_ID and FOLD_PW server options.

For example, you can set the FOLD_ID and FOLD_PW server options to 'N' (do not fold the user ID or password). If you establish these settings, then you must specify the user ID and password in the correct case. The advantage to setting these options to 'N' is that when an invalid user ID or password is specified, the wrapper will not keep trying the various uppercase and lowercase combinations. These two server options can reduce the chance of exceeding the maximum number of failed login attempts and the ID getting locked out.

Related tasks:

- "Registering the server definitions for an Informix data source" on page 282

Related reference:

- "CREATE SERVER statement" in the *SQL Reference, Volume 2*

Creating the user mapping for an Informix data source

Creating the user mapping for an Informix data source is part of the larger task of adding Informix data sources to federated servers.

When you attempt to access an Informix server, the federated server establishes a connection to the data source using a user ID and password that are valid for that data source. You must define an association (a user mapping) between each federated server user ID and password and the corresponding data source user ID and password. Create a user mapping for each user ID that will access the federated system to send distributed requests to the Informix data source.

Procedure:

To map a local user ID to the Informix server user ID and password, issue a CREATE USER MAPPING statement.

For example:

```
CREATE USER MAPPING FOR USERID SERVER INFORMIXSERVER
      OPTIONS (REMOTE_AUTHID 'remote_id', REMOTE_PASSWORD 'remote_password')
```

The next task in this sequence of tasks is testing the connection to the Informix server.

Related tasks:

- "Testing the connection to the Informix server" on page 286

Related reference:

- “CREATE USER MAPPING statement” in the *SQL Reference, Volume 2*
- “CREATE USER MAPPING statement - Examples for Informix wrapper” on page 285

CREATE USER MAPPING statement - Examples for Informix wrapper

This topic provides examples that show you how to use the CREATE USER MAPPING statement to map a federated server user ID to an Informix server user ID and password. This topic includes a complete example with required parameters and an example that shows you how to use the DB2 special register USER with the CREATE USER MAPPING statement.

Complete example:

The following example shows how to map a federated server user ID (*VINCENT*) to an Informix server user ID and password (*′vinnie′* and *′close2call′*):

```
CREATE USER MAPPING FOR VINCENT SERVER asia
      OPTIONS (REMOTE_AUTHID 'vinnie', REMOTE_PASSWORD 'close2call')
```

VINCENT
: Specifies the local user ID that you are mapping to a user ID that is defined at an Informix server.

SERVER *asia*
: Specifies the name of the Informix server that you registered in the CREATE SERVER statement.

REMOTE_AUTHID *′vinnie′*
: Specifies the user ID at the Informix database server to which you are mapping *VINCENT*. Use single quotation marks to preserve the case of this value unless you set the FOLD_ID server option to ′U′ or ′L′ in the CREATE SERVER statement.

REMOTE_PASSWORD *′close2call′*
: Specifies the password associated with *′vinnie′*. Use single quotation marks to preserve the case of this value unless you set the FOLD_PW server option to ′U′ or ′L′ in the CREATE SERVER statement.

Special register example:

The following example shows a CREATE USER MAPPING statement that includes the special register USER:

```
CREATE USER MAPPING FOR USER SERVER asia
      OPTIONS (REMOTE_AUTHID 'vinnie', REMOTE_PASSWORD 'close2call')
```

You can use the DB2 special register USER to map the authorization ID of the person who is issuing the CREATE USER MAPPING statement to the data source authorization ID that is specified in the REMOTE_AUTHID user option.

Related tasks:
- “Creating the user mapping for an Informix data source” on page 284

Related reference:
- “CREATE USER MAPPING statement” in the *SQL Reference, Volume 2*

Testing the connection to the Informix server

Testing the connection to the Informix server is part of the larger task of adding Informix data sources to federated servers.

You can test the connection to the Informix server by using the server definition and user mappings that you defined.

Procedure:

To test the connection:

1. Open a pass-through session to issue an SQL SELECT statement on the Informix system tables.

   For example:

   ```
   SET PASSTHRU server_name
   SELECT count(*) FROM informix.systables
   SET PASSTHRU RESET
   ```

   If the SQL SELECT statement returns a count, your server definition and your user mapping are set up properly.
2. If the SQL SELECT statement returns an error, you might need to:
   - Check the Informix server to make sure that it is configured for incoming connections.
   - Check your user mapping to make sure that the settings for the REMOTE_AUTHID and REMOTE_PASSWORD options are valid for the connections to the Informix server. Alter the user mapping, or create another user mapping as necessary.
   - Check the Informix Client SDK software on the DB2 federated server to make sure that it is installed and configured correctly to connect to the Informix server.
   - Check the settings of your DB2 federated variables to verify that they are correct for the Informix server. These variables include the system environment variables, the db2dj.ini variables, and the DB2 Profile Registry (db2set) variable.
   - Check your server definition. If necessary, drop it and create it again.

The next task in this sequence of tasks is registering nicknames for Informix tables, views, and synonyms.

Related tasks:

- "Adding Informix data sources to federated servers" on page 275
- "Registering nicknames for Informix tables, views, and synonyms" on page 286
- "Setting the Informix environment variables" on page 277

Related reference:

- "ALTER USER MAPPING statement" in the *SQL Reference, Volume 2*

Registering nicknames for Informix tables, views, and synonyms

Registering nicknames for Informix tables, views, and synonyms is part of the larger task of adding Informix data sources to federated servers.

For each Informix server that you define, register a nickname for each table, view, or synonym that you want to access. Use these nicknames, instead of the names of the data source objects, when you query the Informix servers.

The federated database relies on catalog statistics for nicknamed objects to optimize query processing. These statistics are gathered when you register a nickname for a data source object by using the CREATE NICKNAME statement. The federated database verifies the presence of the object at the data source, and then attempts to gather existing data source statistical data. Information that is useful to the optimizer is read from the data source catalogs and placed into the global catalog on the federated server. Because some or all of the data source catalog information might be used by the optimizer, update statistics (using the data source command that is equivalent to the DB2 **RUNSTATS** command) at the data source before you register a nickname.

Procedure:

To register a nickname, issue the CREATE NICKNAME statement.

For example:

```
CREATE NICKNAME informix_name FOR INFOSERVER."remote_schema"."remote.table"
```

Nicknames can be up to 128 characters in length.

Repeat this step for each Informix table, view, or synonym for which you want to create a nickname.

When you create the nickname, DB2 will use the connection to query the data source catalog. This query tests your connection to the data source by using the nickname. If the connection does not work, you will receive an error message.

Related reference:
- "RUNSTATS Command" in the *Command Reference*
- "CREATE NICKNAME statement" in the *SQL Reference, Volume 2*
- "CREATE NICKNAME statement - Examples for Informix wrapper" on page 287

CREATE NICKNAME statement - Examples for Informix wrapper

This topic provides an example that shows you how to use the CREATE NICKNAME statement to register a nickname for an Informix table, view, or synonym that you want to access.

This example shows how to specify a remote object for the Informix server under which the nickname is assigned:

```
CREATE NICKNAME JPSALES FOR asia."salesdata"."japan"
```

JPSALES
: A unique nickname used to identify the Informix table, view, or synonym.

  Note: the nickname is a two-part name—the schema and the nickname. If you omit the schema when you register the nickname, the schema of the nickname will be the authorization ID of the user who is registering the nickname.

asia."salesdata"."japan"
A three-part identifier for the remote object.
- *asia* is the name that you assigned to the Informix database server in the CREATE SERVER statement.
- *salesdata* is the name of the remote schema to which the table, view, or synonym belongs.
- *japan* is the name of the remote table, view, or synonym that you want to access.

The federated server folds the names of the Informix schemas and tables to uppercase unless you enclose the names in quotation marks.

Related tasks:
- "Registering nicknames for Informix tables, views, and synonyms" on page 286

Related reference:
- "CREATE NICKNAME statement" in the *SQL Reference, Volume 2*

Tuning and troubleshooting the configuration to Informix data sources

After you set up the configuration to Informix data sources, you might want to modify the configuration to improve performance.

Improving performance by setting the FOLD_ID and FOLD_PW server options

When the federated server connects to a data source, the server tries to connect using all possible combinations of uppercase and lowercase for the user ID and password. The server might make up to nine connect attempts before successfully connecting to the data source server. These attempts can slow down connect times and might result in the user ID getting locked out.

Procedure:

To improve performance, specify values for the FOLD_ID and FOLD_PW server options by using the ALTER SERVER OPTION statement.
- If all your Informix user IDs and passwords are in lowercase, setting the FOLD_ID and FOLD_PW server options with the value 'L' can improve your connect time. For example:
```
ALTER SERVER TYPE INFORMIX
   OPTIONS (ADD FOLD_ID 'L');
ALTER SERVER TYPE INFORMIX
   OPTIONS (ADD FOLD_PW 'L');
```
- The federated server attempts each combination of uppercase and lowercase values for the user ID and password. You can reduce the chance of the maximum number of failed login attempts being exceeded by setting these options to 'N' (do not fold the user ID and the password). If you establish these settings, then you need to always specify the user ID and password in the correct case. If an invalid user ID and password are specified, the wrapper will not keep trying the various combinations. For example:
```
ALTER SERVER TYPE INFORMIX
   OPTIONS (ADD FOLD_ID 'N');
ALTER SERVER TYPE INFORMIX
   OPTIONS (ADD FOLD_PW 'N');
```

Related tasks:
- "Adding Informix data sources to federated servers" on page 275

Related reference:
- "db2set - DB2 Profile Registry Command" in the *Command Reference*
- "ALTER SERVER statement" in the *SQL Reference, Volume 2*

Chapter 16. Configuring access to Microsoft SQL Server data sources

This chapter explains how to configure your federated server to access data that is stored in Microsoft SQL Server data sources. You can configure access to Microsoft SQL Server data sources by using the DB2 Control Center or by issuing SQL statements.

This chapter lists the tasks that you need to perform when you configure the federated server, contains examples of the SQL statements that you need, and provides tuning and troubleshooting information for configuring the federated server.

Adding Microsoft SQL Server to a federated server

Adding Microsoft SQL Server data sources to a federated server

To configure the federated server to access Microsoft SQL Server data sources, you must provide the federated server with information about the data sources and objects that you want to access.

| You can configure the federated server to access Microsoft SQL Server data sources
| by using the DB2 Control Center or the DB2 command line. The DB2 Control
| Center includes a wizard to guide you through the steps required to configure the
| federated server.

Prerequisites:

- DB2 Information Integrator must be installed on a server that will act as the federated server
- A federated database must exist on the federated server
- | The ODBC driver must be installed and configured on the federated server. The
  | supported drivers are Microsoft ODBC driver (Windows) and DataDirect
  | Technologies Connect for ODBC driver (UNIX).

Procedure:

To add Microsoft SQL Server data sources to a federated server:

1. Prepare the federated server and federated database.
   - On Windows, confirm that the ODBC System DSN is properly set up, and test the connection to the Microsoft SQL Server remote server.
   - On UNIX systems, update or create an `odbc.ini` file, and test the connection to the Microsoft SQL Server remote server.
2. Set the environment variables for the Microsoft SQL Server wrapper.
3. Register the wrapper.
4. Register the server definition.
5. Create the user mappings.
6. Test the connection to the Microsoft SQL Server remote server.

7. Register nicknames for Microsoft SQL Server tables and views.

Related concepts:
- "DB2 Information Integrator installation process - overview" in the *IBM DB2 Information Integrator Installation Guide for Linux, UNIX, and Windows*
- "Fast track to configuring your data sources" on page 55

Related tasks:
- "Creating a federated database" on page 51
- "Preparing the federated server to access Microsoft SQL Server data sources" on page 292
- "Registering the Microsoft SQL Server wrapper" on page 295
- "Registering the server definitions for a Microsoft SQL Server data source" on page 297
- "Creating a user mapping for a Microsoft SQL Server data source" on page 299
- "Testing the connection to the Microsoft SQL Server remote server" on page 301
- "Registering nicknames for Microsoft SQL Server tables and views" on page 302
- "Tuning and troubleshooting the configuration to Microsoft SQL Server data sources" on page 303
- "Checking the setup of the federated server" on page 37

Related reference:
- "Supported operating systems for DB2 Information Integrator (32-bit)" in the *IBM DB2 Information Integrator Installation Guide for Linux, UNIX, and Windows*
- "Supported operating systems for DB2 Information Integrator (64-bit)" in the *IBM DB2 Information Integrator Installation Guide for Linux, UNIX, and Windows*

Preparing the federated server to access Microsoft SQL Server data sources

Preparing the federated server and database to access Microsoft SQL Server data sources is part of the larger task of adding Microsoft SQL Server data sources to federated servers.

The steps to prepare the federated server to access Microsoft SQL Server data sources depends on the operating system that is running on your federated server.

Procedure:

To prepare the federated server:

On Windows:
1. Verify that the ODBC System DSN is set to connect to the Microsoft SQL Server data source by checking this setting in the Control Panel. Locate the existing entry for the Microsoft SQL Server remote server or create a new entry. The entry is the value that you will use for the NODE server option when you register the server in the federated database.
2. From the ODBC Data Source Administrator window, select **Configure** to test the connection from the ODBC Systems DSN to the Microsoft SQL Server data source. Alternatively, you can test the connection by using the Microsoft SQL Server query tool.

On UNIX:

1. Verify that the `odbc.ini` file is updated (or if necessary created) on the federated server.

   Recommendation: Place the `odbc.ini` file or a copy of this file in the home directory of the DB2 instance owner.
2. Verify that the path to the `odbc.ini` is in the ODBCINI environment variable. From an operating system command prompt, issue the following command:
   ```
   export ODBCINI=$HOME/.odbc.ini
   ```
3. Create the appropriate symbolic links:
   - On HP-UX, you need to create the following symbolic link:
     ```
     ln -s $DJX_ODBC_LIBRARY_PATH/libodbcinst.sl /usr/exe/libodbcinst.sl
     ```
     If you are using the DataDirect Technologies Connect for ODBC 4.2 driver, you must also create the following symbolic link:
     ```
     ln -s $DJX_ODBC_LIBRARY_PATH/libivicu19.sl /ivicu/exe/libivicu19.sl
     ```
   - On Linux, you need to create the following symbolic links:
     ```
     ln -s $DJX_ODBC_LIBRARY_PATH/../locale /usr/local/locale
     ln -s $DJX_ODBC_LIBRARY_PATH/libodbcinst.so /usr/lib/libodbcinst.so
     ```
     If you are using the DataDirect Technologies Connect for ODBC 4.2 driver, you must also create the following symbolic link:
     ```
     ln -s $DJX_ODBC_LIBRARY_PATH/libivicu19.so /usr/lib/libivicu19.so
     ```
   - On Solaris, you need to create the following symbolic link:
     ```
     ln -s $DJX_ODBC_LIBRARY_PATH/../locale $HOME/sqllib/locale
     ```
     `$HOME` is the home directory of the DB2 instance owner.
4. Test the connection from the federated server to the Microsoft SQL server data source by using the DataDirect Connect ODBC **demoodbc** tool.
   a. Run the **/opt/odbc/odbc.sh** script. This script sets up several operating system specific environment variables.
   b. Test the connection to the Microsoft SQL server data source by using the DataDirect Connect ODBC **demoodbc** tool. The **demoodbc** tool is located in the `/demo` subdirectory of the Connect ODBC libraries.

The next task in this sequence of tasks is setting the Microsoft SQL Server environment variables.

Related tasks:

- "Registering the Microsoft SQL Server wrapper" on page 295

Setting the Microsoft SQL Server environment variables

Setting the Microsoft SQL Server environment variables is part of the larger task of adding Microsoft SQL Server to a federated server.

When you install DB2 Information Integrator, the installation process attempts to set the Microsoft SQL Server environment variables in the `db2dj.ini` file.

The environment variables will not be set in the `db2dj.ini` file if you install the Microsoft SQL Server ODBC driver after you install DB2 Information Integrator.

The valid environment variables for Microsoft SQL Server are:

- DJX_ODBC_LIBRARY_PATH
- ODBCINI

- LD_LIBRARY_PATH (Solaris only)
- SHLIB_PATH (HP-UX only)
- DB2LIBPATH
- DB2ENVLIST

If your federated server runs HP-UX and you have a multi-partition instance configuration, you must export the SHLIB_PATH values in the userprofile.

Prerequisites:

A federated server that is properly set up to access your data sources. This includes the installation and configuration of any required software, such as the data source client software.

Restrictions:

See the topic: Restrictions for the db2dj.ini file

Procedure:

To set the Microsoft SQL Server ODBC driver environment variables automatically:
1. Install the Microsoft SQL Server ODBC driver on the DB2 federated server, if it is not already installed.
2. Set the required environment variables. You can set the environment variables automatically by running the DB2 Information Integrator installation again. From the launchpad, click **Install Products** and follow the instructions in the wizard.

To manually set the Microsoft SQL Server ODBC driver environment variables:
1. Edit the `db2dj.ini` file.
   - On federated servers running Windows, this file is located in the `sqllib\cfg` directory.
   - On federated servers running UNIX, this file is located in the `sqllib/cfg` directory.

   The `db2dj.ini` file contains configuration information about the Microsoft SQL Server ODBC driver installed on your federated server. If the file does not exist, you can create a new file with the name `db2dj.ini` name using any text editor. In the `db2dj.ini` file you must specify the fully qualified path for the variables, otherwise you will encounter errors.
2. Set the following environment variables (as necessary):

   DJX_ODBC_LIBRARY_PATH

   Set the directory path to the ODBC library files. For example:
   ```
   DJX_ODBC_LIBRARY_PATH=ODBC_driver_directory/lib
   ```
   `ODBC_driver_directory` is the directory path where the ODBC driver is installed.

   ODBCINI

   Set the ODBCINI environment variable to the directory path where your ODBC configuration file (odbc.ini) is located. Do not set the ODBCINI environment variable as a system variable. For example:
   ```
   ODBCINI=/home/db2inst1/.odbc.ini
   ```

LD_LIBRARY_PATH
: On Solaris, set the directory path to the ODBC library files. For example:

```
LD_LIBRARY_PATH=ODBC_driver_directory/lib
```

SHLIB_PATH
: On HP-UX, set the directory path to the ODBC library files. For example:

```
SHLIB_PATH=ODBC_driver_directory/lib
```

3. To access to Microsoft SQL Server, you need to set the directory path to the ODBC library files in the `lib` subdirectory. For example:
   ```
    db2set DB2LIBPATH=ODBC_driver_directory/lib
   ```
4. To use the Connect ODBC driver to access Microsoft SQL Server data sources, set DB2ENVLIST with a value of LIBPATH. For example:
   ```
   db2set DB2ENVLIST=LIBPATH
   ```
   `LIBPATH` is the directory path where the ODBC driver is installed.
5. To ensure that the environment variables are set on the federated server, recycle the DB2 instance. Issue the following commands to recycle the DB2 instance:
   ```
   db2stop
   db2start
   ```

On HP-UX multi-partition instance configurations:

1. Create a userprofile, if one does not already exist. The userprofile is located in the `$HOME/sqllib/` directory.
2. Add the SHLIB_PATH to the userprofile. The value for the SHLIB_PATH is the directory path where the ODBC driver is installed. For example, issue the following command:
   ```
   export SHLIB_PATH=$SHLIB_PATH:/home/DataDirectODBC/lib
   ```

The next task in this sequence of tasks is registering the Microsoft SQL Server wrapper.

Related tasks:

- "Registering nicknames for Microsoft SQL Server tables and views" on page 302

Related reference:

- "Restrictions for the db2dj.ini file" on page 59
- "Restrictions for the db2dj.ini file" on page 59

Registering the Microsoft SQL Server wrapper

Registering the Microsoft SQL Server wrapper is part of the larger task of adding Microsoft SQL Server data sources to a federated server.

You must register a wrapper to access Microsoft SQL Server data sources. Wrappers are used by federated servers to communicate with and retrieve data from data sources. Wrappers are implemented as a set of library files.

Procedure:

To register a wrapper, issue the CREATE WRAPPER statement.

For example:

```
CREATE WRAPPER MSSQLODBC3
```

Recommendation: Use the default wrapper name called MSSQLODBC3. When you register the wrapper by using the default name, the federated server automatically takes the default library name that is associated with that wrapper name.

If the wrapper name conflicts with an existing wrapper name in the federated database, you can substitute the default wrapper name with a name you choose. If you use a name that is different from the default name, you must include the LIBRARY parameter in the CREATE WRAPPER statement.

For example, to register a wrapper with the name `sqlserver_wrapper` on the federated server that uses the AIX operating system, issue the following statement:

```
CREATE WRAPPER sqlserver_wrapper LIBRARY 'libdb2mssql3.a'
```

The name of the wrapper library file that you specify depends on the operating system of the federated server. See the list of Microsoft SQL Server wrapper library files for the correct name to specify in the CREATE WRAPPER statement.

The next task in this sequence of tasks is registering the server definitions for the Microsoft SQL Server wrapper.

Related tasks:

- "Registering the server definitions for a Microsoft SQL Server data source" on page 297

Related reference:

- "Microsoft SQL Server wrapper library files" on page 296
- "CREATE WRAPPER statement" in the *SQL Reference, Volume 2*

Microsoft SQL Server wrapper library files

The following table lists the directory paths and library file names for the Microsoft SQL Server wrapper.

When you install DB2 Information Integrator, 3 library files are added to the directory path listed in the table. For example, if the federated server is running on AIX, the wrapper library files added to the directory path are `libdb2mssql3.a`, `libdb2mssql3F.a`, and `libdb2mssql3U.a`.

When you register a wrapper, specify only the library file name that is listed in the table.

Table 77. Microsoft SQL Server client library locations and file names

Operating system	Directory path	Wrapper library file
AIX	/usr/opt/db2_08_01/lib/	libdb2mssql3.a
HP-UX	/opt/IBM/db2/V8.1/lib	libdb2mssql3.sl
Linux	/opt/IBM/db2/V8.1/lib	libdb2mssql3.so
Solaris	/opt/IBM/db2/V8.1/lib	libdb2mssql3.so
Windows	%DB2PATH%\bin	db2mssql3.dll

%DB2PATH% is the environment variable that is used to specify the directory path where DB2 Information Integrator is installed on Windows. The default Windows directory path is `C:\Program Files\IBM\SQLLIB`.

Related tasks:

- "Registering the Microsoft SQL Server wrapper" on page 295

Registering the server definitions for a Microsoft SQL Server data source

Registering the server definitions for a Microsoft SQL Server data source is part of the larger task of adding Microsoft SQL Server data sources to federated servers.

In the federated database, you must define each Microsoft SQL Server remote server that you want to access. You must first locate the node name of the Microsoft SQL Server remote server, and then use this node name when you register the server definition by issuing the CREATE SERVER statement.

Procedure:

To register a server definition for a Microsoft SQL Server data source:

1. Locate the node name.
   - If your federated server is using Windows, the NODE is the System DSN name that you specified for the Microsoft SQL Server remote server that you are accessing.
   - If your federated server is using UNIX, the NODE is defined in the `.odbc.ini` file.

     The following is an example of a `.odbc.ini` file on AIX.

     Example .odbc.ini file on AIX:

     ```
     rawilson=MS SQL Server 7.0
     medusa=MS SQL Server 7.0
     [rawilson]
     Driver=/opt/odbc/lib/ivmsss16.so
     Description=MS SQL Server Driver for AIX
       Address=9.112.30.39,1433
     [medusa]
     Driver=/opt/odbc/lib/ivmsss16.so
     Description=MS SQL Server Driver for AIX
     Address=9.112.98.123,1433
     ```

     At the top of the `.odbc.ini` file, there is a section labeled [ODBC Data Sources] which lists the nodes. Each of the nodes has a section [node_name] that describes each node.

   Although the node name is specified as an option in the CREATE SERVER statement, it is required for Microsoft SQL Server data sources.
2. Issue the CREATE SERVER statement.

   For example:

   ```
   CREATE SERVER server_name TYPE MSSQLSERVER VERSION 7.0 WRAPPER mssqlodb3
         OPTIONS (NODE 'sqlnode', DBNAME 'mssdb');
   ```

After the server definition is created, use the ALTER SERVER statement to add or drop server options.

The next task in this sequence of tasks is creating a user mapping for a Microsoft SQL Server data source.

Related tasks:

- "Creating a user mapping for a Microsoft SQL Server data source" on page 299

Related reference:

- "ALTER SERVER statement" in the *SQL Reference, Volume 2*
- "CREATE SERVER statement" in the *SQL Reference, Volume 2*
- Appendix D, "Server options for federated systems," on page 575
- "CREATE SERVER statement - Examples for Microsoft SQL Server wrapper" on page 298

CREATE SERVER statement - Examples for Microsoft SQL Server wrapper

This topic provides examples that show you how to use the CREATE SERVER statement to register servers for the Microsoft SQL Server wrapper. This topic includes a complete example, which shows how to register a server with required parameters, and an example with additional server options.

Complete example:

The following example shows you how to register a server definition for a Microsoft SQL Server wrapper by issuing the CREATE SERVER statement:

```
CREATE SERVER sqlserver TYPE MSSQLSERVER VERSION 7.0 WRAPPER mssqlodbc3
      OPTIONS (NODE 'sqlnode', DBNAME 'africa');
```

sqlserver
: A name that you assign to the Microsoft SQL Server remote server. This name must be unique. Duplicate server names are not allowed.

TYPE *MSSQLSERVER*
: The type of data source to which you are configuring access. The TYPE parameter for the Microsoft SQL Server wrappers must be *MSSQLSERVER*.

VERSION *7.0*
: The version of Microsoft SQL Server database server software that you want to access. Supported versions are 6.5, 7.0, and 2000.

WRAPPER *mssqlodbc3*
: The wrapper name that you specified in the CREATE WRAPPER statement.

NODE *'sqlnode'*
: On Windows, the System DSN name for the Microsoft SQL Server remote server that you are accessing. On UNIX, the node that is defined in the `.odbc.ini` file. This value is case sensitive.

: Although the name of the node is specified as an option in the CREATE SERVER statement, it is required for Microsoft SQL Server data sources.

DBNAME *'africa'*
: The name of the database that you want to access. This value is case sensitive.

Although the name of the database is specified as an option in the CREATE SERVER statement, it is required for Microsoft SQL Server data sources.

Server option examples:

When you register the server, you can specify additional server options in the CREATE SERVER statement. These server options include general server options and Microsoft SQL Server-specific server options.

The following example shows how to use the COLLATING_SEQUENCE server option:

```
CREATE SERVER sqlserver TYPE MSSQLSERVER
    VERSION 7.0
    WRAPPER mssqlodbc3
    OPTIONS (NODE 'sqlnode', DBNAME 'africa', COLLATING_SEQUENCE 'I');
```

The COLLATING_SEQUENCE server option specifies whether the data source uses the same collating sequence as the federated server. On a Microsoft SQL Server database server that is running Windows NT or Windows 2000, the default collating sequence is case insensitive (for example, ’STEWART’ and ’StewART’ are considered equal). To guarantee correct results from the federated server, set the COLLATING_SEQUENCE server option to ’I’. This setting indicates that the Microsoft SQL Server data source is case insensitive.

The federated server does not push down queries if the results that are returned from the data sources will be different from the results that are returned when processing the query at the federated server. When you set the COLLATING_SEQUENCE server option to ’I’, the federated server does not push down queries with string data or expressions and that include the following clauses, predicates, or functions:
- GROUP BY clauses
- DISTINCT clauses
- Basic predicates, such as equal to (=)
- Aggregate functions, such as MIN or MAX

Related tasks:
- “Registering the server definitions for a Microsoft SQL Server data source” on page 297

Related reference:
- “CREATE SERVER statement” in the *SQL Reference, Volume 2*

Creating a user mapping for a Microsoft SQL Server data source

Creating a user mapping for a Microsoft SQL Server data source is part of the larger task of adding Microsoft SQL Server data sources to federated servers.

When you attempt to access a Microsoft SQL Server data source, the federated server establishes a connection to the data source using a user ID and password that are valid for that data source. You must define an association (a user mapping) between each federated server user ID and password and the corresponding data source user ID and password. Create a user mapping for each user ID that will access the federated system to send distributed requests.

Procedure:

To map a local user ID to the Microsoft SQL Server remote server user ID and password, issue a CREATE USER MAPPING statement.

For example:

```
CREATE USER MAPPING FOR userid SERVER sqlserver
      OPTIONS (REMOTE_AUTHID 'remote_id', REMOTE_PASSWORD 'remote_password')
```

The next task in this sequence of tasks is testing the connection to the Microsoft SQL Server remote server.

Related tasks:

- "Testing the connection to the Microsoft SQL Server remote server" on page 301

Related reference:

- "CREATE USER MAPPING statement" in the *SQL Reference, Volume 2*
- "CREATE USER MAPPING statement - Examples for Microsoft SQL Server wrapper" on page 300

CREATE USER MAPPING statement - Examples for Microsoft SQL Server wrapper

This topic provides examples that show you how to use the CREATE USER MAPPING statement to map a federated server user ID to a Microsoft SQL Server remote server user ID and password. This topic includes a complete example with required parameters and an example that shows you how to use the DB2 special register USER with the CREATE USER MAPPING statement.

Complete example:

The following example shows how to map a federated server user ID to a Microsoft SQL Server remote server user ID and password:

```
CREATE USER MAPPING FOR elizabeth SERVER sqlserver
      OPTIONS (REMOTE_AUTHID 'liz', REMOTE_PASSWORD 'abc123')
```

elizabeth
: Specifies the local user ID that you are mapping to a user ID that is defined at the Microsoft SQL Server remote server.

SERVER *sqlserver*
: Specifies the name of the Microsoft SQL Server remote server that you defined in the CREATE SERVER statement.

REMOTE_AUTHID *'liz'*
: Specifies the user ID at the Microsoft SQL Server remote server to which you are mapping *elizabeth*. Use single quotation marks to preserve the case of this value unless you set the FOLD_ID server option to 'U' or 'L' in the CREATE SERVER statement.

REMOTE_PASSWORD *'abc123'*
: Specifies the password that is associated with *'liz'*. Use single quotation marks to preserve the case of this value unless you set the FOLD_PW server option to 'U' or 'L' in the CREATE SERVER statement.

Special register example:

The following example shows a CREATE USER MAPPING statement that includes the special register USER:

```
CREATE USER MAPPING FOR USER SERVER sqlserver
      OPTIONS (REMOTE_AUTHID 'liz', REMOTE_PASSWORD 'abc123')
```

You can use the DB2 special register USER to map the authorization ID of the person who is issuing the CREATE USER MAPPING statement to the data source authorization ID that is specified in the REMOTE_AUTHID user option.

Related tasks:

- "Creating a user mapping for a Microsoft SQL Server data source" on page 299

Related reference:

- "CREATE USER MAPPING statement" in the *SQL Reference, Volume 2*

Testing the connection to the Microsoft SQL Server remote server

Testing the connection to the Microsoft SQL Server remote server is part of the larger task of adding Microsoft SQL Server data sources to federated servers.

You can test the connection to the Microsoft SQL Server remote server by using the server definition and user mappings that you defined.

Procedure:

To test the connection:

1. Open a pass-through session to issue an SQL SELECT statement on the Microsoft SQL Server system tables.

   For example:

   ```
   SET PASSTHRU remote_server_name
   SELECT count(*) FROM dbo.sysobjects
   SET PASSTHRU RESET
   ```

   If the SQL SELECT statement returns a count, your server definition and your user mapping are set up properly.
2. If the SQL SELECT statement returns an error, you might need to:
   - Check the Microsoft SQL Server remote server to make sure that it is started.
   - Check the Microsoft SQL Server remote server to make sure that it is configured for incoming connections.
   - Check your user mapping to make sure that the settings for the REMOTE_AUTHID and REMOTE_PASSWORD options are valid for the connections to the Microsoft SQL Server remote server. Alter the user mapping, or create another user mapping as necessary.
   - Check the ODBC drivers on the DB2 federated server to make sure that they are installed and configured correctly to connect to the Microsoft SQL Server remote server.
   - Check the settings of your DB2 federated variables to verify that they are correct for the Microsoft SQL Server remote server. These variables include the system environment variables, the db2dj.ini variables, and the DB2 Profile Registry (db2set) variable.

- Check your server definition. If necessary, drop it and create it again.

The next task in this sequence of tasks is registering nicknames for Microsoft SQL Server tables and views.

Related tasks:

- "Adding Microsoft SQL Server data sources to a federated server" on page 291
- "Registering nicknames for Microsoft SQL Server tables and views" on page 302
- "Setting the Microsoft SQL Server environment variables" on page 293

Related reference:

- "ALTER USER MAPPING statement" in the *SQL Reference, Volume 2*

Registering nicknames for Microsoft SQL Server tables and views

Registering nicknames for Microsoft SQL Server tables and views is part of the larger task of adding Microsoft SQL Server data sources to federated servers.

For each Microsoft SQL Server remote server that you define, register a nickname for each table or view that you want to access. Use these nicknames, instead of the names of the data source objects, when you query the Microsoft SQL Server remote servers.

The federated database relies on catalog statistics for nicknamed objects to optimize query processing. These statistics are gathered when you register a nickname for a data source object by using the CREATE NICKNAME statement. The federated database verifies the presence of the object at the data source, and then attempts to gather existing data source statistical data. Information that is useful to the optimizer is read from the data source catalogs and placed into the global catalog on the federated server. Because some or all of the data source catalog information might be used by the optimizer, update statistics (using the data source command that is equivalent to the DB2 **RUNSTATS** command) at the data source before you register a nickname.

Procedure:

To register a nickname, issue the CREATE NICKNAME statement.

For example:

```
CREATE NICKNAME mss_name FOR sqlserver."remote_schema"."remote.table"
```

Nicknames can be up to 128 characters in length.

Repeat this step for each Microsoft SQL Server table or view for which you want to create a nickname.

When you create the nickname, DB2 uses the connection to query the data source catalog tables (Microsoft SQL Server refers to these tables as system tables). This query tests your connection to the data source by using the nickname. If the connection does not work, you will receive an error message.

Related reference:

- "RUNSTATS Command" in the *Command Reference*

- "CREATE NICKNAME statement" in the *SQL Reference, Volume 2*
- "CREATE NICKNAME statement - Examples for Microsoft SQL Server wrapper" on page 303

CREATE NICKNAME statement - Examples for Microsoft SQL Server wrapper

This topic provides an example that shows you how to use the CREATE NICKNAME statement to register a nickname for a Microsoft SQL Server table or view that you want to access.

This example shows how to specify a remote object for the Microsoft SQL Server remote server under which the nickname is assigned:

```
CREATE NICKNAME cust_africa FOR sqlserver.customers.egypt
```

cust_africa
: A unique nickname for the Microsoft SQL Server table or view.

  Note: The nickname is a two-part name which includes the schema and the nickname. If you omit the schema when you register the nickname, the schema of the nickname will be the authentication ID of the user creating the nickname.

sqlserver.customers.egypt
: A three-part identifier for the remote object.
  - *sqlserver* is the name that you assigned to the Microsoft SQL Server database server in the CREATE SERVER statement.
  - *customers* is the name of the remote schema to which the table or view belongs.
  - *egypt* is the name of the remote table or view that you want to access.

The federated server folds the names of the Microsoft SQL Server schemas and tables to uppercase unless you enclose the names in quotation marks.

Related tasks:
- "Registering nicknames for Microsoft SQL Server tables and views" on page 302

Related reference:
- "CREATE NICKNAME statement" in the *SQL Reference, Volume 2*

Tuning and troubleshooting the configuration to Microsoft SQL Server data sources

After you set up the configuration to Microsoft SQL Server data sources, you might want to modify the configuration to improve performance.

Obtaining ODBC traces

If you experience problems when accessing the data source, you can obtain ODBC tracing information to analyze and resolve the problems. However, activating a trace does impact system performance. You should turn off tracing after you have resolved the problems.

On federated servers that run Windows, use the trace tool that is provided by the ODBC Data Source Administrator to ensure that the ODBC tracing works properly.

On federated servers that run UNIX, you can set tracing on by changing for the odbc.ini file. For example, if you use the DataDirect ODBC 3.x driver, find the example of the odbc.ini file in the client directory. The odbc.ini file contains a sample of what is needed for the trace files:

```
[ODBC]
Trace=0
TraceFile=/home/user1/trace_dir/filename.xxx
TraceDll==ODBC_driver_directory/odbctrac.so
InstallDir=/opt/odbc
```

| To turn tracing ON, set the first line to Trace=1. To turn tracing OFF, set the first
| line to Trace=0. The TraceFile should point to a path and file name that the
| federated database instance has write access to.

Related tasks:

- "Adding Microsoft SQL Server data sources to a federated server" on page 291

Related reference:

- "db2set - DB2 Profile Registry Command" in the *Command Reference*

Chapter 17. Configuring access to ODBC data sources

This chapter explains how to configure your federated server to access data that is stored in ODBC data sources. You can configure access to ODBC data sources by using the DB2 Control Center or by issuing SQL statements.

This chapter lists the tasks that you need to perform when you configure the federated server, contains examples of the SQL statements that you need, and provides tuning and troubleshooting information for configuring the federated server.

Adding ODBC to a federated system

Adding ODBC data sources to a federated server

To configure the federated server to access ODBC data sources, you must provide the federated server with information about the data sources and objects that you want to access.

You can configure the federated server to access ODBC data sources by using the DB2 Control Center or the DB2 command line. The DB2 Control Center includes a wizard to guide you through the steps required to configure the federated server.

The data sources that are accessed through the ODBC API are referred to in this text as ODBC data sources.

Depending on your needs, you can access Excel data using the ODBC wrapper instead of using the Excel wrapper. The specific steps to configure the ODBC wrapper to access Excel data are documented in a separate topic.

Prerequisites:

- DB2 Information Integrator must be installed on a server that will act as the federated server
- A federated database must exist on the federated server
- The ODBC driver must be is installed and configured on the federated server. The ODBC wrapper supports ODBC 3.x.
- The proper setup of the system environment variables, db2dj.ini variables, and DB2 Profile Registry (db2set) variables. Check the vendor documentation for the required variables for your ODBC client. The LIBPATH variable might be required.

Restrictions:

- The ODBC wrapper cannot be used to access any DB2 family data sources. Use the DRDA wrapper to access DB2 family data sources.
- The ODBC wrapper does not support the following functions and statements:
  - LOCK TABLE statements on nicknames
  - Features deprecated in ODBC 3.x
  - X/Open or SQL/CLI drivers
  - Stored procedure nicknames

  - Statement-level atomicity enforcement using remote savepoint statements
  - 64–bit clients
  - WITH HOLD cursors
- For data sources that do not support positioned update and delete operations, positioned UPDATE and DELETE statements and certain searched UPDATE and DELETE statements on a nickname will fail if a unique index on non-nullable columns does not exist on the nickname or its corresponding remote table. The error SQL30090 reason code 21 is returned when these statements fail.
- The ODBC wrapper does not support INSERT, UPDATE, or DELETE statements against data sources that restrict the number of active statements for each connection. Consult the documentation for your data source to determine if the data source restricts the number of active statements for each connection. This restriction applies to IBM Red Brick Warehouse.

Procedure:

To add ODBC data sources to a federated server:
1. Prepare the federated server and federated database.
2. Register the wrapper.
3. Register the server definition.
4. Create the user mappings.
5. Test the connection to the ODBC data source.
6. Register nicknames for ODBC data source tables and views.

Related concepts:
- "DB2 Information Integrator installation process - overview" in the *IBM DB2 Information Integrator Installation Guide for Linux, UNIX, and Windows*
- "Fast track to configuring your data sources" on page 55
- "Methods of accessing Excel data" on page 25

Related tasks:
- "Accessing Excel data using the ODBC wrapper" on page 316
- "Creating a federated database" on page 51
- "Preparing the federated server to access data sources through ODBC" on page 306
- "Registering the ODBC wrapper" on page 307
- "Registering the server definitions for an ODBC data source" on page 309
- "Creating a user mapping for an ODBC data source" on page 311
- "Testing the connection to the ODBC data source server" on page 313
- "Registering nicknames for ODBC data source tables and views" on page 314

Related reference:
- "Choose the correct wrapper" on page 24

Preparing the federated server to access data sources through ODBC

Preparing the federated server to access data sources through ODBC is part of the larger task of adding ODBC data sources to federated servers.

The steps that you need to follow to prepare the federated server to access data sources through ODBC depend on the operating system that is running on your federated server.

Note: The ODBC driver and the operating system that you use have unique library path locations.

Procedure:

To prepare the federated server:

On Windows:

Verify that the ODBC 3.x driver has been installed and configured on the federated server. The node name for the ODBC data source must be defined in the System DSN. See the ODBC driver documentation for the installation and configuration procedures.

If you used the Microsoft ODBC Data Source Administrator to configure the DSN, you can check this setting in the Control Panel. Make sure that ODBC data source is registered as a System DSN. Otherwise DB2 might not be able to find the DSN.

On UNIX:

1. Consult the documentation from the ODBC client vendor for instructions on how to configure the ODBC client.
2. If the client is DataDirect ODBC or RedBrick, verify that the appropriate symbolic link is created:
   - On HP-UX, the symbolic link is from `/usr/exe/libodbcinst.sl` to `$ODBC_LIBRARY_PATH/libodbcinst.sl`.
   - On Linux, the symbolic link is from `/usr/lib/libodbcinst.so` to `$DJX_ODBC_LIBRARY_PATH/libodbcinst.so`.
   - On Solaris, the symbolic link is from `$HOME/sqllib/locale` to `$DJX_ODBC_LIBRARY_PATH/../locale`. `$HOME` is the home directory of the DB2 instance owner.

The next task in this sequence of tasks is registering the ODBC wrapper.

Related tasks:

- "Registering the ODBC wrapper" on page 307

Registering the ODBC wrapper

Registering the ODBC wrapper is part of the larger task of adding ODBC data sources to federated servers.

You must register a wrapper to access ODBC data sources. Wrappers are used by federated servers to communicate with and retrieve data from data sources. Wrappers are implemented as a set of library files.

Procedure:

To register a wrapper, issue the CREATE WRAPPER statement and specify the default name for the wrapper.

For example:

```
CREATE WRAPPER ODBC
```

Recommendation: Use the default wrapper name called ODBC. When you register the wrapper that uses the default name, the federated server automatically takes the default library name that is associated with that wrapper.

If the wrapper name conflicts with an existing wrapper name in the federated database, you can substitute the default wrapper name with a name you choose. If you use a name that is different from the default name, you must include the LIBRARY parameter in the CREATE WRAPPER statement.

For example, to register a wrapper with the name `odbc_wrapper` on the federated server that uses the AIX operating system, issue the following statement:

```
CREATE WRAPPER odbc_wrapper
       LIBRARY 'libdb2rcodbc.a' OPTIONS (MODULE '/usr/lib/odbc.a')
```

MODULE *'/usr/lib/odbc.a'* is the full path of the library that contains the ODBC Driver Manager.

The name of the wrapper library file that you specify depends on the operating system of the federated server. See the list of ODBC wrapper library files for the correct name to specify in the CREATE WRAPPER statement.

The next task in this sequence of tasks is registering the server definitions for the ODBC wrapper.

Related tasks:

- "Registering the server definitions for an ODBC data source" on page 309

Related reference:

- "ODBC wrapper library files" on page 308
- "CREATE WRAPPER statement" in the *SQL Reference, Volume 2*
- "CREATE WRAPPER statement - Examples for ODBC wrapper" on page 309

ODBC wrapper library files

The following table lists the directory paths and library file names for the ODBC wrapper.

When you install DB2 Information Integrator, 3 library files are added to the directory path listed in the table. For example, if the federated server is running on AIX, the wrapper library files added to the directory path are `libdb2rcodbc.a`, `libdb2rcodbcF.a`, and `libdb2rcodbcU.a`.

When you register a wrapper, specify only the library file name that is listed in the table.

Table 78. ODBC client library locations and file names

Operating system	Directory path	Wrapper library file
AIX	/usr/opt/db2_08_01/lib/	libdb2rcodbc.a
HP-UX	/opt/IBM/db2/V8.1/lib	libdb2rcodbc.sl
Linux	/opt/IBM/db2/V8.1/lib	libdb2rcodbc.so

Table 78. ODBC client library locations and file names (continued)

Operating system	Directory path	Wrapper library file
Solaris	/opt/IBM/db2/V8.1/lib	libdb2rcodbc.so
Windows	%DB2PATH%\bin	db2rcodbc.dll

%DB2PATH% is the environment variable that is used to specify the directory path where DB2 Information Integrator is installed on Windows. The default Windows directory path is `C:\Program Files\IBM\SQLLIB`.

Related tasks:
- "Registering the ODBC wrapper" on page 307

CREATE WRAPPER statement - Examples for ODBC wrapper

This topic provides examples that show you how to use the CREATE WRAPPER statement to register wrappers for ODBC data sources.

Example for UNIX systems:

The following example shows you how to register a wrapper by issuing the CREATE WRAPPER statement on a UNIX operating system:

```
CREATE WRAPPER odbc OPTIONS (MODULE '/usr/lib/odbc.so')
```

In this example, *odbc* is the name that you assign to the wrapper that is being registered in the federated database. MODULE *'/usr/lib/odbc.so'* is the full path of the library that contains the ODBC Driver Manager.

You must specify the MODULE option on UNIX operating systems. On Windows, the MODULE option defaults to *'odbc32.dll'*.

Example for Windows:

The following example shows you how to register a wrapper by issuing the CREATE WRAPPER statement on a Windows operating system:

```
CREATE WRAPPER odbc LIBRARY 'db2rcodbc.dll'
```

In this example, *odbc* is the name that you assign to wrapper that is being registered in the federated database. LIBRARY *'db2rcodbc.dll'* is the library name for the ODBC wrapper.

Related tasks:
- "Registering the ODBC wrapper" on page 307

Related reference:
- "CREATE WRAPPER statement" in the *SQL Reference, Volume 2*

Registering the server definitions for an ODBC data source

Registering the server definitions for an ODBC data source is part of the larger task of adding ODBC data sources to federated servers.

In the federated database, you must define each ODBC data source server that you want to access.

Procedure:

To register a server definition for an ODBC data source:

Issue the CREATE SERVER statement.

For example:

```
CREATE SERVER server_name TYPE data_source_type
     VERSION version WRAPPER odbc
     OPTIONS (NODE 'node_name')
```

The TYPE and VERSION parameters are optional.

Although NODE is specified as option in the CREATE SERVER statement, it is required for ODBC data sources.

After the server definition is created, use the ALTER SERVER statement to add or drop server options.

The next task in this sequence of tasks is creating a user mapping for an ODBC data source.

Related tasks:
- "Creating a user mapping for an ODBC data source" on page 311

Related reference:
- "ALTER SERVER statement" in the *SQL Reference, Volume 2*
- "CREATE SERVER statement" in the *SQL Reference, Volume 2*
- Appendix D, "Server options for federated systems," on page 575
- "CREATE SERVER statement - Examples of ODBC wrapper" on page 310

CREATE SERVER statement - Examples of ODBC wrapper

This topic provides examples that show you how to use the CREATE SERVER statement to register servers for the ODBC wrapper. This topic includes a complete example, which shows how to register a server with required parameters, and an example with additional server options.

Complete example:

The following example shows you how to register a server definition for an ODBC wrapper by issuing the CREATE SERVER statement:

```
CREATE SERVER mysql_server TYPE mysql
       VERSION 4.0 WRAPPER odbc
        OPTIONS (NODE 'odbc_node', DBNAME 'venice')
```

mysql_server
: A name that you assign to the ODBC data source server. This name must be unique. Duplicate server names are not allowed.

TYPE *mysql*
: Specifies the type of data source to which you are configuring access. This parameter is optional.

VERSION *4.0*
: The version of the ODBC data source that you want to access. This parameter is optional.

WRAPPER *odbc*
: The wrapper name that you specified in the CREATE WRAPPER statement.

NODE *'mysql_node'*
: The name of the node (the system DSN name) that was assigned to the ODBC data source when the DSN was defined. This value is case sensitive. On Windows, this value must be the name of a system DSN in the ODBC Data Administration window. On UNIX, consult the ODBC client vendor documentation for information about the value to use.

  Although the NODE is specified as an option in the CREATE SERVER statement, it is required for ODBC data sources.

DBNAME *'venice'*
: The name of the database that you want to access. This value is case sensitive.

Server options example:

The following example shows how to use the DB2_TABLE_QUOTE_CHAR, DB2_ID_QUOTE_CHAR, and the DB2_AUTHID_QUOTE_CHAR server options.

Some ODBC data sources (for example, MySQL) cannot process quotation marks around table names and column names in SQL statements. To access these data sources, you must include the following server options in the CREATE SERVER statement:

- DB2_TABLE_QUOTE_CHAR ' ` '
- DB2_ID_QUOTE_CHAR ' ` '
- DB2_AUTHID_QUOTE_CHAR ' ` '

The ` character is the delimiter for identifiers such as schema names, table names, and column names.

For example:

```
CREATE SERVER mysql_server TYPE mysql
     VERSION 4.0 WRAPPER odbc
     OPTIONS (NODE 'mysql_node', DB2_TABLE_QUOTE_CHAR ' ` ',
          DB2_ID_QUOTE_CHAR ' ` ' DB2_AUTHID_QUOTE_CHAR ' ` ')
```

Related tasks:

- “Registering the server definitions for an ODBC data source” on page 309

Related reference:

- “CREATE SERVER statement” in the *SQL Reference, Volume 2*

Creating a user mapping for an ODBC data source

Creating a user mapping for an ODBC data source is part of the larger task of adding ODBC data sources to federated servers.

When you attempt to access an ODBC data source, the federated server establishes a connection to the data source using a user ID and password that are valid for that data source. You must define an association (a user mapping) between each federated server user ID and password and the corresponding data source user ID and password. Create a user mapping for each user ID that will access the federated system to send distributed requests.

Procedure:

To map a local user ID to the ODBC data source user ID and password, issue a CREATE USER MAPPING statement.

For example:

```
CREATE USER MAPPING FOR userid SERVER server_name
      OPTIONS (REMOTE_AUTHID 'remote_id', REMOTE_PASSWORD 'remote_password')
```

The next task in this sequence of tasks is testing the connection to the ODBC data source server.

Related tasks:

- "Testing the connection to the ODBC data source server" on page 313

Related reference:

- "CREATE USER MAPPING statement" in the *SQL Reference, Volume 2*
- "CREATE USER MAPPING statement - Examples for ODBC wrapper" on page 312

CREATE USER MAPPING statement - Examples for ODBC wrapper

This topic provides examples that show you how to use the CREATE USER MAPPING statement to map a local user ID to an ODBC data source user ID and password. This topic includes a complete example with required parameters and an example that shows you how to use the DB2 special register USER with the CREATE USER MAPPING statement.

Complete example:

The following example shows how to map a local user ID to an ODBC data source user ID and password:

```
CREATE USER MAPPING FOR arturo SERVER server_name
      OPTIONS (REMOTE_AUTHID 'art', REMOTE_PASSWORD 'red4blue')
```

arturo Specifies the local user ID that you are mapping to a user ID that is defined at the ODBC data source.

server_name
Specifies the name of the ODBC data source that you defined in the CREATE SERVER statement.

'art' Specifies the user ID at the ODBC data source to which you are mapping *arturo*. Use single quotation marks to preserve the case of this value unless you set the FOLD_ID server option to 'U' or 'L' in the CREATE SERVER statement.

'red4blue'
Specifies the password associated with *'art'*. Use single quotation marks to preserve the case of this value unless you set the FOLD_PW server option to 'U' or 'L' in the CREATE SERVER statement.

Special register example:

The following example shows a CREATE USER MAPPING statement that includes the special register USER:

```
CREATE USER MAPPING FOR USER SERVER server_name
      OPTIONS (REMOTE_AUTHID 'art', REMOTE_PASSWORD 'red4blue')
```

You can use the DB2 special register USER to map the authorization ID of the person who is issuing the CREATE USER MAPPING statement to the data source authorization ID that is specified in the REMOTE_AUTHID user option.

Related tasks:

- "Creating a user mapping for an ODBC data source" on page 311

Related reference:

- "CREATE USER MAPPING statement" in the *SQL Reference, Volume 2*

Testing the connection to the ODBC data source server

Testing the connection to the ODBC data source server is part of the larger task of adding ODBC data sources to federated servers.

You can test the connection to the ODBC data source server by using the server definition and the user mappings that you defined.

Prerequisites:

The data source that you are using must support pass-through sessions.

Procedure:

To test the connection:

1. Open a pass-through session to issue an SQL SELECT statement on the ODBC data source system tables.

   For example:

   ```
   SET PASSTHRU server_name
   SELECT COUNT(*) FROM schema_name.table_name
   SET PASSTHRU RESET
   ```

   The *server_name* is the name of the ODBC data source that you defined in the CREATE SERVER statement.

   The *schema_name* is the name of the schema at the remote ODBC data source. If your ODBC data source does not support schemas, omit the schema from the statement.

   The *table_name* is the name of the table at the remote ODBC data source.

   If the SQL SELECT statement returns a count, your server definition and your user mappings are set up properly.
2. If the SQL SELECT statement returns an error, you might need to:
   - Verify that the data source is available.
   - If applicable, check the data source server to make sure that it is configured for incoming connections.

- Check your user mapping to make sure that the settings for the REMOTE_AUTHID and REMOTE_PASSWORD options are valid for the connections to the ODBC data source. Alter the user mapping, or create another user mapping as necessary.
- Check the ODBC driver on the DB2 federated server to make sure that it is installed and configured correctly to connect to the ODBC data source server. On Windows operating systems, use the ODBC Data Source Administrator tool to check the driver. On UNIX operating systems, consult the ODBC client vendor's documentation.
- Check your server definition. If necessary, drop it and create it again.

The next task in this sequence of tasks is registering nicknames for ODBC data source tables and views.

Related tasks:

- "Adding ODBC data sources to a federated server" on page 305
- "Preparing the federated server to access data sources through ODBC" on page 306
- "Registering nicknames for ODBC data source tables and views" on page 314

Related reference:

- "ALTER USER MAPPING statement" in the *SQL Reference, Volume 2*

Registering nicknames for ODBC data source tables and views

Registering nicknames for ODBC data source tables and views is part of the larger task of adding ODBC data sources to federated servers.

For each ODBC data source server that you define, register a nickname for each table or view that you want to access. Use these nicknames, instead of the names of the data source objects, when you query the ODBC data sources.

For example, suppose that you define the nickname *cust_europe* to represent a Microsoft SQL Server table called *italy* with a schema name of *customers*. The SQL statement SELECT * FROM *cust_europe* is allowed from the federated server. However, the statement SELECT * FROM *server_name."customers"."italy"* is not allowed.

In addition to registering nicknames for ODBC data source tables and views, you can also register nicknames for remote system tables.

If your ODBC data source does not support schemas, omit the schema from the CREATE NICKNAME statement.

The federated database relies on catalog statistics for nicknamed objects to optimize query processing. These statistics are gathered when you register a nickname for a data source object by using the CREATE NICKNAME statement. The federated database verifies the presence of the object at the data source, and then attempts to gather existing data source statistical data. Information that is useful to the optimizer is read from the data source catalogs and placed into the global catalog on the federated server. Because some or all of the data source catalog information might be used by the optimizer, update statistics (using the data source command that is equivalent to the DB2 **RUNSTATS** command) at the data source before you register a nickname.

Restrictions:

When you create a nickname on an ODBC data source that contains indexes, the ODBC wrapper does not record the index information in the federated database system catalog. You must create index specifications for the table by using the CREATE INDEX statement with the SPECIFICATION ONLY clause.

Procedure:

To register a nickname, issue the CREATE NICKNAME statement.

For example:

```
CREATE NICKNAME odbc_name FOR server_name."remote_schema"."remote.table"
```

Nicknames can be up to 128 characters in length.

Repeat this step for each ODBC table or view for which you want to create a nickname.

When you create the nickname, DB2 will use the connection to query the data source catalog tables. This query tests your connection to the ODBC data source by using the nickname. If the connection does not work, you will receive an error message.

Related tasks:

- "Creating index specifications for data source objects" in the *Federated Systems Guide*

Related reference:

- "RUNSTATS Command" in the *Command Reference*
- "CREATE NICKNAME statement" in the *SQL Reference, Volume 2*
- "CREATE NICKNAME statement - Examples for ODBC wrapper" on page 315

CREATE NICKNAME statement - Examples for ODBC wrapper

This topic provides an example that shows you how to use the CREATE NICKNAME statement to register a nickname for an ODBC data source table or view that you want to access.

This example shows how to specify a remote object for the ODBC data source under which the nickname is assigned:

```
CREATE NICKNAME cust_europe FOR server_name."customers"."italy"
```

cust_europe

A unique nickname for the table or view. The nickname must be unique within the schema.

Note: The nickname is a two-part name that includes the schema and the nickname. If you omit the schema when you register the nickname, the schema of the nickname will be the authentication ID of the user who registers the nickname.

server_name."customers"."italy"
A three-part identifier for the remote object.
- *server_name* is the name that you assigned to the ODBC database server in the CREATE SERVER statement.
- *customers* is the name of the remote schema to which the table or view belongs. If your ODBC data source does not support schemas, omit the schema from the CREATE NICKNAME statement.
- *italy* is the name of the remote table or view which you want to access.

ODBC data source objects might be case sensitive. Enclose both the remote schema name and the remote table name in quotation marks. Otherwise, DB2 folds these names to uppercase.

Related tasks:
- "Registering nicknames for ODBC data source tables and views" on page 314

Related reference:
- "CREATE NICKNAME statement" in the *SQL Reference, Volume 2*

Accessing Excel data using the ODBC wrapper

You can access Microsoft Excel workbooks with the ODBC wrapper by using the Excel ODBC driver. The Excel application does not need to be installed on the federated server. The Excel ODBC driver is automatically installed with Microsoft Windows.

With the ODBC wrapper and the Excel ODBC driver, you can access data from any of the worksheets within a workbook. The Excel ODBC driver interprets a workbook as a database and interprets each worksheet within the workbook as a table.

The Excel ODBC driver supports earlier versions of Excel workbooks even if the version of Excel application that produced the workbooks is no longer supported. For example, Microsoft no longer supports worksheets created in Excel Version 4.0, but the driver supports Excel worksheets that were created in that version.

Prerequisites:

The Excel ODBC driver must be on the federated server.

The federated server must be able to open and read the worksheets in the Excel workbook to retrieve the data. Therefore, the Excel workbooks must be on the same computer as the DB2 federated server or on a accessible mapped network drive.

Restrictions:
- The ODBC wrapper cannot access a worksheet when the workbook is already opened by a user or an application in the read/write mode. However, if the ODBC wrapper opens the workbook before a user or an application opens the workbook, the user or application can open the workbook in read-only mode.
- The Excel ODBC driver expects that the first nonblank row contains the labels for the worksheet columns. You must insert a row of column labels in the worksheet if the worksheet does not have the labels.

- Because the Excel ODBC driver is only available for Windows operating systems, you can use the ODBC wrapper to access Excel data only on federated servers that run the Windows operating system.
- You can perform insert and update operations on Excel worksheets, but you cannot perform delete operations. The Excel ODBC driver does not support delete operations. To delete data from the worksheet, you must open the worksheet in Excel to make the changes.

Procedure:

To access Excel worksheets with the ODBC wrapper:

1. Ensure that the Excel workbook that you want to access is either on the federated server or on an accessible mapped network drive.
2. Methods of accessing Excel data:
   - If necessary, change the layout of the data in the Excel worksheets to adhere to the Excel ODBC driver requirements.
   - If necessary, create any named ranges that you want to access.
3. Create a system DSN for the workbook that you want to access. You can use the ODBC Data Source Administrator to configure the system DSN. The name that was specified when you created the system DSN is assigned as the value for the NODE option in the CREATE SERVER statement.
4. Issue the CREATE WRAPPER statement.

   For example:

   ```
   CREATE WRAPPER odbc
   ```
5. Specify the location of the workbook by registering a server object in the federated database system catalog. For the ODBC wrapper, you need a server object for each DSN. The DSN is associated with the workbook when the Excel ODBC driver is used.

   To specify the location of the workbook, issue the CREATE SERVER statement and use the DSN as the system DSN for the NODE option.

   For example:

   ```
   CREATE SERVER compounds_workbook WRAPPER odbc
        OPTIONS (NODE 'compounds_workbook_dsn', PASSWORD 'n')
   ```

   The NODE *compounds_workbook_dsn* is the system DSN that you created. The NODE and PASSWORD options are required for the ODBC wrapper to access Excel worksheets.

   Repeat this step for each workbook that you plan to access.
6. Issue the CREATE NICKNAME statement to create a nickname for the worksheet that you want to access. The syntax is:

   ```
   CREATE NICKNAME nickname FOR server_name.remote_table
   ```

   - If you created a named range to access the data, specify the name of the range as the remote_table portion of the CREATE NICKNAME statement.

     For example, if the name of the range is *testing*, the CREATE NICKNAME statement is:

     ```
     CREATE NICKNAME compounds_nickname FOR compounds_workbook.testing
     ```
   - To access the data in the entire worksheet instead of a range, you specify the name of the worksheet followed by the $ symbol.

     For example, if the name of the worksheet is *Sheet1*, the CREATE NICKNAME statement is:

     ```
     CREATE NICKNAME compounds_nick FOR compounds_workbook.Sheet1$
     ```

Repeat this step for each worksheet or named range that you want to access.

Alter the default data type mappings

When you use the ODBC wrapper, the supported data types are determined by the Excel ODBC driver. The Excel ODBC driver maps the Excel data types to ODBC data types. Then the ODBC wrapper maps the ODBC data types to DB2 data types. The DB2 data types for each column are stored in the federated database catalog table. These are referred to as *local data types*. The following table lists the default data type mappings:

Table 79. Default data type mappings between Excel and DB2 when using the ODBC wrapper

Excel data type	ODBC data type	DB2 data type
CURRENCY	SQL_NUMERIC	DECIMAL/DOUBLE
DATETIME	SQL_TIMESTAMP	TIMESTAMP
LOGICAL	SQL_BIT	SMALLINT
NUMBER	SQL_DOUBLE	DOUBLE
TEXT	SQL_VARCHAR	VARCHAR

However, the ODBC wrapper supports many different data types. You can map the default data types to other DB2 data types. Use the ALTER NICKNAME statement to change the local data types. The list of data types that are available through the ODBC wrapper is comprehensive and includes data types such as LOBs and other double-byte data types.

Attention: There is the potential of running into a data type mismatch when you alter the local data type to something other than the original mapping.

ALTER SERVER statement - Examples for ODBC wrapper to access Excel data

Through the Excel ODBC driver, the ODBC wrapper allows predicates and aggregate functions to be pushed down to the data source for processing. The driver also supports joins between worksheets. Pushing down processing to the data source can improve performance because all of the data source rows are not returned to DB2 when a query references a worksheet.

Suppose that you have defined the server *compounds_workbook* to identify a workbook that contains data that you want to access. For this workbook, you want to specify that the federated server can pushdown operations to the Excel ODBC driver for processing. The ALTER SERVER statement is:

```
ALTER SERVER compounds_workbook
     OPTIONS (ADD PUSHDOWN 'Y')
```

To take advantage of the push down capabilities, the following ODBC server options need to be set using the CREATE SERVER statement or by using the ALTER SERVER statement.

- PUSHDOWN ’Y’
- DB2_GROUP_BY ’Y
- DB2_COLFUNC ’Y’

Related concepts:

- “Methods of accessing Excel data” on page 25

Related tasks:

- “Preparing the federated server to access data sources through ODBC” on page 306

Related reference:

- “Choose the correct wrapper” on page 24

Tuning and troubleshooting the configuration to ODBC data sources

After you set up the configuration to ODBC data sources, you might want to modify the configuration to improve performance.

Obtaining ODBC traces

If you experience problems when accessing the data source, you can obtain ODBC tracing information to analyze and resolve the problems. However, activating a trace does impact system performance. You should turn off tracing after you have resolved the problems.

On federated servers that run Windows, click **Tracing** in the ODBC Data Source Administrator window.

On federated servers that run UNIX, consult the ODBC client vendor’s documentation.

Related tasks:

- “Adding ODBC data sources to a federated server” on page 305

Related reference:

- “db2set - DB2 Profile Registry Command” in the *Command Reference*

Chapter 18. Configuring access to OLE DB data sources

This chapter explains how to configure your federated server to access data that is stored in OLE DB data sources. You can configure access to OLE DB data sources by using the DB2 Control Center or by issuing SQL statements.

This chapter lists the tasks that you need to perform and contains examples of the SQL statements that you need to configure the federated server.

Adding OLE DB data sources to a federated server

Microsoft OLE DB is a set of OLE/COM interfaces that provide applications with uniform access to data that is stored in diverse information sources. The OLE DB component DBMS architecture defines OLE DB consumers and OLE DB providers. An OLE DB consumer is any system or application that consumes OLE DB interfaces. An OLE DB provider is a component that exposes OLE DB interfaces.

The OLE DB wrapper enables you to access OLE DB providers that are compliant with Microsoft OLE DB 2.0 (or later).

You use the OLE DB wrapper to create table functions. You cannot use the wrapper to create nicknames on data source tables and views.

To configure the federated server to access OLE DB data sources, you must provide the federated server with information about the OLE DB providers.

You can configure access to OLE DB data sources through the DB2 Command Center or through the DB2 command line.

Prerequisites:

- DB2 Information Integrator must be installed on a server that will act as the federated server
- A federated database must exist on the federated server

Restriction:

The OLE DB wrapper is used only to assist in registering user-defined OLE DB external table functions. Unlike other wrappers, the OLE DB wrapper does not use nicknames to access data that is stored in data sources.

Procedure:

To add OLE DB data sources to a federated server:

1. Register the wrapper.
2. Register the server definition.
3. Create the user mappings.

After you configure access to the OLE DB data source, use the CREATE FUNCTION statement to register a user-defined OLE DB external table function in the federated database.

Related concepts:

- "DB2 Information Integrator installation process - overview" in the *IBM DB2 Information Integrator Installation Guide for Linux, UNIX, and Windows*
- "Fast track to configuring your data sources" on page 55

Related tasks:

- "Creating a federated database" on page 51
- "Registering the OLE DB wrapper" on page 322

Registering the OLE DB wrapper

Registering the OLE DB wrapper is part of the larger task of adding OLE DB data sources to a federated server.

You must register a wrapper to access OLE DB data sources. Wrappers are used by federated servers to communicate with and retrieve data from data sources. Wrappers are implemented as a set of library files.

Procedure:

To register a wrapper, issue the CREATE WRAPPER statement and specify the default name for the wrapper.

For example:

```
CREATE WRAPPER OLEDB
```

Recommendation: Use the default wrapper name called OLEDB. When you register the wrapper that uses the default name, the federated server automatically takes the default library name that is associated with that wrapper.

If the wrapper name conflicts with an existing wrapper name in the federated database, you can substitute the default wrapper name with a name you choose. If you use a name that is different from the default name, you must include the LIBRARY parameter in the CREATE WRAPPER statement.

For example, to register a wrapper with the name `oledb_wrapper` on the federated server that uses the Windows operating system, issue the following statement:

```
CREATE WRAPPER oledb_wrapper LIBRARY 'db2oledb.dll'
```

The name of the wrapper library file that you specify depends on the operating system of the federated server. See the list of OLE DB wrapper library files for the correct name to specify in the CREATE WRAPPER statement.

The next task in this sequence of tasks is registering the server definitions for the OLE DB wrapper.

Related tasks:

- "Registering the server definitions for an OLE DB data source" on page 323

Related reference:

- "OLE DB wrapper library files" on page 323
- "CREATE WRAPPER statement" in the *SQL Reference, Volume 2*

OLE DB wrapper library files

The following table lists the directory paths and library file names for the OLE DB wrapper.

When you install DB2 Information Integrator, the library file is added to the directory path listed in the table.

When you register a wrapper, specify only the library file name that is listed in the table.

Table 80. OLE DB client library locations and file names

Operating system	Directory path	Wrapper library file
Windows	%DB2PATH%\bin	db2oledb.dll

%DB2PATH% is the environment variable that is used to specify the directory path where DB2 Information Integrator is installed on Windows. The default Windows directory path is `C:\Program Files\IBM\SQLLIB`.

Related tasks:

- "Registering the OLE DB wrapper" on page 322

Registering the server definitions for an OLE DB data source

Registering the server definitions for an OLE DB data source is part of the larger task of adding OLE DB data sources to federated servers.

In the federated database, you must define each OLE DB data source server that you want to access.

Procedure:

To register a server definition for an OLE DB data source:

Issue the CREATE SERVER statement.

For example:

```
CREATE SERVER server_name WRAPPER OLEDB
      OPTIONS (CONNECTSTRING 'Provider=Microsoft.Jet.OLEDB.4.0;
                          Data Source=c:\msdasdk\bin\oledb\nwind.mdb')
```

The next task in this sequence of tasks is creating a user mapping for an OLE DB data source.

Related tasks:

- "Creating a user mapping for an OLE DB data source" on page 324

Related reference:

- "CREATE SERVER statement" in the *SQL Reference, Volume 2*
- "CREATE SERVER statement - Examples for OLE DB wrapper" on page 324

CREATE SERVER statement - Examples for OLE DB wrapper

This topic provides an example that shows you how to use the CREATE SERVER statement to register servers for the OLE DB wrapper.

The following example shows a CREATE SERVER statement:

```
CREATE SERVER Nwind WRAPPER OLEDB
OPTIONS (CONNECTSTRING 'Provider=Microsoft.Jet.OLEDB.4.0;
                Data Source=c:\msdasdk\bin\oledb\nwind.mdb',
          COLLATING_SEQUENCE 'Y')
```

Nwind A name that you assign to the OLE DB data source. This name must be unique. Duplicate server names are not allowed.

WRAPPER *OLEDB*
The wrapper name that you specified in the CREATE WRAPPER statement.

CONNECTSTRING *'Provider=Microsoft.Jet.OLEDB.4.0; Data Source=c:\msdasdk\bin\oledb\nwind.mdb'*
Provides initialization properties that are needed to connect to a data source.

The string contains a series of keyword and value pairs that are separated by semicolons. The equal sign (=) separates each keyword and its value. Keywords are the descriptions of the OLE DB initialization properties (property set DBPROPSET_DBINT) or provider-specific keywords.

For the complete syntax and semantics of the CONNECTSTRING option, see the *Microsoft OLE DB 2.0 Programmer's Reference and Data Access SDK*, Microsoft Press, 1998.

COLLATING_SEQUENCE *'Y'*
Specifies whether the data source uses the same collating sequence as the DB2 for UNIX and Windows collating sequence.

Valid values are 'Y' (the data source uses the DB2 for UNIX and Windows collating sequence) and 'N' (the data source uses a collating sequence that is different from the DB2 for UNIX and Windows collating sequence). The default value is 'N'.

Related tasks:

- "Registering the server definitions for an OLE DB data source" on page 323

Related reference:

- "CREATE SERVER statement" in the *SQL Reference, Volume 2*

Creating a user mapping for an OLE DB data source

Creating a user mapping for an OLE data source is part of the larger task of adding OLE data sources to federated servers.

When you attempt to access an OLE data source, the federated server establishes a connection to the data source using a user ID and password that are valid for that data source. You must define an association (a user mapping) between each federated server user ID and password and the corresponding data source user ID and password. Create a user mapping for each user ID that will access the federated system to send distributed requests.

Procedure:

To map a local user ID to the OLE data source user ID and password, issue a CREATE USER MAPPING statement.

For example:

```
CREATE USER MAPPING FOR userid SERVER server_name
      OPTIONS (REMOTE_AUTHID 'remote_id', REMOTE_PASSWORD 'remote_password')
```

| If the length of either the password on the OLE DB data source or the password
| on the federated server is less then eight characters, SQL statements that access the
| OLE DB data source will fail. The following error message appears:

```
SQL30082N Attempt to establish connection failed with security reason "15"
   ("PROCESSING FAILURE"). SQLSTATE=08001
```

| To avoid this problem, change either the OLE DB data source password or the
| password on the federated server to eight or more characters.

Related reference:

- "CREATE USER MAPPING statement" in the *SQL Reference, Volume 2*
- "CREATE USER MAPPING statement - Examples for OLE DB wrapper" on page 325

CREATE USER MAPPING statement - Examples for OLE DB wrapper

This topic provides examples that show you how to use the CREATE USER MAPPING statement to map a local user ID to an OLE data source user ID and password. This topic includes a complete example with required parameters and an example that shows you how to use the DB2 special register USER with the CREATE USER MAPPING statement.

Complete example:

The following example shows how to map a local user ID to an OLE data source user ID and password:

```
CREATE USER MAPPING FOR laura SERVER Nwind
      OPTIONS (REMOTE_AUTHID 'lulu', REMOTE_PASSWORD 'raiders')
```

laura The local user ID that you are mapping to a user ID that is defined at the OLE DB data source.

SERVER *Nwind*
The name of the OLE DB server that you defined in the CREATE SERVER statement.

REMOTE_AUTHID *'lulu'*
The user ID at the OLE DB server to which you are mapping *laura*. This value is case sensitive.

REMOTE_PASSWORD *'raiders'*
The password that is associated with *'lulu'*. This value is case sensitive.

Special register example:

The following example shows a CREATE USER MAPPING statement that includes the special register USER:

```
CREATE USER MAPPING FOR USER SERVER Nwind
      OPTIONS (REMOTE_AUTHID 'lulu', REMOTE_PASSWORD 'raiders')
```

You can use the DB2 special register USER to map the authorization ID of the person who is issuing the CREATE USER MAPPING statement to the data source authorization ID that is specified in the REMOTE_AUTHID user option.

Related tasks:

- "Creating a user mapping for an OLE DB data source" on page 324

Related reference:

- "CREATE USER MAPPING statement" in the *SQL Reference, Volume 2*

Chapter 19. Configuring access to Oracle data sources

This chapter explains how to configure your federated server to access data that is stored in Oracle data sources. You can configure access to Oracle data sources by using the DB2 Control Center or by issuing SQL statements.

This chapter lists the tasks that you need to perform when you configure the federated server, contains examples of the SQL statements that you need, and provides tuning and troubleshooting information for configuring the federated server.

Adding Oracle to a federated system

Adding Oracle data sources to a federated server

To configure the federated server to access Oracle data sources, you must provide the federated server with information about the data sources and objects that you want to access.

| You can configure the federated server to access Oracle data sources by using the
| DB2 Control Center or the DB2 command line. The DB2 Control Center includes a
| wizard to guide you through the steps required to configure the federated server.

Prerequisites:
- The Oracle client software installed and configured on the federated server.
- DB2 Information Integrator must be installed on a server that will act as the federated server
- A federated database must exist on the federated server

Procedure:

To add Oracle data sources to a federated server:
1. Set up and test the Oracle client configuration file.
2. Set the Oracle environment variables.
3. Register the wrapper.
4. Register the server definition.
5. Create the user mappings.
6. Test the connection to the Oracle server.
7. Register nicknames for Oracle tables, views, and synonyms.

Related concepts:
- "DB2 Information Integrator installation process - overview" in the *IBM DB2 Information Integrator Installation Guide for Linux, UNIX, and Windows*
- "Fast track to configuring your data sources" on page 55

Related tasks:
- "Creating a federated database" on page 51
- "Setting up and testing the Oracle client configuration file" on page 331

- "Registering the Oracle wrapper" on page 332
- "Registering the server definitions for an Oracle data source" on page 334
- "Creating the user mappings for an Oracle data source" on page 335
- "Testing the connection to the Oracle server" on page 337
- "Registering nicknames for Oracle tables and views" on page 338
- "Tuning and troubleshooting the configuration to Oracle data sources" on page 339
- "Setting the Oracle environment variables" on page 328
- "Checking the setup of the federated server" on page 37

Related reference:
- "Supported operating systems for DB2 Information Integrator (32-bit)" in the *IBM DB2 Information Integrator Installation Guide for Linux, UNIX, and Windows*
- "Supported operating systems for DB2 Information Integrator (64-bit)" in the *IBM DB2 Information Integrator Installation Guide for Linux, UNIX, and Windows*

Setting the Oracle environment variables

Setting the Oracle environment variables is part of the larger task of adding Oracle to a federated server.

When you install DB2 Information Integrator, the installation process attempts to set the Oracle environment variables in the `db2dj.ini` file.

The environment variables will not be set in the `db2dj.ini` file if you:
- Install the Oracle client software after the DB2 federated server is set up.
- Have not installed the Oracle client software.

The valid environment variables for Oracle are:
- ORACLE_HOME
- ORACLE_BASE (optional)
- ORA_NLS (optional)
- TNS_ADMIN (optional)
- NLS_LANG (optional)

The optional environment variables must be set manually.

Prerequisites:

A federated server that is properly set up to access your data sources. This includes the installation and configuration of any required software, such as the data source client software.

Restrictions:

See the topic: Restrictions for the db2dj.ini file

Procedure:

To set the required environment variables automatically:

1. Install and configure the client software on the DB2 federated server, if it is not already installed.
2. Set the required environment variables. You can set the environment variables automatically by running the DB2 Information Integrator installation again. From the launchpad, click **Install Products** and follow the instructions in the wizard.

To manually set the Oracle environment variables:

1. Edit the `db2dj.ini` file.
   - On federated servers running Windows, this file is located in the `sqllib\cfg` directory.
   - On federated servers running UNIX, this file is located in the `sqllib/cfg` directory.

   The `db2dj.ini` file contains configuration information about the Oracle client software installed on your federated server. If the file does not exist, you can create a new file with the name `db2dj.ini` using any text editor. In the `db2dj.ini` file you must specify the fully qualified path for the variables, otherwise you will encounter errors.
2. Set the following environment variables as necessary:

   ORACLE_HOME

   Set the ORACLE_HOME environment variable to the directory path where the Oracle client software is installed. Specify the fully qualified path for the variable, ORACLE_HOME=*oracle_home_directory*. For example, if the Oracle home directory is \usr\oracle\8.1.7, the entry in the `db2dj.ini` is ORACLE_HOME=\usr\oracle\8.1.7

   Note: If an individual user of the federated instance has the ORACLE_HOME environment variable set, the federated instance does not use that setting. The federated instance uses only the value of ORACLE_HOME that you set in the `db2dj.ini` file.

   ORACLE_BASE

   ORACLE_BASE represents the root of the Oracle client directory tree. If you set the ORACLE_BASE variable when you installed the Oracle client software, set the ORACLE_BASE environment variable on the federated server. For example:

   `ORACLE_BASE=`*`oracle_root_directory`*

   ORA_NLS*

   If your system is using multiple versions of Oracle, you must ensure that:
   - The appropriate ORA_NLS variable is set.
   - The corresponding NLS data files for the versions you are using are available.

   The location-specific data is stored in a directory specified by the ORA_NLS* environment variable. For each new version of Oracle, there is a different ORA_NLS data directory.

Table 81. Oracle ORA_NLS variable that specifies the location of locale-specific data by version.

Oracle version	Environment variable
7.2	ORA_NLS

Table 81. Oracle ORA_NLS variable that specifies the location of locale-specific data by version. (continued)

Oracle version	Environment variable
7.3	ORA_NLS32
8.x, 9.x	ORA_NLS33
10g	ORA_NLS10

For example, for UNIX federated servers that access Oracle 8.1 data sources, set the ORA_NLS33 environment variable:

```
ORA_NLS33=oracle_home_directory/ocommon/nls/admin/<data>
```

TNS_ADMIN

- On federated servers that run Windows, the Oracle client looks for the `tnsnames.ora` file in the `%ORACLE_HOME%\NETWORK\ADMIN` directory, where `%ORACLE_HOME%` is defined in the `db2dj.ini` file. If the `tnsnames.ora` file is not in the `%ORACLE_HOME%\NETWORK\ADMIN` directory, you need to set the TNS_ADMIN environment variable on the federated server.
- On federated servers that run AIX, Linux, and HP-UX, the client looks for the `tnsnames.ora` file in the `/etc` directory. If the client does not locate the `tnsnames.ora` file in the `/etc` directory, then the client looks for the file in the `$ORACLE_HOME/network/admin` directory, where `$ORACLE_HOME` is defined in `db2dj.ini` file. If client does not find the `tnsnames.ora` file, you need to set the TNS_ADMIN environment variable. You set the variable in the `db2dj.ini` file to the path where the `tnsnames.ora` file is located.

  For example, if the `tnsnames.ora` file is in the `/home/oracle` directory, you set the variable to:

  ```
  TNS_ADMIN=/home/oracle
  ```
- On federated servers that run Solaris, the client looks for the `tnsnames.ora` file in the `/var/opt/oracle` directory. If the client does not find the `tnsnames.ora` file in the `/var/opt/oracle` directory, then the client looks for the file in the `$ORACLE_HOME/network/admin` directory, where `$ORACLE_HOME` is defined in `db2dj.ini` file. If client does not find the `tnsnames.ora` file, you need to set the TNS_ADMIN environment variable. You set the variable in the `db2dj.ini` file to the path where the `tnsnames.ora` file is located.

  For example, if the `tnsnames.ora` file is in the `/home/oracle` directory, you set the variable to:

  ```
  TNS_ADMIN=/home/oracle
  ```

3. On UNIX, update the `.profile` file of the DB2 instance with the Oracle environment variable. You can do this by issuing the following command:

   ```
   export ORACLE_HOME=oracle_home_directory
   export PATH=$ORACLE_HOME/bin:$PATH
   ```

   Where `oracle_home_directory` is the directory where the Oracle client software is installed.
4. On UNIX, execute the DB2 instance `.profile` by entering:

   ```
   . $HOME/ .profile
   ```
5. To ensure that the environment variables are set on the federated server, recycle the DB2 instance. Issue the following commands to recycle the DB2 instance:

```
db2stop
db2start
```

Setting up Oracle code page conversion:

For Oracle code page conversion, you can set the optional environment variable NLS_LANG.

Each time the Oracle wrapper connects to an Oracle data source, the wrapper determines which code page value to use for that connection. If NLS_LANG is set in the `db2dj.ini` file on the federated server, then the wrapper uses the value in the `db2dj.ini` file. The `db2dj.ini` file contains configuration information about the Oracle client software installed on your federated server.

If the NLS_LANG variable is not set on the federated server, the wrapper determines the territory and the code page of the federated database. The wrapper sets NLS_LANG to the closest matching Oracle locale. If there is no closely matching locale, NLS_LANG is set to American_America.US7ASCII.

If you access a data source that contains data that uses the Chinese code page GB 18030, your federated database must use the UTF-8 code page. For Oracle data sources, the Oracle wrapper sets the Oracle NLS_LANG environment variable to:

```
NLS_LANG=Simplified Chinese_China.UTF8
```

If you are using the Oracle 9i client, change the NLS_LANG setting in your `db2dj.ini` file to Simplified Chinese_China.AL32UTF8, so that the Oracle 9i client translates the GB 18030 data into Unicode correctly. For example:

```
NLS_LANG=Simplified Chinese_China.AL32UTF8
```

See the documentation that accompanies your Oracle software for a list of valid locales.

Related tasks:
- "Registering nicknames for Oracle tables and views" on page 338

Related reference:
- "Restrictions for the db2dj.ini file" on page 59
- "Restrictions for the db2dj.ini file" on page 59

Setting up and testing the Oracle client configuration file

Setting up and testing the Oracle client configuration file is part of the larger task of adding Oracle data sources to federated servers.

The client configuration file is used to connect to Oracle databases, using the client libraries that are installed on the federated server. This file specifies the location of each Oracle database server and type of connection (protocol) for the database server. The default name for the Oracle client configuration file is `tnsnames.ora`.

Procedure:

To set up and test the Oracle client configuration file:

1. Use the Oracle NET8/NET Configuration utility that comes with the Oracle client software.

See the installation documentation from Oracle for more information about using this utility. Within the `tnsnames.ora` file, the SID (or SERVICE_NAME) is the name of the Oracle instance, and the HOST is the host name where the Oracle server is located.

The directory in which the `tnsnames.ora` file is created depends on the operating system running on your federated server.

- On UNIX operating systems, the default path and name of this file is `$ORACLE_HOME/network/admin`.
- On Windows operating systems, the default path and name of this file is `%ORACLE_HOME%\NETWORK\ADMIN`.

2. If you want to place the `tnsnames.ora` file in a path other than the default search path, set the TNS_ADMIN environment variable to specify the file location.
   a. Edit the `db2dj.ini` file that is located in the `sqllib/cfg` directory, and set the TNS_ADMIN environment variable:
   ```
   TNS_ADMIN=x:/path/
   ```
   b. Issue the following commands to recycle the DB2 instance and ensure that the environment variable is set in the program:
   ```
   db2stop
   db2start
   ```
3. Test the connection by using the Oracle **sqlplus** tool to ensure that the client software is able to connect to the Oracle server.

The next task in this sequence of tasks is registering the Oracle wrapper.

Related tasks:

- "Registering the Oracle wrapper" on page 332
- "Tuning and troubleshooting the configuration to Oracle data sources" on page 339

Registering the Oracle wrapper

Registering the Oracle wrapper is part of the larger task of adding Oracle data sources a federated server.

You must register a wrapper to access Oracle data sources. Wrappers are used by federated servers to communicate with and retrieve data from data sources. Wrappers are implemented as a set of library files.

Procedure:

To register a wrapper, issue the CREATE WRAPPER statement and specify the default name for the wrapper.

For example:

```
CREATE WRAPPER NET8
```

Recommendation: Use the default wrapper name called NET8. When you register the wrapper by using one of the default names, the federated server automatically takes the default library name associated with that wrapper.

If the wrapper name conflicts with an existing wrapper name in the federated database, you can substitute the default wrapper name with a name that you choose. If you use a name that is different from one of the default names, you must include the LIBRARY parameter in the CREATE WRAPPER statement.

For example, to register a wrapper with the name `oracle_wrapper` on the federated server that uses the AIX operating system, issue the following statement:

```
CREATE WRAPPER oracle_wrapper LIBRARY 'libdb2net8.a'
```

The name of the wrapper library file that you specify depends on the operating system of the federated server. See the list of Oracle wrapper library files for the correct name to specify in the CREATE WRAPPER statement.

The next task in this sequence of tasks is registering the server definitions for the Oracle wrapper.

Related tasks:
- "Registering the server definitions for an Oracle data source" on page 334

Related reference:
- "Oracle wrapper library files" on page 333
- "CREATE WRAPPER statement" in the *SQL Reference, Volume 2*

Oracle wrapper library files

The following table lists the directory paths and library file names for the Oracle wrapper.

When you install DB2 Information Integrator, 3 library files are added to the directory path listed in the table. For example, if the federated server is running on AIX, the wrapper library files added to the directory path are `libdb2net8.a`, `libdb2net8F.a`, and `libdb2net8U.a`.

When you register a wrapper, specify only the library file name that is listed in the table.

Table 82. Oracle wrapper library locations and file names

Operating system	Directory path	Wrapper library file
AIX	/usr/opt/db2_08_01/lib/	libdb2net8F.a (NET8)
HP-UX	/opt/IBM/db2/V8.1/lib	libdb2net8F.sl (NET8)
Linux	/opt/IBM/db2/V8.1/lib	libdb2net8F.so (NET8)
Solaris	/opt/IBM/db2/V8.1/lib	libdb2net8F.so (NET8)
Windows	%DB2PATH%\bin	db2net8.dll (NET8)

%DB2PATH% is the environment variable that is used to specify the directory path where DB2 Information Integrator is installed on Windows. The default Windows directory path is `C:\Program Files\IBM\SQLLIB`.

Related tasks:
- "Registering the server definitions for an Oracle data source" on page 334

Related reference:
- "CREATE WRAPPER statement" in the *SQL Reference, Volume 2*

Registering the server definitions for an Oracle data source

Registering the server definitions for an Oracle data source is part of the larger task of adding Oracle data sources to federated servers.

In the federated database, you must define each Oracle server that you want to access. You must first locate the node name of the Oracle data source, and then use this node name when you register the server.

Procedure:

To register a server definition for an Oracle data source:

1. Locate the node name in the Oracle `tnsnames.ora` file.

   Example tnsnames.ora file:

   ```
   paris_node =
     (DESCRIPTION =
      (ADDRESS_LIST =
         (ADDRESS = (PROTOCOL = TCP)(HOST = somehost)(PORT = 1521)))
        (CONNECT_DATA = (SERVICE_NAME = ora9i.seel)))
   ```

   In this example, the node value to use in the CREATE SERVER statement is `paris_node`.

   Although the *node_name* is specified as an option in the CREATE SERVER SQL statement, it is required for Oracle data sources.
2. Issue the CREATE SERVER statement.

   For example:

   ```
   CREATE SERVER server_name TYPE oracle VERSION 8.1.7 WRAPPER net8
          OPTIONS (NODE 'node_name')
   ```

After the server definition is created, use the ALTER SERVER statement to add or drop server options.

The next task in this sequence of tasks is creating the user mappings for an Oracle data source.

Related tasks:
- "Creating the user mappings for an Oracle data source" on page 335

Related reference:
- "ALTER SERVER statement" in the *SQL Reference, Volume 2*
- "CREATE SERVER statement" in the *SQL Reference, Volume 2*
- Appendix D, "Server options for federated systems," on page 575
- "CREATE SERVER statement - Examples for Oracle wrapper" on page 334

CREATE SERVER statement - Examples for Oracle wrapper

This topic provides examples that show you how to use the CREATE SERVER statement to register servers for the Oracle wrapper. This topic includes a complete example, which shows how to register a server with required parameters, and an example with additional server options.

Complete example:

The following example shows you how to register a server definition for an Oracle wrapper by using the CREATE SERVER statement:

```
CREATE SERVER oraserver TYPE oracle VERSION 8.1.7 WRAPPER net8
      OPTIONS (NODE 'paris_node')
```

oraserver
: A name that you assign to the Oracle database server. This name must be unique. Duplicate server names are not allowed.

TYPE *oracle*
: Specifies the type of data source server to which you are configuring access. The type parameter for the `SQLNET` and `NET8` wrappers must be *oracle*.

VERSION *8.1.7*
: The version of Oracle database server that you want to access. The supported Oracle versions are 7.3.4, 8.x, and 9.x.

WRAPPER *net8*
: The name that you specified in the CREATE WRAPPER statement.

NODE *'paris_node'*
: The name of the node where the Oracle database server resides. Obtain the node name from the `tnsnames.ora` file.

  Although the node name is specified as an option in the CREATE SERVER statement, it is required for Oracle data sources.

Server option example:

When you create the server definition, you can specify additional server options in the CREATE SERVER statement. These server options include general server options and Oracle-specific server options.

DB2 assumes that all of the Oracle VARCHAR columns contain trailing blanks. If you are certain that all of the VARCHAR columns in the Oracle database do not contain trailing blanks, you can set a server option to specify that the data source use a non-blank padded VARCHAR comparison semantic.

The following example shows an Oracle server definition with this server option:

```
CREATE SERVER oraserver TYPE oracle VERSION 8.1.7 WRAPPER net8
      OPTIONS (NODE 'paris_node', VARCHAR_NO_TRAILING_BLANKS 'Y')
```

| Use the VARCHAR_NO_TRAILING_BLANKS server option when none of the
| columns contains trailing blanks. If only some of the VARCHAR columns contain
| trailing blanks, you can set an option on those specific columns with the ALTER
| NICKNAME statement.

Related tasks:

- "Registering the server definitions for an Oracle data source" on page 334

Related reference:

- "CREATE SERVER statement" in the *SQL Reference, Volume 2*

Creating the user mappings for an Oracle data source

Creating the user mappings for an Oracle data source is part of the larger task of adding Oracle data sources to federated servers.

When you attempt to access an Oracle server, the federated server establishes a connection to the data source using a user ID and password that are valid for that data source. You must define an association (a user mapping) between each federated server user ID and password and the corresponding data source user ID and password. Create a user mapping for each user ID that will access the federated system to send distributed requests to the Oracle data source.

Procedure:

To map a local user ID to the Oracle server user ID and password, issue a CREATE USER MAPPING statement.

For example:

```
CREATE USER MAPPING FOR userid SERVER oraserver
      OPTIONS (REMOTE_AUTHID 'remote_id', REMOTE_PASSWORD 'remote_password')
```

The next task in this sequence of tasks is testing the connection to the Oracle server.

Related tasks:

- "Testing the connection to the Oracle server" on page 337

Related reference:

- "CREATE USER MAPPING statement" in the *SQL Reference, Volume 2*
- "CREATE USER MAPPING statement - Examples for Oracle wrapper" on page 336

CREATE USER MAPPING statement - Examples for Oracle wrapper

This topic provides examples that show you how to use the CREATE USER MAPPING statement to map a federated server user ID to an Oracle server user ID and password. This topic includes a complete example with required parameters and an example that shows you how to use the DB2 special register USER with the CREATE USER MAPPING statement.

Complete example:

The following example shows how to map a federated server user ID to an Oracle server user ID and password:

```
CREATE USER MAPPING FOR robert SERVER oraserver
      OPTIONS (REMOTE_AUTHID 'rob', REMOTE_PASSWORD 'then4now')
```

robert Specifies the local user ID that you are mapping to a user ID defined at an Oracle server.

SERVER *oraserver*
Specifies the name of the Oracle server that you defined in the CREATE SERVER statement.

REMOTE_AUTHID *'rob'*
Specifies the user ID at the Oracle database server to which you are mapping *robert*. Use single quotation marks to preserve the case of this value unless you set the FOLD_ID server option to 'U' or 'L' in the CREATE SERVER statement.

REMOTE_PASSWORD *'then4now'*

Specifies the password that is associated with *'rob'*. Use single quotation marks to preserve the case of this value unless you set the FOLD_PW server option to 'U' or 'L' in the CREATE SERVER statement.

Special register example:

The following example shows a CREATE USER MAPPING statement that includes the special register USER:

```
CREATE USER MAPPING FOR USER SERVER oraserver
      OPTIONS (REMOTE_AUTHID 'rob', REMOTE_PASSWORD 'then4now')
```

You can use the DB2 special register USER to map the authorization ID of the person who is issuing the CREATE USER MAPPING statement to the data source authorization ID that is specified in the REMOTE_AUTHID user option.

Restriction: The user ID at the Oracle data source must have been created using the Oracle **create user** command with the 'identified by' clause, instead of the 'identified externally' clause.

Related tasks:

- "Creating the user mappings for an Oracle data source" on page 335

Related reference:

- "CREATE USER MAPPING statement" in the *SQL Reference, Volume 2*

Testing the connection to the Oracle server

Testing the connection to the Oracle server is part of the larger task of adding Oracle data sources to federated servers.

You can test the connection to the Oracle server by using the server definition and user mappings that you defined.

Procedure:

To test the connection:

1. Open a pass-through session to issue an SQL SELECT statement on the Oracle system tables.

   For example:

   ```
   SET PASSTHRU remote_server_name
   SELECT count(*) FROM sys.all_tables
   SET PASSTHRU RESET
   ```

   If the SQL SELECT statement returns a count, your server definition and your user mapping are set up properly.
2. If the SQL SELECT statement returns an error, you might need to:
   - Check the Oracle server to make sure that it is configured for incoming connections.
   - Check your user mapping to make sure that the settings for the REMOTE_AUTHID and REMOTE_PASSWORD options are valid for the connections to the Oracle server. Alter the user mapping, or create another user mapping as necessary.

- Check the Oracle client software on the DB2 federated server to make sure that it is installed and configured correctly to connect to the Oracle server.
- Check the settings of your DB2 federated variables to verify that they are correct for the Oracle server. These variables include the system environment variables, the db2dj.ini variables, and the DB2 Profile Registry (db2set) variable.
- Check your server definition. If necessary, drop it and create it again.

The next task in this sequence of tasks is registering nicknames for Oracle tables and views.

Related tasks:

- "Adding Oracle data sources to a federated server" on page 327
- "Setting up and testing the Oracle client configuration file" on page 331
- "Registering nicknames for Oracle tables and views" on page 338
- "Setting the Oracle environment variables" on page 328

Related reference:

- "ALTER USER MAPPING statement" in the *SQL Reference, Volume 2*

Registering nicknames for Oracle tables and views

Registering nicknames for Oracle tables and views is part of the larger task of adding Oracle data sources to federated servers.

For each Oracle server that you define, register a nickname for each table or view that you want to access. Use these nicknames, instead of the names of the data source objects, when you query the Oracle servers.

The federated database relies on catalog statistics for nicknamed objects to optimize query processing. These statistics are gathered when you register a nickname for a data source object by using the CREATE NICKNAME statement. The federated database verifies the presence of the object at the data source, and then attempts to gather existing data source statistical data. Information that is useful to the optimizer is read from the data source catalogs and placed into the global catalog on the federated server. Because some or all of the data source catalog information might be used by the optimizer, update statistics (using the data source command that is equivalent to the DB2 **RUNSTATS** command) at the data source before you register a nickname.

Procedure:

To register a nickname, issue the CREATE NICKNAME statement.

For example:

```
CREATE NICKNAME oracle_name FOR oraserver."remote_schema"."remote.table"
```

Nicknames can be up to 128 characters in length.

Repeat this step for each Oracle table or view for which you want to create a nickname.

When you create the nickname, DB2 will use the connection to query the data source catalog. This query tests your connection to the data source by using the nickname. If the connection does not work, you will receive an error message.

Related reference:

- "RUNSTATS Command" in the *Command Reference*
- "CREATE NICKNAME statement" in the *SQL Reference, Volume 2*
- "CREATE NICKNAME statement - Examples for Oracle wrapper" on page 339

CREATE NICKNAME statement - Examples for Oracle wrapper

This topic provides an example that shows you how to use the CREATE NICKNAME statement to register a nickname for an Oracle table or view that you want to access.

This example shows how to specify a remote object for the Oracle server under which the nickname is assigned:

```
CREATE NICKNAME PARISINV FOR oraserver."france"."inventory"
```

PARISINV
: A unique nickname used to identify the Oracle table or view.

  Note: the nickname is a two-part name—the schema and the nickname. If you omit the schema when you register the nickname, the schema of the nickname will be the authorization ID of the user who is registering the nickname.

oraserver."france"."inventory"
: A three-part identifier for the remote object:
  - *oraserver* is the name that you assigned to the Oracle database server in the CREATE SERVER statement.
  - *france* is the name of the remote schema to which the table or view belongs.
  - *inventory* is the name of the remote table or view that you want to access.

The federated server folds the names of the Oracle schemas and tables to uppercase unless you enclose the names in quotation marks.

Related tasks:

- "Registering nicknames for Oracle tables and views" on page 338

Related reference:

- "CREATE NICKNAME statement" in the *SQL Reference, Volume 2*

Tuning and troubleshooting the configuration to Oracle data sources

After you set up the configuration to Oracle data sources, you can modify the configuration to improve performance.

Connectivity problems

For each HOST in the DESCRIPTION section of the `tnsnames.ora` file, you might need to update the TCP/IP `hosts` file. Whether you update this file depends on how TCP/IP is configured on your network. Part of the network must translate the remote host name that is specified in the DESCRIPTION section in the `tnsnames.ora` file to an address.

If your network has a named server that recognizes the host name, you do not need to update the TCP/IP `hosts` file. Otherwise, you need an entry for the remote host. See your network administrator to determine how your network is configured. If you need to update the `hosts` file, the file location depends on the federated server operating system:

On federated servers that run UNIX
Update the `/etc/hosts` file.

On federated servers that run Windows
Update the `x:\winnt\system32\drivers\etc\hosts` file.

Related tasks:

- "Adding Oracle data sources to a federated server" on page 327

Related reference:

- "db2set - DB2 Profile Registry Command" in the *Command Reference*

Chapter 20. Configuring access to Sybase data sources

This chapter explains how to configure your federated server to access data that is stored in Sybase data sources. You can configure access to Sybase data sources by using the DB2 Control Center or by issuing SQL statements.

This chapter lists the tasks that you need to perform when you configure the federated server, contains examples of the SQL statements that you need, and provides tuning and troubleshooting information for configuring the federated server.

Adding Sybase to a federated system

Adding Sybase data sources to a federated server

To configure the federated server to access Sybase data sources, you must provide the federated server with information about the data sources and objects that you want to access.

| You can configure the federated server to access Sybase data sources by using the
| DB2 Control Center or the DB2 command line. The DB2 Control Center includes a
| wizard to guide you through the steps required to configure the federated server.

Prerequisites:
- DB2 Information Integrator must be installed on a server that will act as the federated server
- A federated database must exist on the federated server

Procedure:

To add Sybase data sources to a federated server:
1. Set up and test the Sybase client configuration file.
2. Set the Sybase environment variables.
3. Register the wrapper.
4. Register the server definition.
5. Create the user mappings.
6. Test the connection to the Sybase server.
7. Register nicknames for Sybase tables and views.

Related concepts:
- "DB2 Information Integrator installation process - overview" in the *IBM DB2 Information Integrator Installation Guide for Linux, UNIX, and Windows*
- "Fast track to configuring your data sources" on page 55

Related tasks:
- "Creating a federated database" on page 51
- "Checking the setup of the federated server" on page 37
- "Setting up and testing the Sybase client configuration file" on page 344

- “Registering the Sybase wrapper” on page 345
- “Registering the server definitions for a Sybase data source” on page 347
- “Creating a user mapping for a Sybase data source” on page 349
- “Testing the connection to the Sybase server” on page 351
- “Registering nicknames for Sybase tables and views” on page 352
- “Tuning and troubleshooting the configuration to Sybase data sources” on page 353
- “Setting the Sybase environment variables” on page 342

Related reference:

- “Supported operating systems for DB2 Information Integrator (32-bit)” in the *IBM DB2 Information Integrator Installation Guide for Linux, UNIX, and Windows*
- “Supported operating systems for DB2 Information Integrator (64-bit)” in the *IBM DB2 Information Integrator Installation Guide for Linux, UNIX, and Windows*

Setting the Sybase environment variables

Setting the Sybase environment variables is part of the larger task of adding Sybase to a federated server.

When you install DB2 Information Integrator, the installation process attempts to set the Sybase environment variables in the `db2dj.ini` file.

The environment variables will not be set in the `db2dj.ini` file if you:

- Install the Sybase client software after the federated server is set up.
- Have not installed the Sybase client software.

The valid environment variables for Sybase are:

- SYBASE
- SYBASE_OCS (required for Sybase, Version 12 or later)
- SYBASE_CHARSET (optional)

The optional environment variable must be set manually.

Prerequisites:

A federated server that is properly set up to access your data sources. This includes the installation and configuration of any required software, such as the data source client software.

Restrictions:

See the topic: Restrictions for the db2dj.ini file

Procedure:

To set the required environment variables automatically:

1. Install and configure the client software on the federated server, if it is not already installed.

2. Set the required environment variables. You can set the environment variables automatically by running the DB2 Information Integrator installation again. From the launchpad, click **Install Products** and follow the instructions in the wizard.

To manually set the environment variables:

1. Edit the `db2dj.ini` file.
   - On federated servers running Windows, this file is located in the `sqllib\cfg` directory.
   - On federated servers running UNIX, this file is located in the `sqllib/cfg` directory.

   The `db2dj.ini` file contains configuration information about the Sybase client software installed on your federated server. If the file does not exist, you can create a new file with the name `db2dj.ini` name using any text editor. In the `db2dj.ini` file you must specify the fully qualified path for the variables, otherwise you will encounter errors.
2. Set the following environment variables as necessary:

   SYBASE

   Set the SYBASE environment variable to the directory path where the Sybase Open Client software is installed. Specify the fully qualified path for this variable. For example:

   ```
   SYBASE=sybase_home_directory
   ```

   For example, if the directory path is `D:/djxclient/sybase/V12/OCS-12_5`, the SYBASE variable that you specify is:

   ```
   SYBASE=D:/djxclient/sybase/V12
   ```

   SYBASE_OCS

   For Sybase Open Client Version 12 or later, set the SYBASE_OCS environment variable to the name of the OCS directory. Do not specify the fully qualified path. The SYBASE_OCS environment variable specifies the version and release of the Sybase Open Client that is installed:

   ```
   SYBASE_OCS=OCS-version_release
   ```

   For example, if the directory path is `D:/djxclient/sybase/V12/OCS-12_5`, the directory that you specify for the SYBASE_OCS variable is:

   ```
   SYBASE_OCS=OCS-12_5
   ```

   SYBASE_CHARSET

   Set the SYBASE_CHARSET variable to the name of the character set that you want to use. The Sybase wrapper uses the SYBASE_CHARSET to determine which character set to use. If the SYBASE_CHARSET variable is not set, then the wrapper uses the Sybase character set that matches the one specified on the code page of the federated server. If there is no matching Sybase character set, the wrapper uses the iso_1 character set. You can see a list of valid character set names in the `$SYBASE\charsets` directory. For example:

   ```
   SYBASE_CHARSET=iso_1
   ```

   `iso_1` is the name of the character set that you want to use.
3. Update the `.profile` file of the federated database instance with the SYBASE environment variable. You can do this by issuing the following commands:

```
export SYBASE=<sybase_home_directory>
export SYBASE_OCS=OCS-version_release
export PATH=$SYBASE/bin:$PATH
```

4. From the home directory, run the federated database instance `.profile` by entering:
   ```
   . .profile
   ```
5. To ensure that the environment variables are set on the federated server, recycle the federated database instance. Issue the following commands to recycle the federated database instance:
   ```
   db2stop
   db2start
   ```

The next task in this sequence of tasks is registering the Sybase wrapper.

Related tasks:
- "Registering nicknames for Sybase tables and views" on page 352

Related reference:
- "Restrictions for the db2dj.ini file" on page 59
- "Restrictions for the db2dj.ini file" on page 59

Setting up and testing the Sybase client configuration file

Setting up and testing the Sybase client configuration file is part of the larger task of adding Sybase data sources to federated servers.

The client configuration file is used to connect to Sybase using the Sybase Open Client libraries that are installed on the federated server. This file specifies the location of each Sybase SQL Server and Adaptive Server Enterprise instance and the type of connection (protocol) for the database server.

You must set up a client configuration file on each instance in the federated server that will be used to connect to Sybase. The steps that you must use to set up and test this file depend on the operating system that you are running on your federated server.

Procedure:

To set up and test the client configuration file:

On UNIX operating systems:
1. Set up the client configuration file by using the utility that comes with the Sybase Open Client software. This file is created in the $SYBASE directory. The name of the file is `interfaces`. See the Sybase documentation for more information about using this utility.
2. Make the `interfaces` file accessible to the federated database instance by using one of the following methods:
   - Copy this file to the `$HOME/sqllib` directory of the federated database instance.
   - Use the **ln** command to create a link from the `$sybase` directory to the `interfaces` file in the instance `$HOME/sqllib` directory. For example:
     ```
     ln -s -f /home/sybase/interfaces  /home/db2djinst1/sqllib
     ```

- Use the IFILE server option to specify the full path to the Sybase `interfaces` file.

3. Test the connection to ensure that the Sybase Open client software is able to connect to the Sybase server. Use an appropriate Sybase query utility, such as **isql**. For example:

```
isql -Ssybnode -Umary -I/home/db2djinst1/sqllib/interfaces
```

On Windows operating systems:

1. Set up the client configuration file by using the utility that comes with the Sybase Open Client software. This file is created in the `%SYBASE%\ini` directory. The name of the file is `sql.ini`. See the Sybase documentation for more information about using this utility.
2. Make this `sql.ini` file accessible to the federated database instance by using copying this file to the `%DB2PATH%` directory of the federated database instance. The default path is `c:\Program Files\IBM\SQLLIB`.

   Because DB2 Information Integrator uses `interfaces` as the default name for the Sybase client configuration file, rename the Windows `sql.ini` file in the `c:\Program Files\IBM\SQLLIB` directory to `interfaces`.

   Required: If you do not rename the `sql.ini` file to `interfaces`, you must use the IFILE server option when you create the server definition.
3. Test the connection to ensure that the Sybase Open client software is able to connect to the Sybase server. Use an appropriate Sybase query utility, such as **isql**. For example:

```
 isql -Ssybnode -Umary -I"c:\Program Files\IBM\SQLLIB\interfaces"
```

The next task in this sequence of tasks is registering the Sybase wrapper.

Related tasks:

- "Registering the Sybase wrapper" on page 345

Registering the Sybase wrapper

Registering the Sybase wrapper is part of the larger task of adding Sybase data sources to a federated server.

You must register a wrapper to access Sybase data sources. Wrappers are used by federated servers to communicate with and retrieve data from data sources. Wrappers are implemented as a set of library files.

Procedure:

To register a wrapper, issue the CREATE WRAPPER statement and specify the default name for the wrapper.

For example:

```
CREATE WRAPPER CTLIB
```

Recommendation: Use the default wrapper name called CTLIB. When you register the wrapper by using the default name, the federated server automatically takes the default library name that is associated with that wrapper name.

If the wrapper name conflicts with an existing wrapper name in the federated database, you can substitute the default wrapper name with a name you choose. If you use a name that is different from the default name, you must include the LIBRARY parameter in the CREATE WRAPPER statement.

For example, to register a wrapper with the name `sybase_wrapper` on the federated server that uses the AIX operating system, issue the following statement:

```
CREATE WRAPPER sybase_wrapper LIBRARY 'libdb2ctlib.a'
```

The name of the wrapper library file that you specify depends on the operating system of the federated server. See the list of Sybase wrapper library files for the correct name to specify in the CREATE WRAPPER statement.

The next task in this sequence of tasks is registering the server definitions for the Sybase wrapper.

Related tasks:

- "Registering the server definitions for a Sybase data source" on page 347

Related reference:

- "Sybase wrapper library files" on page 346
- "CREATE WRAPPER statement" in the *SQL Reference, Volume 2*

Sybase wrapper library files

The following table lists the directory paths and library file names for the Sybase wrapper.

When you install DB2 Information Integrator, 3 library files are added to the directory path listed in the table. For example, if the federated server is running on AIX, the wrapper library files added to the directory path are `libdb2ctlib.a`, `libdb2ctlibF.a`, and `libdb2ctlibU.a`.

When you register a wrapper, specify only the library file name that is listed in the table.

Table 83. Sybase wrapper library locations and file names

Operating system	Directory path	Wrapper library file
AIX	/usr/opt/db2_08_01/lib/	libdb2ctlib.a (CTLIB)
HP-UX	/opt/IBM/db2/V8.1/lib	libdb2ctlib.sl (CTLIB)
Linux	/opt/IBM/db2/V8.1/lib	libdb2ctlib.so (CTLIB)
Solaris	/opt/IBM/db2/V8.1/lib	libdb2ctlib.so (CTLIB)
Windows	%DB2PATH%\bin	db2ctlib.dll (CTLIB)

%DB2PATH% is the environment variable that is used to specify the directory path where DB2 Information Integrator is installed on Windows. The default Windows directory path is `C:\Program Files\IBM\SQLLIB`.

Related tasks:

- "Registering the Sybase wrapper" on page 345

Registering the server definitions for a Sybase data source

Registering the server definitions for a Sybase data source is part of the larger task of adding Sybase data sources to federated servers.

In the federated database, you must define each Sybase server that you want to access.

Restriction:

If you use the DB2 Control Center to register the server definitions for a Sybase data source, the Sybase client configuration file, `interfaces`, must be in the default directory:

- On federated servers that run UNIX, default directory is `$HOME/sqllib/`
- On federated servers that run Windows, default directory is `c:\Program Files\IBM\SQLLIB`

Procedure:

You can register a server definition from the DB2 Control Center or the DB2 command line:

- To do this task from the DB2 Control Center, use the Federated Objects wizard or right-click the **Server Definitions** folder and click **Create**. The Discover tool retrieves the node names for the Sybase servers. You must specify the information for the DBNAME server option to register the server definition.
- To do this task from the DB2 command line, use the CREATE SERVER statement:
  1. Locate the node name in the Sybase `interfaces` file.

     Example interfaces file on UNIX operating systems:

     ```
     sybase119
     query tcp ether anaconda 4100
     ```

     Example interfaces file on Windows NT or Windows 2000 operating systems:

     ```
     [sybase119]
     query=TCP,anaconda,4100
     ```

     In these examples, the node name is sybase119. The node name is followed by the type of connection (TCP/IP) and the host name (anaconda).

     Although the node name is specified as an option in the CREATE SERVER statement, it is required for Sybase data sources.
  2. Issue the CREATE SERVER statement.

     For example:

     ```
     CREATE SERVER server_name TYPE SYBASE
         VERSION 12.0 WRAPPER CTLIB
         OPTIONS (NODE 'sybnode', DBNAME 'sybdb')
     ```

After the server definition is created, use the ALTER SERVER statement to add or drop server options.

Important: If you did not rename the `sql.ini` file to `interfaces` when you set up the Sybase client configuration file, you must use the IFILE server option when you register the server definition.

The next task in this sequence of tasks is creating a user mapping for a Sybase data source.

Related tasks:

- "Creating a user mapping for a Sybase data source" on page 349

Related reference:

- "ALTER SERVER statement" in the *SQL Reference, Volume 2*
- "CREATE SERVER statement" in the *SQL Reference, Volume 2*
- Appendix D, "Server options for federated systems," on page 575
- "CREATE SERVER statement - Examples for Sybase wrapper" on page 348

CREATE SERVER statement - Examples for Sybase wrapper

This topic provides examples that show you how to use the CREATE SERVER statement to register server definitions for the Sybase wrapper. This topic includes a complete example, which shows how to register a server definition with required parameters, and an example with additional server options.

Complete example:

The following example shows you how to register a server definition for a Sybase wrapper by issuing the CREATE SERVER statement:

```
CREATE SERVER SYBSERVER TYPE SYBASE VERSION 12.0 WRAPPER CTLIB
      OPTIONS (NODE 'sybnode', DBNAME 'sybdb')
```

SYBSERVER
: A name that you assign to the Sybase server. This name must be unique. Duplicate server names are not allowed.

TYPE *SYBASE*
: Specifies Sybase as the type of data source to which you are configuring access. The TYPE parameter for the CTLIB wrapper must be *SYBASE*.

VERSION *12.0*
: | The version of the Sybase database server software that you want to
| access. The supported versions are 11, 11.5, 11.9, 12, and 12.5.

WRAPPER *CTLIB*
: The wrapper name that you specified in the CREATE WRAPPER statement.

NODE *'sybnode'*
: The name of the node where *SYBSERVER* resides. Obtain the node name from the `interfaces` file. This value is case sensitive.

  Although the node name is specified as an option in the CREATE SERVER statement, it is required for Sybase data sources.

DBNAME *'sybdb'*
: The name of the Sybase database that you want to access. Obtain this name from the Sybase server. This value is case sensitive.

  Although the database name is specified as an option in the CREATE SERVER statement, it is required for Sybase data sources.

Important: If you do not rename the `sql.ini` file to `interfaces` when you set up the Sybase client configuration file, you must use the IFILE server option when you register the server definition.

Server option examples:

When you register the server, you can specify additional server options in the CREATE SERVER statement. These server options include general server options and Sybase-specific server options.

The following example shows how to use the TIMEOUT server option when you register a server definition on a federated server that runs UNIX:

```
|   CREATE SERVER SYBSERVER TYPE SYBASE
|          VERSION 12.0 WRAPPER CTLIB
|           OPTIONS (NODE 'sybnode', DBNAME 'sybdb',
|            TIMEOUT '60')
```

The timeout value is the number of seconds that the wrapper waits for a response from the Sybase server. Use the TIMEOUT option to avoid deadlocks on transactions.

The following example shows how to use the IFILE server option when you register a server definition on a federated server that runs Windows:

```
|   CREATE SERVER SYBSERVER TYPE SYBASE
|          VERSION 12.0 WRAPPER CTLIB
|           OPTIONS (NODE 'sybnode', DBNAME 'sybdb',
|            IFILE 'C:\Sybase\ini\sql.ini')
```

The IFILE value is full path and name of the Sybase Open Client `interfaces` file. Use this server option if you did not copy or link the `sql.ini` file as `$SQLLIB\interfaces` (on UNIX systems) or as `%SQLLIB%/interfaces` (on Windows operating systems).

The additional Sybase-specific server options are:
- LOGIN_TIMEOUT
- PACKET_SIZE

Related tasks:
- "Registering the server definitions for a Sybase data source" on page 347

Related reference:
- "CREATE SERVER statement" in the *SQL Reference, Volume 2*
- Appendix D, "Server options for federated systems," on page 575

Creating a user mapping for a Sybase data source

Creating a user mapping for a Sybase data source is part of the larger task of adding Sybase data sources to federated servers.

When you attempt to access a Sybase server, the federated server establishes a connection to the data source using a user ID and password that are valid for that data source. You must define an association (a user mapping) between each federated server user ID and password and the corresponding data source user ID and password. Create a user mapping for each user ID that will access the federated system to send distributed requests to the Sybase data source.

Procedure:

To map a local user ID to the Sybase server user ID and password, issue a CREATE USER MAPPING statement.

For example:

```
CREATE USER MAPPING FOR userid SERVER SYBSERVER
      OPTIONS (REMOTE_AUTHID 'remote_id', REMOTE_PASSWORD 'remote_password')
```

The next task in this sequence of tasks is testing the connection to the Sybase server.

Related tasks:

- "Testing the connection to the Sybase server" on page 351

Related reference:

- "CREATE USER MAPPING statement" in the *SQL Reference, Volume 2*
- "CREATE USER MAPPING statement - Examples for Sybase wrapper" on page 350

CREATE USER MAPPING statement - Examples for Sybase wrapper

This topic provides examples that show you how to use the CREATE USER MAPPING statement to map a federated server user ID to a Sybase server user ID and password. This topic includes a complete example with required parameters and an example that shows you how to use the DB2 special register USER with the CREATE USER MAPPING statement.

Complete example:

The following example shows how to map a federated server user ID to a Sybase server user ID and password:

```
CREATE USER MAPPING FOR maria SERVER SYBSERVER
      OPTIONS (REMOTE_AUTHID 'mary', REMOTE_PASSWORD 'day2night')
```

maria Specifies the local user ID that you are mapping to a user ID that is defined at the Sybase server.

SERVER *SYBSERVER*
Specifies the name of the Sybase server that you defined in the CREATE SERVER statement.

REMOTE_AUTHID *'mary'*
Specifies the user ID at the Sybase server to which you are mapping *maria*. Use single quotation marks to preserve the case of this value unless you set the FOLD_ID server option to 'U' or 'L' in the CREATE SERVER statement.

REMOTE_PASSWORD *'day2night'*
Specifies the password that is associated with *'mary'*. Use single quotation marks to preserve the case of this value unless you set the FOLD_PW server option to 'U' or 'L' in the CREATE SERVER statement.

Special register example:

The following example shows a CREATE USER MAPPING statement that includes the special register USER:

```
CREATE USER MAPPING FOR USER SERVER SYBSERVER
      OPTIONS (REMOTE_AUTHID 'mary', REMOTE_PASSWORD 'day2night')
```

You can use the DB2 special register USER to map the authorization ID of the person who is issuing the CREATE USER MAPPING statement to the data source authorization ID that is specified in the REMOTE_AUTHID user option.

Related tasks:

- "Creating a user mapping for a Sybase data source" on page 349

Related reference:

- "CREATE USER MAPPING statement" in the *SQL Reference, Volume 2*

Testing the connection to the Sybase server

Testing the connection to the Sybase server is part of the larger task of adding Sybase data sources to federated servers.

You can test the connection to the Sybase server by using the server definition and user mappings that you defined.

Procedure:

To test the connection:

1. Open a pass-through session to issue an SQL SELECT statement on the Sybase system tables.

   For example:

   ```
   SET PASSTHRU remote_server_name
   SELECT count(*) FROM dbo.sysobjects
   SET PASSTHRU RESET
   ```

   Where *remote_server_name* is the name that you specified for the remote server when you registered the server definition in the CREATE SERVER statement. If the SELECT statement returns a count, your server definition and your user mapping are set up properly.
2. If the SELECT statement returns an error, you might need to:
   - Check the Sybase server to make sure that it is configured for incoming connections.
   - Check your user mapping to make sure that the settings for the REMOTE_AUTHID and REMOTE_PASSWORD options are valid for the connections to the Sybase server. Alter the user mapping, or create another user mapping as necessary.
   - Check the Sybase client software on the DB2 federated server to make sure that it is installed and configured correctly to connect to the Sybase server.
   - Check the settings of your DB2 federated variables to verify that they are correct for the Sybase server. These variables include the system environment variables, the `db2dj.ini` variables, and the DB2 Profile Registry (db2set) variable.
   - Check your server definition. If necessary, drop it and create it again.

The next task in this sequence of tasks is registering nicknames for Sybase tables and views.

Related tasks:

- "Adding Sybase data sources to a federated server" on page 341
- "Setting up and testing the Sybase client configuration file" on page 344
- "Registering the server definitions for a Sybase data source" on page 347
- "Registering nicknames for Sybase tables and views" on page 352
- "Setting the Sybase environment variables" on page 342

Related reference:

- "ALTER USER MAPPING statement" in the *SQL Reference, Volume 2*

Registering nicknames for Sybase tables and views

Registering nicknames for Sybase tables and views is part of the larger task of adding Sybase data sources to federated servers.

For each Sybase server that you define, register a nickname for each table or view that you want to access. Use these nicknames, instead of the names of the data source objects, when you query the Sybase servers.

The federated database relies on catalog statistics for nicknamed objects to optimize query processing. These statistics are gathered when you register a nickname for a data source object by using the CREATE NICKNAME statement. The federated database verifies the presence of the object at the data source, and then attempts to gather existing data source statistical data. Information that is useful to the optimizer is read from the data source catalogs and placed into the global catalog on the federated server. Because some or all of the data source catalog information might be used by the optimizer, update statistics (using the data source command that is equivalent to the DB2 **RUNSTATS** command) at the data source before you register a nickname.

Procedure:

To register a nickname, issue the CREATE NICKNAME statement.

For example:

```
CREATE NICKNAME sybase_name FOR SYBSERVER."remote_schema"."remote.table"
```

Nicknames can be up to 128 characters in length.

Repeat this step for each Sybase table or view for which you want to create a nickname.

When you create the nickname, the federated server uses the information that you specify to query the data source catalog. This query tests your connection to the data source by using the nickname. If the connection does not work, you will receive an error message.

Related reference:

- "RUNSTATS Command" in the *Command Reference*
- "CREATE NICKNAME statement" in the *SQL Reference, Volume 2*

- “CREATE NICKNAME statement - Examples for Sybase wrapper” on page 353

CREATE NICKNAME statement - Examples for Sybase wrapper

This topic provides an example that shows you how to use the CREATE NICKNAME statement to register a nickname for a Sybase table or view that you want to access.

This example shows how to specify a remote object for the Sybase server under which the nickname is assigned:

```
CREATE NICKNAME SYBSALES FOR SYBSERVER."salesdata"."europe"
```

SYBSALES
: Is a unique nickname for the Sybase table or view.

  The nickname is a two-part name—the schema and the nickname. If you omit the schema when creating the nickname, the schema of the nickname will be the authentication ID of the user creating the nickname.

SYBSERVER."salesdata"."europe"
: Is a three-part identifier for the remote object.
  - *SYBSERVER* is the name you assigned to the Sybase database server in the CREATE SERVER statement.
  - *salesdata* is the name of the remote schema to which the table or view belongs.
  - *europe* is the name of the remote table or view that you want to access.

The federated server folds the names of the Sybase schemas and tables to uppercase unless you enclose the names in quotation marks.

Related tasks:
- “Registering nicknames for Sybase tables and views” on page 352

Related reference:
- “CREATE NICKNAME statement” in the *SQL Reference, Volume 2*

Tuning and troubleshooting the configuration to Sybase data sources

After you set up the configuration to Sybase data sources, you might want to modify the configuration to improve performance.

Resolving the sp_helpindex error

The federated system relies on one of the Sybase catalog stored procedures, `sp_helpindex`. If you receive the following SQL error, the Sybase catalog stored procedures might not be installed on the Sybase server.

```
SQL0204N "sp_helpindex" is an undefined name.
```

Have the Sybase administrator install the catalog stored procedures on the Sybase server.

Related tasks:
- “Adding Sybase data sources to a federated server” on page 341

Related reference:

- "db2set - DB2 Profile Registry Command" in the *Command Reference*

Chapter 21. Configuring access to Table-structured file data sources

This chapter explains how to configure your federated server to access data that is stored in table-structured file data sources. You can configure access to table-structured file data sources by using the DB2 Control Center or by issuing SQL statements.

This chapter:
- Explains what table-structured files are
- Lists the tasks that you need to perform
- Contains examples of the SQL statements that you need
- Lists the error messages associated with the table-structured file wrapper

What are table-structured files?

A table-structured file has a regular structure consisting of a series of records, where each record contains the same number of fields, separated by an arbitrary delimiter. Null values are represented by two delimiters next to each other.

The following example shows the contents of a file called DRUGDATA1.TXT. It contains three records, each with three fields, separated by commas:

```
234,DrugnameA,Manufacturer1
332,DrugnameB,Manufacturer2
333,DrugnameC,Manufacturer2
```

The first field is the drug's unique ID number. The second field is the name of the drug. The third field is the name of the manufacturer who produces the drug.

Related concepts:
- "Attributes of table-structured files" on page 355
- "How DB2 Information Integrator works with table-structured files" on page 356

Related tasks:
- "Adding table-structured file data sources to a federated server" on page 357

Attributes of table-structured files

Table-structured files can be sorted or unsorted. The table-structured files wrapper can search sorted data files much more efficiently than non-sorted files.

Sorted files

DRUGDATA1.TXT contains sorted records. The file is sorted by the first field, the unique ID number for the drug. This field is the primary key because it is unique for each drug. Sorted files must be sorted in ascending order.

```
234,DrugnameA,Manufacturer1
332,DrugnameB,Manufacturer2
333,DrugnameC,Manufacturer2
```

Unsorted files

DRUGDATA2.TXT contains unsorted records. There is no order to the way the records are listed in the file.

```
332,DrugnameB,Manufacturer2
234,DrugnameA,Manufacturer1
333,DrugnameC,Manufacturer2
```

Related concepts:

- "What are table-structured files?" on page 355
- "How DB2 Information Integrator works with table-structured files" on page 356

Related tasks:

- "Adding table-structured file data sources to a federated server" on page 357

How DB2 Information Integrator works with table-structured files

Using a module called a wrapper, DB2® Information Integrator can process SQL statements that query data in a table-structured file as if it were contained in an ordinary relational table or view. This enables data in a table-structured file to be joined with relational data or data in other table-structured files. This process is illustrated in Figure 27.

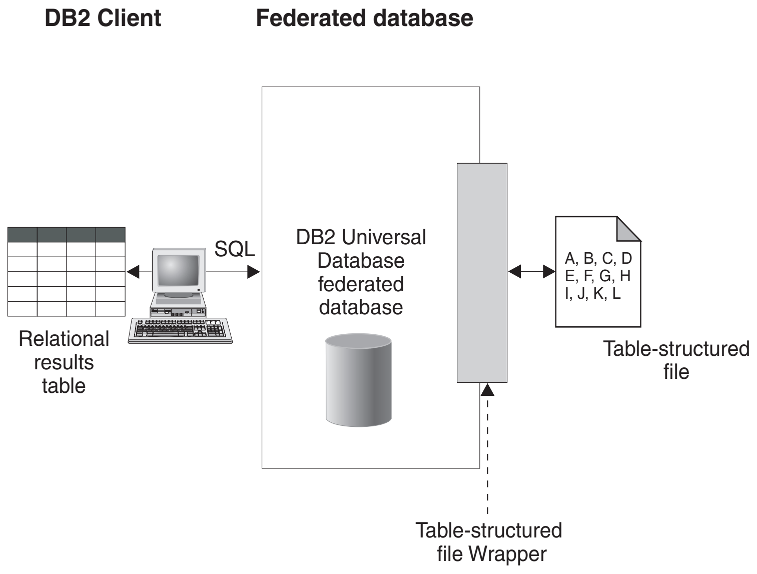


Figure 27. How the table–structured file wrapper works

For example, suppose that the table-structured file `DRUGDATA2.TXT` is located on your computer in your laboratory. Trying to query this data and match it up with other tables from other data sources that you use can be tedious.

After you register `DRUGDATA2.TXT` with DB2 Information Integrator, the file behaves as if it is a relational data source. You can now query the file together with other relational and nonrelational data sources and analyze the data together.

For example, you could run the following query:

```
SELECT * FROM DRUGDATA2 ORDER BY DCODE
```

This query produces the following results.

Dcode	Drug	Manufacturer
234	DrugnameA	Manufacturer1
332	DrugnameB	Manufacturer2
333	DrugnameC	Manufacturer2

Related concepts:

- "What are table-structured files?" on page 355
- "Attributes of table-structured files" on page 355

Related tasks:

- "Adding table-structured file data sources to a federated server" on page 357

Adding table-structured files to a federated system

Adding table-structured file data sources to a federated server

To configure the federated server to access table-structured file data sources, you must provide the federated server with information about the data sources and objects that you want to access.

| You can configure the federated server to access table-structured file data sources
| by using the DB2 Control Center or the DB2 command line. The DB2 Control
| Center includes a wizard to guide you through the steps required to configure the
| federated server.

Prerequisites:

- DB2 Information Integrator must be installed on a server that will act as the federated server
- A federated database must exist on the federated server

Procedure:

To add table-structured file data sources to a federated server:

1. Register the wrapper.
2. Register the server definition.
3. Register nicknames for the table-structured files.

Related concepts:

- "DB2 Information Integrator installation process - overview" in the *IBM DB2 Information Integrator Installation Guide for Linux, UNIX, and Windows*

Related tasks:

- "Creating a federated database" on page 51
- "Registering the table-structured file wrapper" on page 358
- "Registering the server for table-structured files" on page 359
- "Registering nicknames for table-structured files" on page 359

Related reference:

- “Supported operating systems for DB2 Information Integrator (32-bit)” in the *IBM DB2 Information Integrator Installation Guide for Linux, UNIX, and Windows*
- “Supported operating systems for DB2 Information Integrator (64-bit)” in the *IBM DB2 Information Integrator Installation Guide for Linux, UNIX, and Windows*

Registering the table-structured file wrapper

Registering the table-structured file wrapper is part of the larger task of adding table-structured file data sources to a federated server.

You must register a wrapper to access table-structured file data sources. Wrappers are used by federated servers to communicate with and retrieve data from data sources. Wrappers are implemented as a set of library files.

Procedure:

To register a wrapper, issue the CREATE WRAPPER statement with the name of the wrapper and the name of the wrapper library file.

For example, to register a wrapper with the name `laboratory_flat_files` on the federated server that uses the AIX operating system, issue the following statement:

```
CREATE WRAPPER laboratory_flat_files LIBRARY 'libdb2lsfile.a';
```

The name of the wrapper library file that you specify depends on the operating system of the federated server. See the list of Table-structured files wrapper library files for the correct name to specify in the CREATE WRAPPER statement.

The next task in this sequence of tasks is registering the server definition for the table-structured files.

Related reference:
- “Table-structured files wrapper library files” on page 358
- “CREATE WRAPPER statement” in the *SQL Reference, Volume 2*

Table-structured files wrapper library files

The following table lists the directory paths and library file names for the Table-structured files wrapper.

When you install DB2 Information Integrator, 3 library files are added to the directory path listed in the table. For example, if the federated server is running on AIX, the wrapper library files added to the directory path are `libdb2lsfile.a`, `libdb2lsfileF.a`, and `libdb2lsfileU.a`.

When you register a wrapper, specify only the library file name that is listed in the table.

Table 84. Table-structured files client library locations and file names

Operating system	Directory path	Wrapper library file
AIX	/usr/opt/db2_08_01/lib/	libdb2lsfile.a
HP-UX	/opt/IBM/db2/V8.1/lib	libdb2lsfile.sl
Linux	/opt/IBM/db2/V8.1/lib	libdb2lsfile.so
Solaris	/opt/IBM/db2/V8.1/lib	libdb2lsfile.so

Table 84. Table-structured files client library locations and file names (continued)

Operating system	Directory path	Wrapper library file
Windows	%DB2PATH%\bin	db2lsfile.dll

%DB2PATH% is the environment variable that is used to specify the directory path where DB2 Information Integrator is installed on Windows. The default Windows directory path is `C:\Program Files\IBM\SQLLIB`.

Related tasks:

- "Registering the table-structured file wrapper" on page 358

Registering the server for table-structured files

Registering the server for table-structured files is part of the larger task of adding table-structured files to a federated system. After the wrapper is registered, you must register a corresponding server.

Procedure:

To register the table-structured file server to the federated system, use the CREATE SERVER statement. For example:

```
CREATE SERVER biochem_lab WRAPPER laboratory_flat_files
```

In this example, `biochem_lab` is the name assigned to the table-structured file server. The name must be unique to the database in which the server is being registered.

The next task in this sequence of tasks is registering nicknames for table-structured files.

Related tasks:

- "Registering nicknames for table-structured files" on page 359

Related reference:

- "CREATE SERVER statement" in the *SQL Reference, Volume 2*

Registering nicknames for table-structured files

Registering nicknames for table-structured files is part of the larger task of adding table-structured files to a federated system. After you register a server, you must register a corresponding nickname. Nicknames are used when you refer to a table-structured file data source in a query.

Nicknames are associated with your table-structured file in one of two ways:

- in a fixed manner using the FILE_PATH nickname option. When this option is used, the nickname represents data from a specific table-structured file.
- with a filename specified at query time using the DOCUMENT nickname column option. When this option is used, the nickname can be used to represent data from any table-structured file whose schema matches the nickname definition.

Restrictions:

If a non-numeric field is too long for its column type, the excess data is truncated. If a decimal field in the file has more digits after the radix char than are allowed by the scale parameter of its column type, the excess data is truncated. The radix character is determined by the RADIXCHAR item of the LC_NUMERIC National Language Support category.

| The maximum line length is 10 MB (10485760 bytes)..

Procedure:

To register a nickname, use the CREATE NICKNAME statement for each table-structured file that you want to access.

There are no further tasks in this sequence of tasks.

Related tasks:
- "Adding table-structured file data sources to a federated server" on page 357
- "Specifying nickname columns for a nonrelational data source" on page 65

Related reference:
- "File access control model for the table-structured file wrapper" on page 361
- "Optimization tips and considerations for the table-structured file wrapper" on page 361
- "CREATE NICKNAME statement syntax - Table-structured file wrapper" on page 552
- "CREATE NICKNAME statement - Examples for table-structured file wrapper" on page 360

CREATE NICKNAME statement - Examples for table-structured file wrapper

This topic provides a complete example of using a CREATE NICKNAME statement to register nicknames for the table-structured file wrapper. It also includes examples for specific options.

Complete example:

The following example shows a CREATE NICKNAME statement for the table-structured file DRUGDATA1.TXT:

```
CREATE NICKNAME DRUGDATA1(Dcode Integer NOT NULL, Drug CHAR(20),
  Manufacturer CHAR(20))
FOR SERVER biochem_lab OPTIONS(FILE_PATH '/usr/pat/DRUGDATA1.TXT',
COLUMN_DELIMITER ',', SORTED 'Y', KEY_COLUMN 'DCODE', VALIDATE_DATA_FILE 'Y')
```

KEY COLUMN option examples:

These examples show that the column designated as the key is designated not nullable by adding the NOT NULL option to its definition in the nickname statement:

```
CREATE NICKNAME tox (tox_id INTEGER NOT NULL, toxicity VARCHAR(100))
FOR SERVER tox_server1
  OPTIONS (FILE_PATH'/tox_data.txt', SORTED 'Y')
```

```
CREATE NICKNAME weights (mol_id INTEGER, wt VARCHAR(100) NOT NULL)
FOR SERVER wt_server
  OPTIONS (FILE_PATH'/wt_data.txt', SORTED 'Y', KEY_COLUMN 'WT')
```

This option is case-sensitive. However, DB2 folds column names to uppercase unless the column is defined with double quotes. The following example will not work properly because the empno column will be folded to uppercase by DB2, and the empno key column will be submitted in lowercase. Thus the column designated as the key will not be found.

```
CREATE NICKNAME depart (
 empno char(6) NOT NULL)
 FOR SERVER DATASTORE
 OPTIONS(FILE_PATH'data.txt', SORTED 'Y', KEY_COLUMN 'empno');
```

Related tasks:

- "Registering nicknames for table-structured files" on page 359

Related reference:

- "CREATE NICKNAME statement syntax - Table-structured file wrapper" on page 552

File access control model for the table-structured file wrapper

The database management system will access table-structured files with the authority of the DB2 instance owner. The wrapper can only access files that can be read by this user ID (or group ID). The authorization ID of the application (the ID that establishes the connection to the federated database) is not relevant.

On DB2 Universal Database Enterprise Server Edition, any table-structured file for which a nickname has been created must be accessible with the same path name from each node. The file does not have to be on a DB2 Universal Database node as long as it can be accessed from any node with a common path.

To access a table-structured file on a mapped drive if the network has a Windows NT or Windows 2000 domain configured, the DB2 service logon account must be an account from the domain that has access to he shared folder where data files reside.

To access a table-structured file on a mapped drive if the network doesn't have a Windows NT or Windows 2000 domain, and your user logs on locally to each workstation, the DB2 service logon account should have the same user name and password as a valid user on the machine that shares that folder. That user must be on the permissions list for the shared folder with at least read access.

Related reference:

- "Access control for the Documentum wrapper" on page 188
- "File access control model for the Excel wrapper" on page 224

Optimization tips and considerations for the table-structured file wrapper

- The system can search sorted data files much more efficiently than non-sorted files.
- For sorted files, you can improve performance by specifying a value or range for the key column when submitting a query.

- Statistics for nicknames of table-structured files must be updated manually by updating the SYSSTAT and SYSCAT views.

Related reference:

- "Optimization tips for the BLAST wrapper" on page 116

Messages for the table-structured file wrapper

This section lists and describes messages you might encounter while working with the wrapper for table-structured files.

Table 85. Messages issued by the wrapper for table-structured files

Error Code	Message	Explanation
SQL0405N	The numeric literal "<literal>" is not valid because its value is out of range.	A column in the data file, or a predicate value in an SQL statement, contains a value that is out of the possible range for that data type. Correct the data file or redefine the column to a more appropriate type.
SQL0408N	A value is not compatible with the data type of it's assignment target. Target name is "<column_name>".	A column in the data file contains characters that are invalid for that data type. Correct the data file or redefine the column to a more appropriate type.
SQL0901N	The SQL statement failed because of a non-severe system error. Subsequent SQL statements can be processed. (Reason "Data source path is NULL".)	Contact IBM Software Support.
SQL0901N	The SQL statement failed because of a non-severe system error. Subsequent SQL statements can be processed. (Reason "Key Column retrieval failure".)	Contact IBM Software Support.
SQL0901N	The SQL statement failed because of a non-severe system error. Subsequent SQL statements can be processed. (Reason "STAT failed on data source. ERRNO = <error_number>".)	Ensure that you have the proper directory permissions. Ensure that the file exists.
SQL0901N	The SQL statement failed because of a non-severe system error. Subsequent SQL statements can be processed. (Reason "No column info found".)	Contact IBM Software Support.
SQL0901N	The SQL statement failed because of a non-severe system error. Subsequent SQL statements can be processed. (Reason "Unsupported operator".)	Contact IBM Software Support.

Table 85. Messages issued by the wrapper for table-structured files (continued)

Error Code	Message	Explanation
SQL1816N	Wrapper "<wrapper_name>" cannot be used to access the "type" of data source ("<type>" "") that you are trying to define to the federated database.	The server type was invalid. No server type should be specified in the CREATE SERVER statement. Remove the TYPE keyword and value and rerun it.
SQL1822N	Unexpected error code "ERRNO = <error_number>" received from data source "<server_name>". Associated text and tokens are "Unable to read file".	Check the value of the error number. Make sure that the file can be read by the DB2 instance owner. Then rerun the SQL command.
SQL1822N	Unexpected error code "Data Error" received from data source "<server_name>". Associated text and tokens are "Data source is a non-standard file".	The data source file is a directory, socket, or FIFO. Only standard files can be accessed as data source. Change the FILE_PATH option to point to a valid file and reissue the SQL command.
SQL1822N	Unexpected error code "ERRNO = <error_number>" received from data source "<server_name>". Associated text and tokens are "File open error".	The wrapper was unable to open the file. Check the error number to determine why the error occurred. Correct the problem with the data source and reissue the SQL command.
SQL1822N	Unexpected error code "Data Error" received from data source "<server_name>". Associated text and tokens are "Key column missing".	A record retrieved from the data source was missing the key field. The key column must not be null. Correct the data, or register the file with an unsorted nickname.
SQL1822N	Unexpected error code "Data Error" received from data source "<server_name>". Associated text and tokens are "File not sorted".	The file was not sorted on the key column. Do one of the following: change the KEY_COLUMN option to point to the correct column; resort the data file; or register the nickname as an unsorted nickname.
SQL1822N	Unexpected error code "Data Error" received from data source "<server_name>". Associated text and tokens are "Key exceeds definition size".	The key column field read from the data source was larger than the DB2 column definition which could cause the wrapper search routines to function incorrectly. Correct the data or correct the nickname definition, and reregister the nickname.
SQL1822N	Unexpected error code "Data Error" received from data source "<server_name>". Associated text and tokens are "Line in data file exceeds 32k".	A line in the data file exceeded the maximum line length allowed by the wrapper. The line length cannot be greater than 32768. Shorten the length of the line in the data file.
SQL1823N	No data type mapping exists for data type "<data_type>" from server "<server_name>".	The nickname was defined with an unsupported data type. Redefine the nickname using only supported data types.
SQL1881N	"<option_name>" is not a valid "<component>" option for "<object_name>".	The listed value is not a valid option for the listed object. Remove or change the invalid option then resubmit the SQL statement.

Table 85. Messages issued by the wrapper for table-structured files (continued)

Error Code	Message	Explanation
SQL1882N	The "Nickname" option "COLUMN_DELIMITER" cannot be set to "<delimiter>" for "<nickname_name>".	The column delimiter was more than one character long. Redefine the option with a single character. Then rerun the SQL statement command.
SQL1882N	The "Nickname" option "KEY_COLUMN" cannot be set to "<column_name>" for "<nickname_name>".	The column selected as the key column is not defined for this nickname. Correct the KEY_COLUMN option to be one of the sorted columns for this nickname, then reissue the SQL command.
SQL1882N	The "Nickname" option "VALIDATE_DATA_FILE" cannot be set to "<option_value>" for "<nickname_name>".	The option value was invalid. Valid values are "Y" or "N". Correct the option and register the nickname again.
SQL1883N	"<option_name>" is a required "<component>" option for "<object_name>".	A required option for the wrapper was missing from the SQL statement. Add the required option and resubmit the SQL statement.
SQL30090N	Operation invalid for application execution environment. Reason code = "21".	You attempted a pass-through session. The table-structured file wrapper does not support pass-through sessions.

Related concepts:

- "Introduction to messages" in the *Message Reference Volume 1*

Related reference:

- "SQLSTATE messages" in the *Message Reference Volume 2*

Chapter 22. Configuring access to Teradata data sources

This chapter explains how to configure your federated server to access data that is stored in Teradata data sources. You can configure access to Teradata data sources by using the DB2 Control Center or by issuing SQL statements.

This chapter lists the tasks that you need to perform when you configure the federated server, contains examples of the SQL statements that you need, and provides tuning and troubleshooting information for configuring the federated server.

Adding Teradata to a federated system

Adding Teradata data sources to a federated server

To configure the federated server to access Teradata data sources, you must provide the federated server with information about the data sources and objects that you want to access.

You can configure the federated server to access Teradata data sources by using the DB2 Control Center or the DB2 command line. The DB2 Control Center includes a wizard to guide you through the steps required to configure the federated server.

Prerequisites:

- DB2 Information Integrator must be installed on a server that will act as the federated server
- A federated database must exist on the federated server
- Teradata client software must be installed and configured on the federated server.
  - To access Teradata version V2R5, the Teradata client must support the Teradata Call-Level Interface, Version 2 (CLIv2) Release 04.07 (or later).
  - To access Teradata version V2R3 or V2R4 , the Teradata client must support the Teradata Call-Level Interface, Version 2 (CLIv2) Release 04.06 (or later).

Procedure:

To add Teradata data sources to a federated server:

1. Optional: Test the connection to the Teradata server.
2. Verify that the Teradata library is enabled for run-time linking (AIX).
3. Set the environment variables for the Teradata wrapper.
4. Register the wrapper.
5. Register the server definition.
6. Create the user mappings.
7. Test the connection from the federated server to the Teradata server.
8. Register nicknames for Teradata tables and views.

Related concepts:

- “DB2 Information Integrator installation process - overview” in the *IBM DB2 Information Integrator Installation Guide for Linux, UNIX, and Windows*
- “Fast track to configuring your data sources” on page 55

Related tasks:
- “Creating a federated database” on page 51
- “Tuning and troubleshooting the configuration to Teradata data sources” on page 378
- “Testing the connection to the Teradata server” on page 366
- “Registering the Teradata wrapper” on page 370
- “Registering the server definitions for a Teradata data source” on page 371
- “Creating the user mapping for a Teradata data source” on page 373
- “Registering nicknames for Teradata tables and views” on page 377
- “Checking the setup of the federated server” on page 37
- “Setting the Teradata environment variables” on page 368

Related reference:
- “Supported operating systems for DB2 Information Integrator (32-bit)” in the *IBM DB2 Information Integrator Installation Guide for Linux, UNIX, and Windows*
- “Supported operating systems for DB2 Information Integrator (64-bit)” in the *IBM DB2 Information Integrator Installation Guide for Linux, UNIX, and Windows*

Testing the connection to the Teradata server

Testing the connection to the Teradata server is part of the larger task of adding Teradata data sources to federated servers.

Before you create a wrapper, server definition, or user mapping, you can test the connection to the Teradata server. Test the connection first to verify that the client software is properly set and to prevent errors when you issue the CREATE WRAPPER, CREATE SERVER, and CREATE USER MAPPING statements.

You can use the Basic Teradata Query (BTEQ) utility to submit an SQL query to verify that you can connect to the Teradata server. See the Teradata documentation for more information about the BTEQ utility.

Prerequisite:

Ensure that the BTEQ utility and the Teradata Data Connector Application Program Interface (PIOM) were installed during the Teradata client software installation process.

Procedure:

To test the connection to the Teradata server:
1. Start a BTEQ utility session, and log on to the Teradata server.
2. Issue an SQL command to verify that you can successfully connect to the Teradata server. For example:
   ```
   select count(*) from dbc.tables;
   ```

   If the connection is successful, you should see the query output on the screen. For example:

```
*** Query completed. One row found. One column returned.
*** Total elapsed time was 1 second.

    Count(*)
____________
         497
```

If the connection is unsuccessful, check the Teradata client software to verify that it is properly installed and configured on the federated server.
3. Log off from the Teradata server, and end the BTEQ utility session.

The next task in this sequence of tasks is verifying that the Teradata library is enabled for run-time linking.

Related tasks:
- "Adding Teradata data sources to a federated server" on page 365
- "Verifying that the Teradata library is enabled for run-time linking (AIX)" on page 367

Verifying that the Teradata library is enabled for run-time linking (AIX)

Verifying that the Teradata library is enabled for run-time linking is part of the larger task of adding Teradata data sources to federated servers.

When you add a Teradata data source to your federated server on AIX, you must verify that run-time linking is enabled before you register wrappers or servers.

Procedure:

To verify that the Teradata library is enabled for run-time linking:
1. Go to the directory in which the libcliv2.so file resides.
   By default, the installation process places this file in the /usr/lib directory.
2. Issue the following UNIX command.
   ```
   dump - H libcliv2.so | grep libtli.a
   ```
3. Check the file names that appear on the screen.
   If the libtli.a file name appears, the Teradata library is enabled for run-time linking.
4. If the libtli.a file name does not appear, issue the following UNIX commands.
   ```
   rtl_enable libcliv2.so -F libtli.a
   mv libcliv2.so libcliv2.so.old
   mv libcliv2.so.new libcliv2.so
   chmod a+r libcliv2.so
   ```
   These commands enable run-time linking for the Teradata library.

The next task in this sequence of tasks is registering the Teradata wrapper.

Related tasks:
- "Registering the Teradata wrapper" on page 370

Setting the Teradata environment variables

Setting the Teradata environment variables is part of the larger task of adding Teradata to a federated server.

When you install DB2 Information Integrator, the installation process attempts to set the Teradata environment variables in the `db2dj.ini` file.

The environment variables will not be set in the `db2dj.ini` file if you:
- Install the Teradata client software after the DB2 federated server is set up.
- Have not installed the Teradata client software.

The valid environment variables for Teradata are:
- COPERR
- COPLIB
- TERADATA_CHARSET (optional)
- NETRACE (optional)
- COPANOMLOG (optional)

The optional environment variables must be set manually.

Prerequisites:

A federated server that is properly set up to access your data sources. This includes the installation and configuration of any required software, such as the data source client software.

Restrictions:

See the topic: Restrictions for the db2dj.ini file

Procedure:

To set the required environment variables automatically:
1. Install and configure the client software on the DB2 federated server, if it is not already installed.
2. Set the required environment variables. You can set the environment variables automatically by running the DB2 Information Integrator installation again. From the launchpad, click **Install Products** and follow the instructions in the wizard.

To manually set the environment variables:
1. Edit the `db2dj.ini` file.
   - On federated servers running Windows, this file is located in the `sqllib\cfg` directory.
   - On federated servers running UNIX, this file is located in the `sqllib/cfg` directory.

   The `db2dj.ini` file contains configuration information about the Teradata client software installed on your federated server. If the file does not exist, you can create a new file with the name `db2dj.ini` name using any text editor. In the `db2dj.ini` file you must specify the fully qualified path for the variables, otherwise you will encounter errors.

2. Set the following environment variables as necessary.

   COPERR

   Set the COPERR environment variable to the directory path where the `errmsg.txt` file resides. Specify the fully qualified path for the variable, COPERR=*teradata_lib_directory*. For example:

   ```
   COPERR=/usr/lib
   ```

   COPLIB

   Set the COPLIB environment variable to the directory path where the `libcliv2.so` file resides. Specify the fully qualified path for the variable, COPLIB=*teradata_lib_directory*. For example:

   ```
   COPLIB=/usr/lib
   ```

   The `libcliv2.so` and `errmsg.txt` files typically reside in the same directory.

   TERADATA_CHARSET

   If you do not set the TERADATA_CHARSET variable, DB2 Information Integrator detects the client character set based on the code page of the database. When this variable is set, DB2 Information Integrator uses variable value as the client character set. The value in the TERADATA_CHARSET variable is not validated, but if it is not set to the correct value, the remote data source returns an error.

   Set the TERADATA_CHARSET environment variable in the db2dj.ini file to one of the following valid character sets:

   On federated servers running UNIX:
   - HANGULKSC5601_2R4
   - KanjiEUC_0U
   - LATIN1_0A
   - LATIN9_0A
   - LATIN1252_0A
   - SCHGB2312_1T0
   - TCHBIG5_1R0
   - UTF8

   On federated servers running Windows:
   - HANGULKSC5601_2R4
   - KanjiSJIS_0S
   - LATIN1_0A
   - LATIN1252_0A
   - SCHGB2312_1T0
   - TCHBIG5_1R0
   - UTF8

3. To ensure that the environment variables are set on the federated server, recycle the DB2 instance. Issue the following commands to recycle the DB2 instance:

   ```
   db2stop
   db2start
   ```

Enabling and disabling Teradata tracing:

The Teradata NETRACE and COPANOMLOG variables are used to enable and disable Teradata tracing. You can enable Teradata tracing if you need to preserve a listing of SQL statements that are sent to the Teradata server.

To enable Teradata tracing, edit the `db2dj.ini` file and use the following settings for these environment variables:

```
NETRACE=1
COPANOMLOG=trace_file
```

The `trace_file` is the fully qualified name of the file that will contain the trace data. These variables enable the Teradata tracing facility only and do not affect the DB2 tracing.

To disable Teradata tracing, edit the `db2dj.ini` file and remove both the NETRACE and COPANOMLOG variables.

The next task in this sequence of tasks is registering the Teradata wrapper.

Related tasks:

- "Registering nicknames for Teradata tables and views" on page 377

Related reference:

- "Restrictions for the db2dj.ini file" on page 59
- "Restrictions for the db2dj.ini file" on page 59

Registering the Teradata wrapper

Registering the Teradata wrapper is part of the larger task of adding Teradata data sources to a federated server.

You must register a wrapper to access Teradata data sources. Wrappers are used by federated servers to communicate with and retrieve data from data sources. Wrappers are implemented as a set of library files.

Procedure:

To register a wrapper, issue the CREATE WRAPPER statement and specify the default name for the wrapper.

For example:

```
CREATE WRAPPER TERADATA
```

Recommendation: Use the default wrapper name called TERADATA. When you register the wrapper by using the default name, the federated server automatically takes the default library name that is associated with that wrapper name.

If the wrapper name conflicts with an existing wrapper name in the federated database, you can substitute the default wrapper name with a name you choose. If you use a name that is different from the default name, you must include the LIBRARY parameter in the CREATE WRAPPER statement.

For example, to register a wrapper with the name `tera_wrapper` on the federated server that uses the AIX operating system, issue the following statement:

```
CREATE WRAPPER tera_wrapper LIBRARY 'libdb2teradata.a';
```

The name of the wrapper library file that you specify depends on the operating system of the federated server. See the list of Teradata wrapper library files for the correct name to specify in the CREATE WRAPPER statement.

The next task in this sequence of tasks is registering the server definition for the Teradata wrapper.

Related tasks:
- "Registering the server definitions for a Teradata data source" on page 371

Related reference:
- "Teradata wrapper library files" on page 371
- "CREATE WRAPPER statement" in the *SQL Reference, Volume 2*

Teradata wrapper library files

The following table lists the directory paths and library file names for the Teradata wrapper.

When you install DB2 Information Integrator, 3 library files are added to the directory path listed in the table. For example, if the federated server is running on AIX, the wrapper library files added to the directory path are `libdb2teradata.a`, `libdb2teradataF.a`, and `libdb2teradataU.a`.

When you register a wrapper, specify only the library file name that is listed in the table.

Table 86. Teradata wrapper library locations and file names

Operating system	Directory path	Wrapper library file
AIX	/usr/opt/db2_08_01/lib/	libdb2teradata.a
HP-UX	/opt/IBM/db2/V8.1/lib	libdb2teradata.sl
Solaris	/opt/IBM/db2/V8.1/lib	libdb2teradata.so
Windows	%DB2PATH%\bin	db2teradata.dll

%DB2PATH% is the environment variable that is used to specify the directory path where DB2 Information Integrator is installed on Windows. The default Windows directory path is `C:\Program Files\IBM\SQLLIB`.

Related tasks:
- "Registering the Teradata wrapper" on page 370

Registering the server definitions for a Teradata data source

Registering the server definitions for a Teradata data source is part of the larger task of adding Teradata data sources to federated servers.

In the federated database, you must define each Teradata server that you want to access. You must first locate the node name of the Teradata data source, and then use this node name when you register the server.

Procedure:

To register a server definition for a Teradata data source:

1. Locate the node name.
   a. Find the hosts file.

      On AIX operating systems, the hosts file is /etc/hosts.

      On Windows operating systems, the hosts file is *x:*\WINNT\system32\drivers\etc\hosts. *x:* is the drive where the \WINNT directory resides.
   b. Search the hosts file for the alias of the remote server.

      This alias begins with an alphabetic string and ends with the suffix `COP`*n*. The value *n* is the number of the application processor that is associated with the Teradata communications processor.
   c. Find the line in the hosts file that contains this alias.
   d. Find the first non-numeric field on that line.

      Example hosts file:

      ```
      127.0.0.1       localhost

      9.22.5.77       nodexyz        nodexyzCOP1      # teradata server

      9.66.111.133   rtplib05.data.xxx.com aap
      9.66.111.161   rtpscm11.data.xxx.com aaprwrt
      9.66.111.161   rtpscm11.data.xxx.com accessm
      ```

      In this example, the **nodexyz** field is the node name.
2. Issue the CREATE SERVER statement.

   For example:

   ```
   CREATE SERVER server_name TYPE TERADATA VERSION 2.4 WRAPPER wrapper
         OPTIONS (NODE 'node_name')
   ```

   You must specify a server name. The name that you specify must be unique.

   You must set the TYPE parameter to *TERADATA* for all Teradata servers.

   | The Teradata wrapper supports all versions of Teradata V2R3, V2R4, and V2R5.
   | You specify the version number as two digits with a decimal point. Examples
   | of valid version numbers are 2.3, 2.4, 2.5.

   You must specify a name for the wrapper. The name that you specify must correspond to a Teradata wrapper that you registered with the CREATE WRAPPER statement.

   You must also specify the name of the node where the Teradata server resides. This node name is case sensitive.

When you register a Teradata server definition, you can specify additional server options in the CREATE SERVER statement, if required.

After you register the server definition, you can add or drop server options by issuing the ALTER SERVER statement.

The next task in this sequence of tasks is creating the user mapping for a Teradata data source.

Related tasks:

- "Creating the user mapping for a Teradata data source" on page 373

Related reference:

- "CREATE SERVER statement" in the *SQL Reference, Volume 2*

- “CREATE SERVER statement - Examples for Teradata wrapper” on page 373

CREATE SERVER statement - Examples for Teradata wrapper

This topic provides several examples that show you how to use the CREATE SERVER statement to register servers for the Teradata wrapper. This topic includes a complete example, which shows how to create a server with all required parameters, and an example with optional server options.

Complete example:

The following example shows you how to create a server definition for a Teradata wrapper by using the CREATE SERVER statement:

```
CREATE SERVER TERASERVER TYPE TERADATA
     VERSION 2.4 WRAPPER my_wrapper
     OPTIONS (NODE 'tera_node');
```

The server option *TERASERVER* specifies the name that you assign to the Teradata server. TYPE *TERADATA* specifies that you are configuring access to a Teradata data source. VERSION *2.4* is the version of the Teradata server software that you want to access. WRAPPER *my_wrapper* specifies the name of the Teradata wrapper that you registered through the CREATE WRAPPER statement. NODE *'tera_node'* is the name of the node where the Teradata server resides.

Server option example:

The following example shows a Teradata server definition with statistics for the optimizer:

```
CREATE SERVER TERASERVER1 TYPE TERADATA
    VERSION 2.4 WRAPPER WRAPPERNAME1
    OPTIONS (NODE 'tera_node1', CPU_RATIO '2.0', IO_RATIO '3.0');
```

In this example, *TERASERVER1* is the name of the Teradata server, *WRAPPERNAME1* is the wrapper name that you registered through the CREATE WRAPPER statement, and *'tera_node1'* is the name of the node where the Teradata server resides. The CPU_RATIO and IO_RATIO server options provide the following information to the optimizer:

- The CPU resources of the federated server are twice as powerful as the CPU resources of the Teradata server.
- The I/O devices of the federated server process data three times faster than the I/O devices of the Teradata server.

Related tasks:

- “Registering the server definitions for a Teradata data source” on page 371

Related reference:

- “CREATE SERVER statement” in the *SQL Reference, Volume 2*

Creating the user mapping for a Teradata data source

Creating the user mapping for a Teradata data source is part of the larger task of adding Teradata data sources to federated servers.

When you attempt to access a Teradata server, the federated server establishes a connection to the data source using a user ID and password that are valid for that data source. You must define an association (a user mapping) between each federated server user ID and password and the corresponding data source user ID and password. Create a user mapping for each user ID that will access the federated system to send distributed requests to the Teradata data source.

You must create user mappings for each Teradata server that you registered in the associated CREATE SERVER statement.

Procedure:

To map the federated user ID to the Teradata server user ID and password, issue a CREATE USER MAPPING statement.

For example:

```
CREATE USER MAPPING FOR USERID SERVER TERASERVER
      OPTIONS (REMOTE_AUTHID 'remote_id', REMOTE_PASSWORD 'remote_password')
```

Alternatively, you can create user mappings by using the Create User Mapping window of the DB2 Control Center.

The next task in this sequence of tasks is testing the connection from the federated server to the Teradata server.

Related tasks:

- "Testing the connection from the federated server to the Teradata server" on page 375

Related reference:

- "CREATE USER MAPPING statement" in the *SQL Reference, Volume 2*
- "CREATE USER MAPPING statement - Examples for Teradata wrapper" on page 374

CREATE USER MAPPING statement - Examples for Teradata wrapper

This topic provides examples that show you how to use the CREATE USER MAPPING statement to map a local federated user ID to a Teradata server user ID and password. This topic includes a complete example with all the required parameters and an example that shows you how to use the DB2 special register USER with the CREATE USER MAPPING statement.

Complete example:

The following example shows how to map a local federated user ID (*MICHAEL*) to a Teradata server user ID and password (*′mike′* and *′passxyz123′*):

```
CREATE USER MAPPING FOR MICHAEL SERVER TERASERVER
      OPTIONS (REMOTE_AUTHID 'mike', REMOTE_PASSWORD 'passxyz123')
```

The option *MICHAEL* specifies the federated user ID that you are mapping to a user ID that is defined at the Teradata server. SERVER *TERASERVER* specifies the name of the Teradata server that you defined in the CREATE SERVER statement. REMOTE_AUTHID *′mike′* is the user ID at the Teradata server to which you are mapping the local user ID called *MICHAEL*. REMOTE_PASSWORD *′passxyz123′* is the password that is associated with the REMOTE_AUTHID value of *′mike′*.

Special register example:

The following example shows a CREATE USER MAPPING statement that includes the special register USER:

```
CREATE USER MAPPING FOR USER SERVER TERASERVER
      OPTIONS (REMOTE_AUTHID 'mike', REMOTE_PASSWORD 'passxyz123')
```

You can use the DB2 special register USER to map the authorization ID of the person who is issuing the CREATE USER MAPPING statement to the data source authorization ID that is specified in the REMOTE_AUTHID user option.

Related tasks:
- "Creating the user mapping for a Teradata data source" on page 373

Related reference:
- "CREATE USER MAPPING statement" in the *SQL Reference, Volume 2*

Testing the connection from the federated server to the Teradata server

Testing the connection from the federated server to the Teradata server is part of the larger task of adding Teradata data sources to federated servers.

You can test the connection from the federated server to the Teradata server by using the server definition and the user mapping that you defined.

Procedure:

To test the connection:
1. From the DB2 command line processor, open a pass-through session to issue an SQL SELECT statement on a Teradata system table.

   For example:
   ```
   SET PASSTHRU server_name
   SELECT count(*) FROM dbc.tables
   SET PASSTHRU RESET
   ```
   If the SQL SELECT statement returns a count, your server definition and your user mapping are set up properly.
2. If the SQL SELECT statement returns an error, you might need to:
   - Check the Teradata server to make sure that it is configured for incoming connections.
   - Check your user mapping to make sure that the settings for the REMOTE_AUTHID and REMOTE_PASSWORD options are valid for the connections to the Teradata server. Alter the user mapping, or create another user mapping as necessary.
   - Check the Teradata client software on the DB2 federated server to make sure that the software is correctly installed and configured to connect to the Teradata server.
   - Check the settings of your DB2 federated variables to verify that you can access the Teradata server. These variables include the system environment variables, the db2dj.ini variables, and the DB2 Profile Registry (db2set) variable.
   - Check your server definition. If necessary, drop the server definition and create it again.

When you initiate a pass-through session to issue SQL statements on Teradata objects, you cannot submit an SQL PREPARE statement with an INTO parameter if the statement contains host variables.

The next task in this sequence of tasks is registering nicknames for Teradata tables and views.

Related tasks:
- "Adding Teradata data sources to a federated server" on page 365
- "Testing the connection to the Teradata server" on page 366
- "Registering the server definitions for a Teradata data source" on page 371
- "Registering nicknames for Teradata tables and views" on page 377
- "Setting the Teradata environment variables" on page 368

Related reference:
- "ALTER USER MAPPING statement" in the *SQL Reference, Volume 2*

Teradata nicknames on federated servers

You must create a nickname for each Teradata® table and view that you want to access on each Teradata server that you defined. Use these nicknames, instead of the names of the data source objects, when you query the Teradata servers.

The federated server connects to the Teradata data source by using the nickname that you assigned with the CREATE NICKNAME statement. The federated server then queries the data source catalog and verifies the connection to the data source. If the connection does not work, DB2® generates an error message.

The federated database relies on catalog statistics for nicknamed objects to optimize query processing. These statistics are gathered when you create a nickname for a data source object.

The federated database verifies the presence of the object at the data source, and then attempts to gather existing statistical data from that data source. Information that is useful to the optimizer is read from the data source catalogs and placed into the global catalog on the federated server. Because some or all of the data source catalog information might be used by the optimizer, update the statistics at the data source before you create a nickname. Update these statistics at the data source by using a command or utility that is equivalent to the DB2 **RUNSTATS** command.

You cannot submit an SQL INSERT, UPDATE, or DELETE statement to a nickname that references an updatable Teradata view unless that SQL statement can be completely pushed down to the Teradata data source.

Related tasks:
- "Registering nicknames for Teradata tables and views" on page 377

Related reference:
- "RUNSTATS Command" in the *Command Reference*
- "CREATE NICKNAME statement - Examples for Teradata wrapper" on page 377

Registering nicknames for Teradata tables and views

Registering nicknames for Teradata tables and views is part of the larger task of adding Teradata data sources to federated servers.

For each Teradata server that you define, register a nickname for each table and view that you want to access.

Procedure:

To register a nickname, issue the CREATE NICKNAME statement.

For example:

```
CREATE NICKNAME TERANICKNAME FOR TERASERVER."remote_schema"."remote.table"
```

Recommendation: Because the federated database uses catalog statistics for nicknamed objects to optimize query processing, update the statistics at the Teradata data source before registering a nickname. You can use a command or utility that is equivalent to the DB2 **RUNSTATS** command.

Nicknames can be up to 128 characters in length.

You can specify the NUMERIC_STRING column option when you issue the CREATE NICKNAME statement. You can also specify this column option by using the ALTER NICKNAME statement.

Related concepts:

- "Teradata nicknames on federated servers" on page 376

Related reference:

- "RUNSTATS Command" in the *Command Reference*
- "CREATE NICKNAME statement" in the *SQL Reference, Volume 2*
- "CREATE NICKNAME statement - Examples for Teradata wrapper" on page 377

CREATE NICKNAME statement - Examples for Teradata wrapper

This topic provides an example that shows you how to use the CREATE NICKNAME statement to register a nickname for a Teradata table or view that you want to access.

This example shows how to specify a remote object for the Teradata server under which the nickname is assigned:

```
CREATE NICKNAME TERASALES FOR TERASERVER."salesdata"."europe"
```

TERASALES is the unique nickname that you assign for the Teradata table or view. A nickname is a two-part name: the schema and the actual nickname. If you omit the schema when you create the nickname, DB2 creates the nickname using your authentication ID as the schema.

TERASERVER."salesdata"."europe" specifies a three-part identifier for the remote object:

- *TERASERVER* is the name that you assigned to the Teradata database server in the CREATE SERVER statement.

- *salesdata* is the name of the remote schema to which the table or view belongs.
- *europe* is the name of the remote table or view that you want to access.

Related concepts:
- "Teradata nicknames on federated servers" on page 376

Related tasks:
- "Registering nicknames for Teradata tables and views" on page 377

Related reference:
- "CREATE NICKNAME statement" in the *SQL Reference, Volume 2*

Tuning and troubleshooting the configuration to Teradata data sources

After you set up the configuration to Teradata data sources, you can change the configuration to improve performance and to eliminate potential errors.

UPDATE or DELETE operation errors on nicknames

By default, rows are not uniquely identified on Teradata data source tables. You might receive an SQL30090N, RC="21" error when you try to update or delete a nickname that is associated with a Teradata table or a Teradata view. If the SQL30090N, RC="21" error occurs, create at least one unique index on the Teradata table that is being updated or deleted, and then try the operation again.

Tuning and disabling Teradata access logging

The Teradata product provides an access logging feature that generates log entries when Teradata checks the specific security privileges of various users on one or more databases. Although access logging provides considerable and meaningful security information, this feature significantly increases processor usage and can degrade system performance.

If you need to improve system performance, evaluate the checking privilege rules that you defined for access logging. Then, terminate any unnecessary rules by defining END LOGGING statements.

For the best performance, turn off all access logging. Drop the **Teradata DBC.AccLogRules** macro and then force a trusted parallel application (TPA) reset to stop access logging completely.

See the Teradata documentation for more information.

Enabling run-time linking for libcliv2.so (AIX)

If you run the `djxlinkTeradata.sh` file to link to the Teradata shared library called `libcliv2.so`, you might receive an error message when you issue a CREATE NICKNAME statement.

An example of an error message that you might receive is:

```
DB21034E  The command was processed as an SQL statement because it was not a
valid Command Line Processor command.  During SQL processing it returned:
SQL30081N  A communication error has been detected.  Communication protocol
being used: "TCP/IP".  Communication API being used: "SOCKETS".  Location
where the error was detected: "9.112.26.28".  Communication function detecting
the error: "recv".  Protocol specific error code(s): "*", "*", "0".
SQLSTATE=08001
```

If you receive an error message, check the `/sqllib/db2dump` directory for any trap files. Trap file names begin with the letter `t` and end with a suffix of `000`. For example:

```
 t123456.000
```

Check the trace information in the trap file for any `OsCall` function references that indicate that the `OsCall` function caused the federated server to stop.

The following example shows trace information with an `OsCall` function reference that you might find in a trap file:

```
*** Start stack traceback ***

0x239690E0 OsCall + 0x28C
0x23973FB0 mtdpassn + 0x8A4
0x239795A4 mtdp + 0x208
0x2395A928 MTDPIO + 0x28C
0x239609C4 CLICON + 0xD50
0x23962350 DBCHCL + 0xC4
```

If you find an `OsCall` function reference in one of the trap files, issue the following UNIX commands:

```
cd /usr/lib
rtl_enable libcliv2.so -F libtli.a
mv libcliv2.so libcliv2.so.old
mv libcliv2.so.new libcliv2.so
chmod a+r libcliv2.so
```

These commands enable run-time linking for the `libcliv2.so` shared library.

Related tasks:

- "Adding Teradata data sources to a federated server" on page 365
- "Verifying that the Teradata library is enabled for run-time linking (AIX)" on page 367

Related reference:

- "db2set - DB2 Profile Registry Command" in the *Command Reference*

Chapter 23. Configuring access to Web services data sources

The information in this section explains how to add Web service data sources to your federated system.

The Web services wrapper and the Web services description language document

Web service providers are described by Web Services Description Language (WSDL) documents. You can use the Web services wrapper to access Web service providers. As Figure 28 shows, the Web service provider implements a service and publishes the interface to a service broker, such as UDDI. The service requester can then use the service broker to find a Web service. When the requester finds a service, the requester binds to the service provider so that the requester can use the Web service. The requester invokes the service by exchanging SOAP (simple object access protocol) messages between the requester and provider.

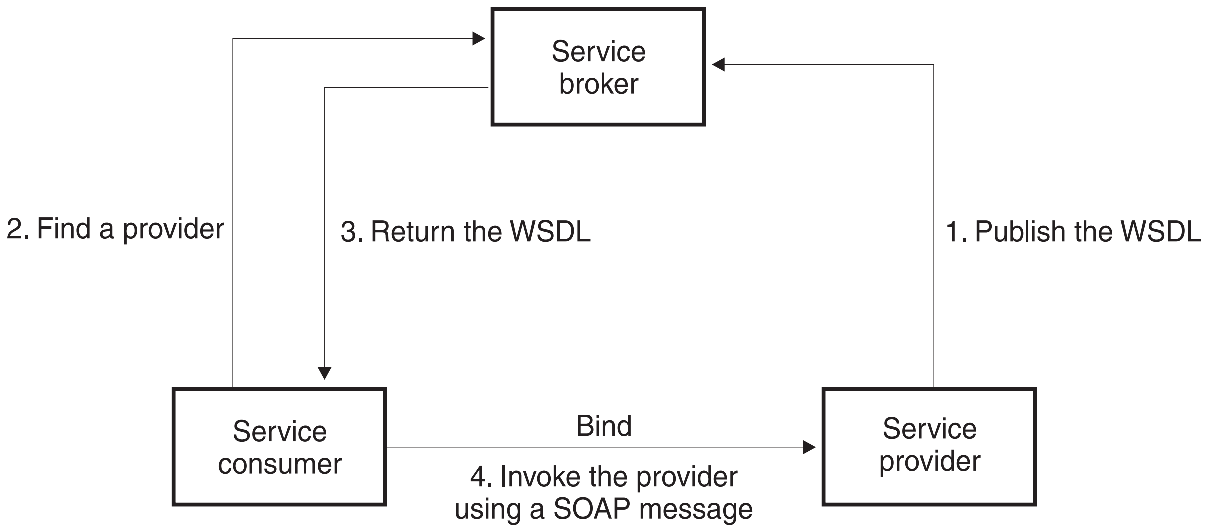


Figure 28. Web services: a service-oriented architecture

The SOAP specification defines the layout of an XML-based message. A SOAP message is contained in a SOAP envelope. The envelope consists of an optional SOAP header and a mandatory SOAP body. The SOAP header can contain information about the message, such as encryption information or authentication information. The SOAP body contains the message. The SOAP specification also defines a default encoding for programming language bindings, which is called the SOAP encoding.

The key to the Web service is the WSDL document. The WSDL document is an XML document that describes Web services in terms of the messages that it sends and receives. Messages are described by using a type system, which is typically the XML schema. A Web service operation associates a message exchange pattern with one or more messages. A message exchange pattern identifies the sequence and cardinality of messages that are sent or received, as well as who the messages are logically sent to or received from. An interface groups together operations without any commitment to the transport or wire format. A WSDL binding specifies transport and wire format details for one or more interfaces. An endpoint associates a network address with a binding. A service groups together endpoints that implement a common interface. The messages can contain document-oriented information or process-oriented information, which is also known as remote procedure calls (RPC). A WSDL document can contain one or more Web services.

The example in Figure 29 on page 383 shows the WSDL definition of a simple service that provides stock quotes. The Web service supports a single operation that is named GetLastTradePrice. The service can be accessed with the SOAP 1.1 protocol over HTTP. The request reads a ticker symbol as input, which is a string data type, and returns the price, which is a float data type. The string and float data types are predefined types in the XML schema standards. A Web service can also define data types and use those user-defined data types in messages. Predefined and user-defined XML data types map to columns of the nicknames. The complete example and the WSDL specification is at the W3C Web site.

```
<?xml version="1.0"?>
<definitions name="StockQuote"
...


<types>
       <schema targetNamespace="http://example.com/stockquote.xsd"
              xmlns="http://www.w3.org/2000/10/XMLSchema">
           <element name="TradePriceRequest">
              <complexType>
                  <all>
                      <element name="tickerSymbol" type="string"/>
                  </all>
              </complexType>
           </element>
           <element name="TradePrice">
              <complexType>
                  <all>
                      <element name="price" type="float"/>
                  </all>
              </complexType>
           </element>
       </schema>
    </types>

<message name="GetLastTradePriceInput">
...
</message>

    <portType name="StockQuotePortType">
        <operation name="GetLastTradePrice">
           <input message="tns:GetLastTradePriceInput"/>
           <output message="tns:GetLastTradePriceOutput"/>
        </operation>
    </portType>


    <binding name="StockQuoteSoapBinding"
          type="tns:StockQuotePortType">
        <soap:binding style="document"
          transport="http://schemas.xmlsoap.org/soap/http"/>
        <operation name="GetLastTradePrice">
           <soap:operation soapAction="http://example.com/GetLastTradePrice"/>
           <input>
               <soap:body use="literal"/>
           </input>
           <output>
               <soap:body use="literal"/>
           </output>
        </operation>
    </binding>

   <service name="StockQuoteService">
        <documentation>My first service</documentation>
        <port name="StockQuotePort" binding="tns:StockQuoteBinding">
           <soap:address location="http://example.com/stockquote"/>
        </port>
    </service>
</definitions>
```

Figure 29. Example of a WSDL document

| The Web services wrapper uses the operations in a port type that have a SOAP
| binding with an HTTP transport. The input messages in the operation, and the

associated types or elements, become columns in the nickname. The output messages in the operation are extracted into the nickname hierarchy. You can create a separate hierarchy of nicknames for each operation in the WSDL document.

Figure 30 uses a WSDL document that contains a portType with an operation name of GETTEMP. With this Web service, you enter a zip code as input, and receive a temperature for that zip code.

```
<?xml version="1.0"?>
<definitions name="TemperatureService" targetNamespace=http://www.xmethods.net/
   sd/TemperatureService.wsdl"
 xmlns:tns="http://www.xmethods.net/sd/TemperatureService.wsdl"
 xmlns:xsd="http://www.w3.org/2001/XMLSchema"
 xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
 xmlns="http://schemas.xmlsoap.org/wsdl/">
 <message name="getTempRequest">
   <part name="zipcode" type="xsd:string"/>
 </message>
<message name="getTempResponse">
   <part name="return" type="xsd:float"/>
</message>
<portType name="TemperaturePortType">
  <operation name="getTemp">
      <input message="tns:getTempRequest"/>
      <output message="tns:getTempResponse"/>
  </operation>
</portType>
<binding name="TemperatureBinding" type="tns:TemperaturePortType">
  <soap:binding style="rpc"
      transport="http://schemas.xmlsoap.org/soap/http" />
  <operation name="getTemp">
     <soap:operation soapAction="" />
    <input>
     <soap:body use="encoded" namespace="urn:xmethods-Temperature"
        encodingStyle="http://schemas.xmlsoap.org/soap/encoding/" />
    </input>
    <output>
      <soap:body use="encoded" namespace="urn:xmethods-Temperature"
         encodingStyle="http://schemas.xmlsoap.org/soap/encoding/" />
    </output>
  </operation>
</binding>
<service name="TemperatureService">
  <documentation>
      Returns current temperature in a given U.S. zipcode
  </documentation>
  <port name="TemperaturePort" binding="tns:TemperatureBinding">
     <soap:address
       location="http://services.xmethods.net:80/soap/servlet/rpcrouter" />
  </port>
</service>
</definitions>
```

Figure 30. GETTEMP Web service

The input value is described by the zipcode column. The output value is described by the return column. In the WSDL document, those columns are identified in the messages element. The messages element represents the logical definition of the data that is sent between the Web service provider and the Web service consumer. If more explanation of the information in the message element is needed, then the

WSDL document can also contain a type element. The type element can refer to predefined types that are based on the XML schema specifications, or types that are defined by a user.

Figure 31 shows the nickname that the DB2® Control Center Discover tool produces from the WSDL document. The zipcode column is a required input column because of the nickname TEMPLATE syntax:

```
CREATE NICKNAME GETTEMP (
  ZIPCODE VARCHAR (48) OPTIONS(TEMPLATE '&column'),
   RETURN VARCHAR (48) OPTIONS(XPATH './return/text()')
   )
  FOR SERVER "EHPWSSERV"
   OPTIONS(URL 'http://services.xmethods.net:80/soap/servlet/rpcrouter',
           SOAPACTION ' ' ,
           TEMPLATE '<soapenv:Envelope>
                       <soapenv:Body>
                          <ns2:getTemp>
                            <zipcode>&zipcode[1,1]</zipcode>
                          </ns2:getTemp>
                       </soapenv:Body>
                     </soapenv:Envelope>',
           XPATH '/soapenv:Envelope/soapenv:Body/*' ,
           NAMESPACES ' ns1="http://www.xmethods.net/sd/TemperatureService.wsdl",
                        ns2="urn:xmethods-Temperature" ,
                          soapenv="http://schemas.xmlsoap.org/soap/envelope/"');
```

Figure 31. GETTEMP nickname

The Web services wrapper nickname options URL and SOAPACTION provide the ability to override the endpoint, or the address that you specified when you created the nickname. When you use the URLCOLUMN or SOAPACTIONCOLUMN enabled columns in a query, you can use dynamic addresses with the same nicknames. If you define the nickname options URL and SOAPACTION when you create a nickname, and enable the URLCOLUMN and SOAPACTIONCOLUMN on the column option, then you are using the late binding functions of Web services wrappers. The value for the SOAPACTION nickname option becomes an attribute in the HTTP header. The value for the URL nickname option is the HTTP URL to which the request is sent.

The URL and SOAPACTION nickname options provide dynamic nickname associations. These dynamic addresses are useful if several companies implement a Web service portType. The Web services wrapper requires that the only differences between the WSDL documents are different URLs and SOAPACTIONS. You can use the late binding function to create and use the same nickname for different service endpoints that different companies might want to use. The URL and SOAPACTION values are derived from the WSDL document.

The following example shows how you can use the URLCOLUMN and SOAPACTIONCOLUMN column options:

```
CREATE NICKNAME GetPartQuote(
  partnumber INTEGER OPTIONS (TEMPLATE'&column'),
  price FLOAT OPTIONS (XPATH './price')),
  urlcol VARCHAR(100) OPTIONS (URLCOLUMN 'Y'),
  soapactioncol VARCHAR(100) OPTIONS (SOAPACTIONCOLUMN 'Y'),
 FOR SERVER myServer
  OPTIONS (
  ...
  SOAPACTION 'http://example.com/GetPartPrice' ,
  URL 'http://mycompany.com:9080/GetPartPrice'',
  ...
   )
```

Figure 32. GetPartQuote nickname

The following example uses the columns URLCOL and SOAPACTIONCOL that were defined with the URLCOLUMN column option enabled, and the SOAPACTIONCOLUMN column option enabled:

```
SELECT * FROM supplier_endpoints p,
    GetPartQuote q
 WHERE partnumber=1234 AND
       p.url=q.urlcol AND
       p.soapaction=q.soapactioncol;
```

The SQL application can defer choosing which endpoints to use until the time that a query is run, instead of defining a specific endpoint at the time that the nickname is created.

The Web services wrapper can separate a large amount of WSDL document data into fragments to decrease the total memory that is used. Specify the **STREAMING** option in the DB2 Control Center in the Settings page of the Properties window, when you create a Web services nickname. The Web services wrapper processes the resulting stream of XML data and then extracts the information that is requested by a query fragment. The Web services wrapper parses one fragment at a time. Use the **STREAMING** option to parse large XML documents only.

Related concepts:
- "WSDL from a DADX file" in the *IBM DB2 Information Integrator Application Developer's Guide*
- "Web services description language" in the *IBM DB2 Information Integrator Application Developer's Guide*

Related tasks:
- "Adding Web services data sources to a federated server" on page 387
- "Registering the Web services wrapper" on page 388
- "Registering nicknames for Web services data sources" on page 390

Related reference:
- "CREATE NICKNAME statement – examples for the Web services wrapper" on page 398
- "Messages for the Web services wrapper" on page 411

Adding Web services to a federated system

Adding Web services data sources to a federated server

To configure the federated server to access Web services data sources, you must provide the federated server with information about the data sources and objects that you want to access, such as a valid Web services description language (WSDL) document.

You can configure the federated server to access Web services data sources by using the DB2 Control Center or the DB2 command line. The DB2 Control Center includes a wizard to guide you through the steps required to configure the federated server.

Prerequisites:

- DB2 Information Integrator must be installed on a server that will act as the federated server
- A federated database must exist on the federated server

Procedure:

To add Web services data sources to a federated server:

1. Register the wrapper.
2. Register the server definition.
3. Optional: Create a user mapping.
4. Register nicknames for the Web services data sources.
5. Optional: Create federated views for the Web services nicknames.

Related concepts:

- “DB2 Information Integrator installation process - overview” in the *IBM DB2 Information Integrator Installation Guide for Linux, UNIX, and Windows*
- “Web services wrapper security” on page 411
- “Web services wrapper security” on page 411
- “The Web services wrapper and the Web services description language document” on page 381

Related tasks:

- “Editing the Oracle genclntsh script and creating the libclntsh file after you install DB2 Information Integrator (HP-UX, Linux, Solaris)” in the *IBM DB2 Information Integrator Installation Guide for Linux, UNIX, and Windows*
- “Registering the Web services wrapper” on page 388
- “Registering the server definition for Web services data sources” on page 389
- “Registering nicknames for Web services data sources” on page 390
- “Creating federated views for Web services nicknames” on page 397
- “Creating a federated database” on page 51
- “Registering the Web services wrapper” on page 388

Related reference:

- “Supported operating systems for DB2 Information Integrator (32-bit)” in the *IBM DB2 Information Integrator Installation Guide for Linux, UNIX, and Windows*

- “CREATE NICKNAME statement – examples for the Web services wrapper” on page 398
- “Messages for the Web services wrapper” on page 411
- “Web services data sources – example queries” on page 408
- “Query restrictions for wrappers for business applications and Web services” on page 151

Registering the Web services wrapper

Registering the Web services wrapper is part of the larger task of adding Web services data sources to a federated server.

You must register a wrapper to access Web services data sources. Wrappers are used by federated servers to communicate with and retrieve data from data sources. Wrappers are implemented as a set of library files.

Procedure:

To register a wrapper, issue a CREATE WRAPPER statement with the name of the wrapper and the name of the wrapper library file.

For example, to register a wrapper with the name `websr_wrapper` on the federated server that uses the Windows operating system, issue the following statement:

```
CREATE WRAPPER websr_wrapper LIBRARY 'db2ws.dll';
```

The name of the wrapper library file that you specify depends on the operating system of the federated server. See the list of Web services wrapper library files for the correct name to specify in the CREATE WRAPPER statement.

The next task in this sequence of tasks is registering the server definitions for the Web services wrapper.

Related concepts:
- “Web services wrapper security” on page 411
- “The Web services wrapper and the Web services description language document” on page 381

Related tasks:
- “Adding Web services data sources to a federated server” on page 387
- “Registering the server definition for Web services data sources” on page 389

Related reference:
- “Web services wrapper library files” on page 388

Web services wrapper library files

The following table lists the directory paths and library file names for the Web services wrapper.

When you install DB2 Information Integrator, 3 library files are added to the directory path listed in the table. For example, if the federated server is running on AIX, the wrapper library files added to the directory path are `libdb2ws.a`, `libdb2wsF.a`, and `libdb2wsU.a`.

When you register a wrapper, specify only the library file name that is listed in the table.

Table 87. Web services wrapper library locations and file names

Operating system	Directory path	Wrapper library file
AIX	/usr/opt/db2_08_01/lib/	libdb2ws.a
Windows	%DB2PATH%\bin	db2ws.dll

%DB2PATH% is the environment variable that is used to specify the directory path where DB2 Information Integrator is installed on Windows. The default Windows directory path is `C:\Program Files\IBM\SQLLIB`.

Related tasks:
- "Registering the Web services wrapper" on page 388

Registering the server definition for Web services data sources

Registering the server definition for a Web services data source is part of the larger task of adding Web services to a federated system.

After you register the wrapper, you must register a corresponding server. A server definition must be registered for each Web service that you want to access.

You can register the server definition from a DB2 command line or from the DB2 Control Center.

Procedure:

To register a server definition to the federated system for the Web services wrapper, issue the CREATE SERVER statement.

For example, to register a Web services server definition named `ws_server` on Windows, issue the following statement:

```
CREATE SERVER ws_server WRAPPER websr_wrapper;
```

The next task in this sequence of tasks is registering the nicknames for the Web services data sources.

Related tasks:
- "Registering the Web services wrapper" on page 388
- "Registering nicknames for Web services data sources" on page 390

Related reference:
- "CREATE NICKNAME statement – examples for the Web services wrapper" on page 398
- "Query restrictions for wrappers for business applications and Web services" on page 151

Registering nicknames for Web services data sources

Registering nicknames for Web services data sources

Registering nicknames for Web services data sources is part of the larger task of adding Web services to a federated system.

You create one nickname hierarchy for each Web service operation. Web service operations are defined in the Web services description language (WSDL) document. Parent nicknames contain at least one child nickname. Child nicknames correspond to the elements that are nested within the element for the parent nickname.

You can create the nickname from a DB2 command line or from the DB2 Control Center.

In the DB2 Control Center, you can use the Discovery tool to quickly create the nicknames. The input to the Discover tool is a URL of the location of a WSDL document. The Discover tool creates nicknames as a result of processing the WSDL document. The WSDL document can contain schema definitions that are embedded in the WSDL file or in an external XML schema file that is imported in the WSDL file. These schema definitions are imported by using a URL address.

Prerequisites:

You must have access to a valid WSDL document that describes the Web service with which you want to communicate.

Restrictions:

- Only request-response operations are supported.
- A SOAP binding with an HTTP transport is the only binding that is supported.
- You must use either the TEMPLATE option or the XPATH option on each column, except for the special columns with the SOAPACTIONCOLUMN, URLCOLUMN, PRIMARY_KEY or FOREIGN_KEY options.

Procedure:

To register nicknames for Web services data sources from the DB2 command line, issue a CREATE NICKNAME statement.

For example, to register the nicknames on Windows for a Web service named `GETTEMP`, issue the following statement:

```
CREATE NICKNAME GETTEMP (
  ZIPCODE VARCHAR (48) OPTIONS(TEMPLATE '&column'),
   RETURN VARCHAR (48) OPTIONS(XPATH './return/text()')
   )
  FOR SERVER "EHPWSSERV"
   OPTIONS(URL 'http://services.xmethods.net:80/soap/servlet/rpcrouter',
           SOAPACTION ' ' ,
           TEMPLATE '<soapenv:Envelope>
                        <soapenv:Body>
                           <ns2:getTemp>
                             <zipcode>&zipcode[1,1]</zipcode>
                           </ns2:getTemp>
                        </soapenv:Body>
                      </soapenv:Envelope>',
           XPATH '/soapenv:Envelope/soapenv:Body/*' ,
          NAMESPACES ' ns1="http://www.xmethods.net/sd/TemperatureService.wsdl",
                       ns2="urn:xmethods-Temperature" ,
                         soapenv="http://schemas.xmlsoap.org/soap/envelope/"');
```

The data definition language (DDL) that the DB2 Control Center generates maps all input elements to columns of the root nickname in the nickname hierarchy. The nickname that is created is derived from the WSDL document.

To register nicknames for Web services data sources from the DB2 Control Center:

1. Expand the **Federated Database Objects** folder.
2. Expand the wrapper folder for which you want to register nicknames.
3. Expand the **Server Definitions** folder.
4. Expand the server folder for which you want to register nicknames.
5. Right click the **Nicknames** folder and select **Create**.
6. In the Create Nicknames window, click **Discover** to define search criteria to help you select objects at the data source.
7. Specify the WSDL document that contains the definition of the Web service that you want DB2 Information Integrator users to access.

   The WSDL document can be a local document or you can specify its location by using a URL.
8. Click **OK** to create the nickname according to the selected WSDL document.

   The DB2 Control Center extracts the WSDL document into multiple create nickname DDL statements, with the appropriate parent-child relationship definitions. The nicknames that are created represent the Web services hierarchy that is defined in the WSDL document.

Optional: The next task in this sequence of tasks is creating federated views for the Web services nicknames.

Related concepts:

- "Web services wrapper security" on page 411

Related tasks:

- "Specifying nickname columns for a nonrelational data source" on page 65
- "Adding Web services data sources to a federated server" on page 387
- "Registering the Web services wrapper" on page 388

Related reference:

- "CREATE NICKNAME statement – examples for the Web services wrapper" on page 398
- "Messages for the Web services wrapper" on page 411
- "Web services data sources – example queries" on page 408
- "Query restrictions for wrappers for business applications and Web services" on page 151

The TEMPLATE option at the nickname and column levels

This topic applies to the WebSphere® Business Integration wrapper and the Web services wrapper.

The WebSphere® Business Integration wrapper and the Web services wrapper build XML documents that are required by the WebSphere Business Integration Adapter and the Web services environment. The wrappers need the nickname level and the column level template fragments, which is the TEMPLATE option on the CREATE NICKNAME statement, at the time that the nickname is created. The wrappers use this information during the query planning and the query execution phases.

Web services wrapper

For the Web services wrapper, the required and optional attributes vary according to the definitions in the WSDL document and how a column is derived. A column can be derived from either an element or an attribute of an element.

- If the column is derived from an element, then the minOccurs value determines if a column is optional.
- If the value of minOccurs equals 0, then the column is optional.
- If the value of minOccurs equals 1, then the column is required.
- If the column is derived from an attribute of an element, then the value of use on the attribute determines if a column is optional.
- If an attribute contains the value use=optional, then the column is optional.
- If an attribute contains the value use=required, then the column is required.

The following example is an attribute in a schema definition that is associated with a column:

```
<xsd:complexType>
  <xsd:sequence>
    <xsd:element ref="tns:ZooName"/>
    <xsd:element ref="tns:Count"/>
    <xsd:element ref="tns:LastModified"/>
    <xsd:element maxOccurs="unbounded" minOccurs="0" ref="tns:Zookeeper"/>
  </xsd:sequence>
  <xsd:attribute name="id" type="xsd:string" use="optional"/>
</xsd:complexType>
```

WebSphere Business Integration wrapper

For the WebSphere Business Integration wrapper, the required and optional columns vary according to the application and the associated adapter. You need to identify the required and optional input columns by specifying the appropriate template option values for those columns. Before you use the DB2® Control Center to create the nicknames, you must modify the XML schema definition file to flag the required and optional input columns.

SAP BAPI

The IBM® DB2 Control Center determines the required and optional input columns based on the value of specific flags in the XML schema definition (XSD) files that represent the business object definition

In the annotation section of an element at any level of the business object hierarchy (parent or child business objects), an `I` prefix in the appSpecificInfo value indicates an import parameter for the SAP BAPI to which the business object definition maps. An `E` prefix indicates an export parameter for the SAP BAPI. Some elements can be both import and export parameters for a BAPI. The following example shows an element which is both an import and an export parameter:

```
<bx:appSpecificInfo>ICOMPANYCODE:ECOMPANYCODE</bx:appSpecificInfo>
```

The prefixes are generated automatically by the WebSphere Business Integration Object Discovery Agent tool based on information that is extracted from the SAP business object repository.

If an element that represents an import parameter (an I prefix in the appSpecificInfo value) is specified with the attribute `minOccurs=1`, the DB2 Control Center identifies the element as a required input parameter and flags the elements as a required input column in the nickname definition. The WebSphere Business Integration Object Discovery Agent tool does not automatically set the value of `minOccurs` to 1 for the required input parameters of the SAP BAPI. You must reference the SAP Business Object Repository to determine all the required input parameters for the BAPI that you want to access. Then, you must edit the corresponding elements in the XML schema file by manually setting the attribute to `minOccurs=1`. If the `minOccurs` attribute value for an input parameter remains as the default value of 0, then the DB2 Control Center specifies the column as an optional input column in the nickname hierarchy that is generated.

The following example shows an optional input column:

```
<xsd:element name="Company_code" minOccurs="0">
  <xsd:annotation>
   <xsd:appinfo>
     <bx:boAttribute>
       <bx:appSpecificInfo>ICOMPANYCODE:</bx:appSpecificInfo>
       <bx:attributeInfo isForeignKey="false" isKey="true" />
     </bx:boAttribute>
   </xsd:appinfo>
  </xsd:annotation>
  <xsd:simpleType>
   <xsd:restriction base="xsd:string">
     <xsd:maxLength value="4" />
   </xsd:restriction>
  </xsd:simpleType>
</xsd:element>
```

The following example shows a required input column:

```
<xsd:element name="Company_id" minOccurs="1">
  <xsd:annotation>
   <xsd:appinfo>
     <bx:boAttribute>
       <bx:appSpecificInfo>ICOMPANYID:</bx:appSpecificInfo>
       <bx:attributeInfo isForeignKey="true" isKey="true" />
     </bx:boAttribute>
   </xsd:appinfo>
  </xsd:annotation>
  <xsd:simpleType>
   <xsd:restriction base="xsd:string">
     <xsd:maxLength value="4" />
   </xsd:restriction>
  </xsd:simpleType>
</xsd:element>
```

The required and optional input columns for SAP business applications are designated by the syntax shown in the following table:

Table 88. Flagging schema for SAP input column information

Flags used in SAP XSD files	Required input column	Column reference in nickname template
Any element anywhere in the hierarchy with the prefix = 'I' and minOccurs=1	Yes	&columnname[1,1]

Table 88. Flagging schema for SAP input column information (continued)

Flags used in SAP XSD files	Required input column	Column reference in nickname template
Any element anywhere in the hierarchy with the prefix = 'I' and minOccurs=0	No	&columnname[0,1]

Siebel and PeopleSoft

The DB2 Control Center determines the required and optional input columns based on the existence and the value of the `isRequired` flag in the attributeInfo section of the annotation for the element. If there is no isRequired flag, then the column is not an input column. The WebSphere Business Integration Object Discovery Agent tool does not automatically generate these flags in the XSD file. You must identify the required and optional input columns, and flag them appropriately in the XSD file before you use the DB2 Control Center to generate the nickname DDL.

The following example shows the flags for a required input column and optional input columns in the XSD file for a Siebel or PeopleSoft business object definition.

```
<xsd:element name="sieb_ssa_Contact_Contact">
 <xsd:annotation>
  <xsd:appinfo>
   <bx:boDefinition version="1.0.0">
   <bx:appSpecificInfo>ON=Contact;CN=Contact</bx:appSpecificInfo>
   </bx:boDefinition>
  </xsd:appinfo>
 </xsd:annotation>
<xsd:complexType>
<xsd:sequence>
<xsd:element name="Id" minOccurs="0">
 <xsd:annotation>
  <xsd:appinfo>
   <bx:boAttribute>
    <bx:appSpecificInfo>FN=Id</bx:appSpecificInfo>
    <bx:attributeInfo isForeignKey="false"
       isKey="true" isRequired="true" />
   </bx:boAttribute>
  </xsd:appinfo>
 </xsd:annotation>
...
</xsd:element>

...
```

Figure 33. Portion of a Siebel business object definition (Part 1 of 2)

```
...
<xsd:element name="FirstName" minOccurs="1">
<xsd:annotation>
 <xsd:appinfo>
  <bx:boAttribute>
  <bx:appSpecificInfo>FN=First Name</bx:appSpecificInfo>
  <bx:attributeInfo isForeignKey="false" isKey="false"
        isRequired="false" />
  </bx:boAttribute>
 </xsd:appinfo>
</xsd:annotation>
<xsd:simpleType>
 <xsd:restriction base="xsd:string">
  <xsd:maxLength value="50" />
 </xsd:restriction>
</xsd:simpleType>
</xsd:element>
<xsd:element name="LastName" minOccurs="1">
<xsd:annotation>
 <xsd:appinfo>
  <bx:boAttribute>
   <bx:appSpecificInfo>FN=Last Name</bx:appSpecificInfo>
   <bx:attributeInfo isForeignKey="false" isKey="false"
         isRequired="false" />
  </bx:boAttribute>
 </xsd:appinfo>
</xsd:annotation>
...
```

Figure 33. Portion of a Siebel business object definition (Part 2 of 2)

The required and optional input columns for Siebel and PeopleSoft business applications are designated by the syntax shown in the following table:

Table 89. Flagging schema for Siebel and PeopleSoft input column information

Flags used in Siebel and PeopleSoft XSD files	Required input column	Column reference in nickname template
isRequired=″true″	Yes	&columnname[1,1]
isRequired=″false″	No	&columnname[0,1]

The following example shows the DDL that the DB2 Control Center creates based on the XSD file that is shown in the figure labeled *Portion of a Siebel business object definition*. The XSD file in that figure included a value of false for the isRequired attribute.

```
CREATE NICKNAME sieb_ssa_Contact_Contact_NN(
 Id VARCHAR(15)  OPTIONS(XPATH './ns1:Id/text()',
           TEMPLATE '<ns1:Id>&column</ns1:Id>'),
 FirstName VARCHAR(50)  OPTIONS(XPATH './ns1:FirstName/text()',
           TEMPLATE '<ns1:FirstName>&column</ns1:FirstName>'),
 LastName VARCHAR(50)  OPTIONS(XPATH './ns1:LastName/text()',
           TEMPLATE '<ns1:LastName>&column</ns1:LastName>'),
 AccountId VARCHAR(255)  OPTIONS(XPATH './ns1:AccountId/text()'),
 PrimaryAccountName VARCHAR(100)
           OPTIONS(XPATH './ns1:PrimaryAccountName/text()'),
 PrimaryPostalCode VARCHAR(30)
           OPTIONS(XPATH './ns1:PrimaryPostalCode/text()'),
 PrimaryStreetAddress VARCHAR(200)
           OPTIONS(XPATH './ns1:PrimaryStreetAddress/text()'),
SalesRep VARCHAR(255)  OPTIONS(XPATH './ns1:SalesRep/text()'),
State VARCHAR(255)  OPTIONS(XPATH './ns1:State/text()'))
 FOR SERVER siebel_server
 OPTIONS(XPATH '//ns1:sieb_ssa_Contact_Contact',
 TEMPLATE '<ns1:sieb_ssa_Contact_Contact>
               &Id[1,1] &FirstName[0,1] &LastName[0,1]
          </ns1:sieb_ssa_Contact_Contact>',
 BUSOBJ_NAME 'sieb_ssa_Contact_Contact',
 NAMESPACES  'ns1="http://www.ibm.com/websphere/
               crossworlds/2002/BOSchema/sieb_ssa_Contact_Contact"');
```

Related concepts:
- "The WebSphere Business Integration wrapper" on page 119
- "The Web services wrapper and the Web services description language document" on page 381

Related tasks:
- "Adding business application data sources to a federated system" on page 125
- "Adding Web services data sources to a federated server" on page 387
- "Registering nicknames for Web services data sources" on page 390

Related reference:
- "Business application data sources – example queries" on page 155
- "CREATE NICKNAME statement – examples for the WebSphere Business Integration wrapper" on page 138
- "CREATE NICKNAME statement – examples for the Web services wrapper" on page 398
- "Messages for the Web services wrapper" on page 411
- "Web services data sources – example queries" on page 408
- "Query restrictions for wrappers for business applications and Web services" on page 151

Nicknames and XPATH expressions

This topic applies to the WebSphere® Business Integration wrapper and the Web services wrapper.

Nicknames correspond to the tree structure of your XML document data. Parent nicknames and child nicknames correspond to the root structure and nested elements of the data tree structure. These parent and child nicknames are connected by primary and foreign keys that are specified with the CREATE NICKNAME statement.

Each nickname is defined by XPath expressions that represent output values. The WebSphere Business Integration wrapper and the Web services wrapper use XPath expressions to establish a correspondence between the data in an XML document and the rows in a relational table. These XPath expressions identify the values in the XML document and determine how these values correspond to the columns of each row. The WebSphere Business Integration wrapper and the Web services wrapper read the XML document data only. The wrappers do not update the data. The XPATH option contains the information to find the SOAP messages through the SOAP envelope and SOAP body tags. The getQuote message is contained in the SOAP envelope and body elements.

The NICKNAME option XPATH expression points to repeating tags that are in the output element. The XPath expression determines how many or which rows will be in the nickname. The column option XPATH expression is relative to the NICKNAME XPATH expression. The column option XPATH identifies the values in a row. A NICKNAME option XPATH in a child nickname is relative to a NICKNAME option XPATH expression in a parent nickname.

When you create a nickname, you choose options that specify the association between the nickname and the XML document. Nicknames created for WebSphere Business Integration wrappers are associated with an XML schema definition (XSD) document. Nicknames that are created for Web services wrappers are associated with a Web services description language (WSDL) document.

Related concepts:
- "What is XML?" on page 415
- "The Web services wrapper and the Web services description language document" on page 381
- "Data associations between nicknames and XML documents" on page 422

Related tasks:
- "Adding XML to a federated system" on page 418
- "Registering nicknames for XML data sources" on page 424
- "Creating federated views for nonroot nicknames (XML wrapper)" on page 430
- "Adding business application data sources to a federated system" on page 125
- "Registering nicknames for business application data sources" on page 129
- "Adding Web services data sources to a federated server" on page 387
- "Registering nicknames for Web services data sources" on page 390

Related reference:
- "Business application data sources – example queries" on page 155
- "CREATE NICKNAME statement – examples for the WebSphere Business Integration wrapper" on page 138
- "CREATE NICKNAME statement – examples for the Web services wrapper" on page 398
- "Web services data sources – example queries" on page 408
- "Query restrictions for wrappers for business applications and Web services" on page 151
- "CREATE NICKNAME statement - Examples for XML wrapper" on page 425

Creating federated views for Web services nicknames

Creating federated views for Web services nicknames is part of the larger task of adding Web services data sources to a federated system.

You can define federated views for the hierarchy of nicknames that describe a Web services document. Defining federated views ensures that the queries that join pieces of a Web services nickname hierarchy can run properly.

Procedure:

To define federated views for Web services nicknames:

1. Define a view for one or more Web services nicknames. If you want to join all of the nicknames that are related to an operation in the Web service, you must define a view that includes all of those nicknames.
2. In the WHERE clause of the view, use join predicates for all columns that are related by the PRIMARY_KEY and FOREIGN_KEY column options.

In the following example, the primary key is on column `ooport_getzooreport_pk` in nickname `zooport_getzooreport_report_nn`. The foreign key is on column `ooport_getzooreport_fkey` in nickname `zooport_getzooreport_report_report_nn`.

```
CREATE VIEW zooreport
  (zooid, zooname, number_of_zookeeper,
   lastmodified,zookeeper_id, zookeeper_name,
   fingers_left, animal_name, animal_species, animal_lot)
AS ( SELECT zooid, report_zooname,
      report_count, report_lastmodified,
      zookeeper_id, zk.report_name, report_numberfingersleft,
      a.report_name, report_species,
      report_lot
     FROM zooport_getzooreport_nn ,
          zooport_getzooreport_report_nn as zk,
          zooport_getzooreport_report_report_nn as a
     WHERE zk.ooport_getzooreport_pkey=a.ooport_getzooreport_fkey
      AND zooport_getzooreport_pkey=zk.ooport_getzooreport_fkey);
```

You can get information from all of the nicknames with the following SELECT statement:

```
SELECT * FROM zooreport WHERE zooid='1';
```

There are no further tasks in this sequence of tasks.

Related reference:
- "CREATE NICKNAME statement – examples for the Web services wrapper" on page 398
- "Web services data sources – example queries" on page 408
- "Query restrictions for wrappers for business applications and Web services" on page 151

CREATE NICKNAME statement – examples for the Web services wrapper

When you create a nickname to access a Web service, you create an input column for each value in the input message of a Web service operation and an output column for each value in the output message of a Web service operation. You control the input and output column definitions with the nickname column option definitions.

The TEMPLATE column option specifies that a column is an input column. The XPATH column option specifies that a column is an output column. When the TEMPLATE nickname option contains a bracketed notation ([1,1]), the column is a required input column. When the TEMPLATE nickname option contains a bracketed notation ([0,1]), the column is an optional input column.

The NAMESPACES nickname option is a comma-separated list of name-value pairs that a federated system uses to resolve the namespaces that are used for elements in input and output XML documents. The namespaces are used in the message request so that the prefixes that are used in the TEMPLATE nickname option are defined. The NAMESPACES nickname option is used to resolve the prefixes that are used in XPath expressions with the namespace URIs that are defined in the WSDL or XML schemas. The XPath expressions are applied on the XML document that is returned from the Web service.

Example 1: Required input columns

The following example shows a nickname that uses a Web service named getQuote. The Web service reads a stock ticker symbol, and returns a trading price. The following DDL is created by the Discover tool in the DB2 Control Center.

```
CREATE NICKNAME "stockquote.stockquoteport_getquote_nn" (
 symbol VARCHAR (48) OPTIONS(TEMPLATE '&column'),
 result VARCHAR (48) OPTIONS(XPATH './Result/text()'))
FOR SERVER "xmethods_server"  OPTIONS(
 URL 'http://66.28.98.121:9090/soap' ,
 SOAPACTION 'urn:xmethods-delayed-quotes#getQuote' ,
 TEMPLATE '<soapenv:Envelope>
                  <soapenv:Body>
                   <ns2:getQuote>
                     <symbol>&symbol[1,1]</symbol>
                   </ns2:getQuote>
                  </soapenv:Body>
                 </soapenv:Envelope>',
   XPATH '/soapenv:Envelope/soapenv:Body/*' ,
   NAMESPACES 'ns2="urn:xmethods-delayed-quotes" ,
              ns1="http://www.themindelectric.com/wsdl/
                  net.xmethods.services.stockquote.StockQuote/" ,
                  soapenv="http://schemas.xmlsoap.org/soap/envelope/" ');
```

The nickname TEMPLATE option specifies column SYMBOL as a required input column, because it contains the [1,1] designation. In the nickname TEMPLATE option, the complete SOAP envelope is specified for the Web service. The getQuote input value is contained in the SOAP envelope and body elements. The XPATH nickname option contains the information to find the trading price value through the SOAP envelope and body tags.

Use the "stockquote.stockquoteport_getquote_nn" nickname to access the Web service, such as in the following query:

```
SELECT * FROM "stockquote.stockquoteport_getquote_nn"
  WHERE symbol='IBM';
```

You must use the predicate, `symbol='IBM'` in this statement because symbol is a required input column. The equality predicate is the only valid predicate on input columns. Each of the equality predicates sets a value in the input message. If the input column is optional, an equality predicate on that column is not necessary. If the input column is required, then you must issue the query with an equality predicate. You can use a literal value such as IBM in an equality expression or a value from a joined table or nickname.

Example 2: Repeating elements and child nicknames

The following example uses a Web service named `getZooReport` that produces a report for zoos. The input value is a zoo identifier. The output value is a report that is described by the following schema:

```
<wsdl:definitions name="Name"
    targetNamespace="http://myzoo.com"
...
<wsdl:types>
 <xsd:schema elementFormDefault="qualified" targetNamespace="http://myzoo.com"
    xmlns:xsd="http://www.w3.org/2001/XMLSchema">
   <xsd:element name="Animal">
      <xsd:complexType>
         <xsd:sequence>
          <xsd:element ref="tns:Name"/>
          <xsd:element ref="tns:Species"/>
          <xsd:element ref="tns:Lot"/>
         </xsd:sequence>
      </xsd:complexType>
  </xsd:element>
  <xsd:element name="AnimalCareList">
     <xsd:complexType>
       <xsd:sequence>
         <xsd:element maxOccurs="unbounded" minOccurs="1" ref="tns:Animal"/>
       </xsd:sequence>
     </xsd:complexType>
  </xsd:element>
  <xsd:element name="Count" type="xsd:string"/>
  <xsd:element name="LastModified" type="xsd:string"/>
  <xsd:element name="Lot" type="xsd:string"/>
  <xsd:element name="Name" type="xsd:string"/>
  <xsd:element name="NumberFingersLeft" type="xsd:string"/>
  <xsd:element name="Species" type="xsd:string"/>
  <xsd:element name="Zoo">
     <xsd:complexType>
       <xsd:sequence>
         <xsd:element ref="tns:ZooName"/>
         <xsd:element ref="tns:Count"/>
         <xsd:element ref="tns:LastModified"/>
         <xsd:element maxOccurs="unbounded" minOccurs="0" ref="tns:Zookeeper"/>
       </xsd:sequence>
       <xsd:attribute name="id" type="xsd:string" use="optional"/>
      </xsd:complexType>
 </xsd:element>
 <xsd:element name="ZooName" type="xsd:string"/>
 <xsd:element name="Zookeeper">
     <xsd:complexType>
       <xsd:sequence>
         <xsd:element ref="tns:Name"/>
         <xsd:element ref="tns:NumberFingersLeft"/>
         <xsd:element ref="tns:AnimalCareList"/>
       </xsd:sequence>
      <xsd:attribute name="id" type="xsd:string" use="optional"/>
    </xsd:complexType>
</xsd:element>
</xsd:schema>
</wsdl:types>
...
```

Figure 34. getZooReport Web service

Here is the DDL that is generated by the DB2 Control Center Discover tool based on the WSDL that contains the schema:

```
CREATE NICKNAME zooport_getzooreport_nn (
 zooid VARCHAR (48) OPTIONS(TEMPLATE '&column'),
 zoo_id VARCHAR (48) OPTIONS(XPATH './ns1:Zoo/@ns1:id'),
 report_zooname VARCHAR (48) OPTIONS(XPATH './ns1:Zoo/ns1:ZooName/text()'),
 report_count VARCHAR (48) OPTIONS(XPATH './ns1:Zoo/ns1:Count/text()'),
 report_lastmodified VARCHAR (48)
       OPTIONS(XPATH './ns1:Zoo/ns1:LastModified/text()'),
 zooport_getzooreport_pkey VARCHAR (16) FOR BIT DATA NOT NULL
       OPTIONS(PRIMARY_KEY 'YES'))
FOR SERVER "zooserver"  OPTIONS(
 URL 'http://localhost:9080/MaelstromTest/services/ZooPort' ,
 SOAPACTION 'http://myzoo.com/getZooReport' ,
 TEMPLATE '<soapenv:Envelope>
              <soapenv:Body>
                <zooId>&zooId[1,1]</zooId>
              </soapenv:Body>
             </soapenv:Envelope>' ,
 XPATH '/soapenv:Envelope/soapenv:Body' ,
     NAMESPACES ' soapenv="http://schemas.xmlsoap.org/soap/envelope/" ,
                 ns1="http://myzoo.com " ');
```

Figure 35. Zoo report – parent nickname – zooport_getzooreport_nn

```
CREATE NICKNAME zooport_getzooreport_report_nn (
 zooport_getzooreport_fkey VARCHAR (16)
        FOR BIT DATA NOT NULL
           OPTIONS(FOREIGN_KEY 'ZOOPORT_GETZOOREPORT_NN'),
   zookeeper_id VARCHAR (48) OPTIONS(XPATH './ns1:Zookeeper/@ns1:id'),
   report_name VARCHAR (48) OPTIONS(XPATH './ns1:Zookeeper/ns1:Name/text()'),
   report_numberfingersleft VARCHAR (48)
           OPTIONS(XPATH './ns1:Zookeeper/ns1:NumberFingersLeft/text()'),
   zooport_getzooreport_pkey VARCHAR (16)
           FOR BIT DATA NOT NULL OPTIONS(PRIMARY_KEY 'YES'))
  FOR SERVER "zooserver"  OPTIONS(
        XPATH './ns1:Zoo' ,
        NAMESPACES ' soapenv="http://schemas.xmlsoap.org/soap/envelope/" ,
               ns1="http://myzoo.com" ');
```

Figure 36. Zoo report – child of nickname zooport_getzooreport_nn

```
CREATE NICKNAME zooport_getzooreport_report_report_nn (
        zooport_getzooreport_fkey VARCHAR (16) FOR BIT DATA NOT NULL
             OPTIONS(FOREIGN_KEY 'zooport_getzooreport_report_nn'),
        report_name VARCHAR (48)
             OPTIONS(XPATH './ns1:Animal/ns1:Name/text()'),
        report_species VARCHAR (48)
             OPTIONS(XPATH './ns1:Animal/ns1:Species/text()'),
        report_lot VARCHAR (48) OPTIONS(XPATH './ns1:Animal/ns1:Lot/text()'))
 FOR SERVER "zooserver"  OPTIONS(
        XPATH './ns1:Zookeeper/ns1:AnimalCareList' ,
        NAMESPACES ' soapenv="http://schemas.xmlsoap.org/soap/envelope/" ,
                  ns1="http://myzoo.com" ');
```

Figure 37. Zoo report – child of zooport_getzooreport_report_nn

The schema includes some elements that are repeated, or sequence elements. These repeated elements become child nicknames of the parent nickname, as shown in Figure 35, Figure 36, and Figure 37. For example, zooname, count, lastmodified, and zookeeper are all elements of zoo. The element zoo contains 0 or more zookeeper elements. The root nickname, zoo, contains the columns zooname, count, and lastmodified. A child nickname, zookeeper, is created by the DB2 Control Center Discover tool to describe the repeating elements of zookeeper. The third element in the zookeeper column, animalcarelist, also contains 0 or more elements and so it becomes a child nickname, zooport_getzooreport_report_report_nn. The following figure shows the nickname hierarchy:

Root nickname: zooport_getzooreport_nn

Zoo (parent):

- ZooName
- Count
- LastModified
- ZooKeeper (there are 0 or more ZooKeeper elements)

  Child nickname: zooport_getzooreport_report_nn

  ZooKeeper elements

  - Name
  - NumberFingersLeft
  - AnimalCareList (there are 0 or more Animal elements)

    Child nickname: zooport_getzooreport_report_report_nn

    Animal

    - Name
    - Species
    - Lot

Figure 38. Parent —> Child —> nickname hierarchies

The following statement is a typical query that you might issue on the nicknames to access the zoo report Web service. When you issue this statement, you retrieve the information from the zoo report based on a specific identifier and on where the primary and foreign keys of the child nickname zoo reports match.

```
SELECT * FROM zooport_getzooreport_nn ,
       zooport_getzooreport_report_nn zk ,
       zooport_getzooreport_report__report__nn a
   WHERE zooid='1'AND zooport_getzooreport_pkey=zk.zooport_getzooreport_fkey
   and zk.zooport_getzooreport_pkey=a.zooport_getzooreport_fkey;
```

Example 3: Late binding

The following example shows how you can use the Late Binding option. You can use this option from the DB2 Control Center or from a DB2 Command line. If you define the nickname options URL and SOAPACTION, and if you enable the column options URLCOLUMN and SOAPACTIONCOLUMN when you create a nickname, you are using the late binding functions. The DB2 Control Center creates two column options, URLCOLUMN and SOAPACTIONCOLUMN, and sets the values of the columns to `yes`.

The following example is for a Web service that provides price quotes for parts that is implemented by all suppliers for a company. Here is the CREATE statement that includes the URLCOLUMN and SOAPACTIONCOLUMN definitions:

```
CREATE NICKNAME GetPartQuote(
  partnumber INTEGER OPTIONS (TEMPLATE'&column'),
  price FLOAT OPTIONS (XPATH './price')),
  urlcol VARCHAR(100) OPTIONS (URLCOLUMN 'Y'),
  soapactioncol VARCHAR(100) OPTIONS (SOAPACTIONCOLUMN 'Y'),
 FOR SERVER myServer
  OPTIONS (
  ...
  SOAPACTION 'http://example.com/GetPartPrice' ,
  URL 'http://mycompany.com:9080/GetPartPrice'',
  ...
   )
```

To get price quotes from all of the suppliers with a single query, the values that each supplier uses for the SOAPACTION and URL column options are needed. The query looks like this:

```
SELECT * FROM supplier_endpoints p,
    GetPartQuote q
 WHERE partnumber=1234 AND
       p.url=q.urlcol AND
       p.soapaction=q.soapactioncol;
```

Local table supplier_endpoints contains all of the URLs and SOAP addresses with which you can call the Web service. You can include an ORDER BY price clause to determine the least expensive supplier for this part.

Example 4: ESCAPE_INPUT column option

You can include XML fragments as input values in your query. When you register a nickname, include the column option ESCAPE_INPUT=N. This option maintains the special characters, such as <, and >, in XML fragments in the input values.

When a schema contains repeating input values that would require you to send XML as part of the SOAP message, you can use the ESCAPE_INPUT column option to build the output message with the correct XML.

For example, the zoo Web service includes an operation to add a new zoo keeper and the animals that are associated with that zoo keeper. In the schema for this example, an AnimalCareList can have multiple animals.

```
CREATE NICKNAME add_zookeeper(
 zookeeper_id VARCHAR(48)  OPTIONS(TEMPLATE '...'),
 name VARCHAR(48) OPTIONS(TEMPLATE '...'),
 numberfingersleft VARCHAR(48) OPTIONS(TEMPLATE '...'),
 animals VARCHAR(3000) OPTIONS( TEMPLATE '...' , ESCAPE_INPUT 'N')
...
```

To add a new zoo keeper with two animals, issue a query such as the following example:

```
SELECT * FROM add_zookeeper
  WHERE zookeeper_ID='37' AND
        name='Amit Tsunami' AND
        numberfingersleft='3'  AND
        animals='<AnimalCareList xmlns="http://myzoo.com">
                  <Animal>
                    <Name>Larry</Name>
                    <Species>Gorilla</Species>
                    <Lot>7</Lot>
                  </Animal>
                  <Animal>
                    <Name>Bill</Name>
                    <Species>Chimpanzee</Species>
                    <Lot>8H</Lot>
                  </Animal>
                </AnimalCareList>';
```

The add_zookeeper nickname is a Web service operation that can change the state of the Web service, or update information. Although nonrelational wrappers cannot be updated, the SELECT statement in this example updates the zoo information to add a new zoo keeper.

You can also use the ESCAPE_INPUT column option for a schema that uses an element such as xsd:anyType. In this case, the type of the element is unknown. You can use the ESCAPE_INPUT column option on the input column for that element so that you can specify arbitrary XML fragments for your input.

Related concepts:
- "The Web services wrapper and the Web services description language document" on page 381

Related tasks:
- "Adding Web services data sources to a federated server" on page 387
- "Registering nicknames for Web services data sources" on page 390

Related reference:
- "Messages for the Web services wrapper" on page 411
- "Web services data sources – example queries" on page 408
- "Query restrictions for wrappers for business applications and Web services" on page 151

Query restrictions for wrappers for business applications and Web services

Equal predicates

The only valid predicates on input columns are equal predicates. For output columns, any predicate is valid.

The following example returns an error with a message that indicates that the predicate is not supported on that column. In this example, the column `zipcode` is an input column:

```
SELECT return FROM gettemp WHERE zipcode<'95141'
```

The following example shows a valid query using an equal predicate on the input columns. The customers nickname is joined with a local DB2 UDB table that contains customer IDs. The query contains an additional predicate on the Sales column, which is an output only column.

```
SELECT a.name, a.address
  FROM customers a, local_table b
  WHERE
    a.customer_id=b.custid AND
    a.Sales > 300000;
```

Predicates for required input columns

You must provide equality predicate values for all required input columns in your SQL queries for the nickname hierarchy that you reference. The wrapper returns an SQLCODE 901 for all queries that violate this restriction.

IN or OR predicates

For WebSphere Business Integration wrappers and Web services wrappers, no IN or OR predicates are allowed for input columns.

The following examples show invalid queries. The customers nickname has one required input column, customer_id:

```
SELECT * FROM customers
  WHERE customer_id IN (12345, 67890, 11223);
SELECT * FROM customers
  WHERE customer_id IN (SELECT custid FROM local_table);  )
```

However, for the WebSphere Business Integration wrappers, you can use IN list predicates with required input columns if you define a unique index with the SPECIFICATION ONLY parameter for the required input columns:

```
CREATE UNIQUE INDEX myuindex ON customers(customer_id) SPECIFICATION ONLY;
```

Joins on optional input columns

The following example demonstrates a restriction on joining optional input columns. You cannot join optional input columns from a local table or nickname. If the WSDL generates an input nickname column as optional and you need to use that column in a join, then you must edit the DDL to change the column to a required input column.

In this example, a Web service wrapper nickname named `order` is created with `shipping_method` as an optional input column. The following statement is a valid query because it uses a literal in the predicate:

```
SELECT * FROM order
  WHERE part="hammer" AND shipping_method="FEDEX";
```

However, if you include a local table named `orderparts`, which defines parts and shipping methods, in the query, and the table contains a column named `shipping_method` that is optional, the statement is invalid:

```
SELECT * FROM
   order o, orderparts op
  WHERE
     o.part="hammer" AND
     o.shipping_method=op.shipping_method
```

For a WebSphere Business Integration wrapper, predicates on optional input columns of a nickname might be pushed down to the WebSphere Business Integration Adapter. DB2 UDB can decide to apply those predicates locally on the rows fetched from the application data source. To ensure that predicates for a given input column always get pushed down to the adapter, declare the input column as a required input column. Every query on the nickname hierarchy must include predicate values for the required input columns.

To ensure valid results, joined input columns must be required columns for Web Services wrappers.

Outer joins

Outer joins between nicknames using the primary key from a parent nickname and the foreign key from child nickname columns are not supported.

When a parent element in an XML document contains no child elements, and if you use an inner join between the parent nickname and the child nickname, then no rows are returned for that element. For example, for a given customer, if there is no bankdetail information in the SAP system, then no rows are returned for the sap_bapi_customer_getdetail2_sap_customerbankdetail_NN nickname for the particular customer.

The following CREATE NICKNAME statements define the columns that are used in the example query:

```
CREATE NICKNAME sap_bapi_customer_getdetail2_NN(
 ...
 NAME VARCHAR(35)
     OPTIONS(XPATH './ns3:sap_customeraddress/
           ns1:sap_customeraddress/ns1:NAME/text()'),
 ...
 NN__PKEY VARCHAR(16)  OPTIONS(PRIMARY_KEY 'YES'),
 COMPANYCODE VARCHAR(4)  OPTIONS(XPATH './ns3:COMPANYCODE/text()',
       TEMPLATE '<ns3:COMPANYCODE>&column</ns3:COMPANYCODE>'),
 CUSTOMERNO VARCHAR(10)  OPTIONS(XPATH './ns3:CUSTOMERNO/text()',
       TEMPLATE '<ns3:CUSTOMERNO>&column</ns3:CUSTOMERNO>'),
 ...
  FOR SERVER sap_server
  OPTIONS(XPATH '//ns3:sap_bapi_customer_getdetail2',
  TEMPLATE '<ns3:sap_bapi_customer_getdetail2>
                  &sap_bapi_customer_getdetail2_sap_customerbankdetail_NN[0,1]
                  &COMPANYCODE[0,1]
                  &CUSTOMERNO[1,1]
             </ns3:sap_bapi_customer_getdetail2>',
  ...
```

Figure 39. Excerpt from getdetail2 nickname

```
CREATE NICKNAME sap_bapi_customer_getdetail2_sap_customerbankdetail_NN(
 CUSTOMER VARCHAR(10)  OPTIONS(XPATH './ns2:CUSTOMER/text()',
          TEMPLATE '<ns2:CUSTOMER>&column</ns2:CUSTOMER>'),
 BANK_KEY VARCHAR(15)  OPTIONS(XPATH './ns2:BANK_KEY/text()',
          TEMPLATE '<ns2:BANK_KEY>&column</ns2:BANK_KEY>'),
 BANK_ACCT VARCHAR(18)  OPTIONS(XPATH './ns2:BANK_ACCT/text()',
          TEMPLATE '<ns2:BANK_ACCT>&column</ns2:BANK_ACCT>'),
 CTRL_KEY VARCHAR(2)  OPTIONS(XPATH './ns2:CTRL_KEY/text()',
          TEMPLATE '<ns2:CTRL_KEY>&column</ns2:CTRL_KEY>'),
 BANK_REF VARCHAR(20)  OPTIONS(XPATH './ns2:BANK_REF/text()',
          TEMPLATE '<ns2:BANK_REF>&column</ns2:BANK_REF>'),
 NN__FKEY VARCHAR(16)  OPTIONS(FOREIGN_KEY 'SAP_BAPI_CUSTOMER_GETDETAIL2_NN'))
  FOR SERVER sap_server
  OPTIONS(XPATH './ns3:sap_customerbankdetail/ns2:sap_customerbankdetail',
  TEMPLATE '<ns3:sap_customerbankdetail>
                <ns2:sap_customerbankdetail>
                  &CUSTOMER[0,1]
                  &BANK_KEY[0,1]
                  &BANK_ACCT[0,1]
                  &CTRL_KEY[0,1]
                  &BANK_REF[0,1]
                </ns2:sap_customerbankdetail>
              </ns3:sap_customerbankdetail>',
 ...
```

Figure 40. Excerpt from customer bank detail nickname

In the following example, the query returns no rows because there is an inner join condition between the two nicknames:

```
SELECT a.name, b.bank_key
  FROM sap_bapi_customer_getdetail2_NN a,
     sap_bapi_customer_getdetail2_sap_customerbankdetail_NN b
  WHERE a.customerno='1234567890'
  AND a.NN__PKEY=b.NN__FKEY;
```

If a WebSphere Business Integration wrapper or a Web services wrapper nickname definition contains required input columns, then a left outer join between this nickname and any other local DB2 UDB table or other nicknames is not supported.

Related concepts:
- "The TEMPLATE option at the nickname and column levels" on page 131

Related tasks:
- "Adding business application data sources to a federated system" on page 125
- "Adding Web services data sources to a federated server" on page 387

Related reference:
- "Business application data sources – example queries" on page 155
- "CREATE NICKNAME statement – examples for the WebSphere Business Integration wrapper" on page 138
- "CREATE NICKNAME statement – examples for the Web services wrapper" on page 398
- "Web services data sources – example queries" on page 408

Web services data sources – example queries

Example 1: Using materialized query tables

You use materialized query tables to locally cache the results of a query and to improve the performance of queries. You can use nicknames from Web services data sources to create materialized query tables. For some queries, the database can automatically determine whether the materialized query table can answer a query without accessing the base tables. The following procedure shows how to create and populate a materialized query table:

1. Create a local or base table:

   ```
   CREATE TABLE mystocks(ticker VARCHAR(10));
   ```

   You can use the local table to maintain all the values that you want to cache.
2. Insert all of the values that you want to cache into the table:

   ```
   INSERT INTO mystocks VALUES('IBM');
   INSERT INTO mystocks VALUES('MSFT');
   ...
   ```
3. Create a Web services nickname:

   ```
   CREATE NICKNAME stockquote_nn (
           ticker VARCHAR(40) OPTIONS (TEMPLATE   '&column'),
           price VARCHAR(16) OPTIONS (XPATH   './Result/text()')
           )
    FOR SERVER stock_server
    OPTIONS (TEMPLATE '<ticker>&column</ticker>'
               XPATH './Result/text()' );
   ```
4. Create a view that consists of the nickname and the local table:

   ```
   CREATE VIEW  stock_quote_view (ticker, price)
     AS (
       SELECT nn.ticker, nn.price
        FROM stockquote_nn nn, mystocks s
        WHERE nn.ticker=s.ticker
   );
   ```
5. Create a materialized query table:

   ```
   CREATE TABLE stockquote_MQT (ticker, ticker2, price)
           as (SELECT nn.ticker,s.ticker as ticker2, nn.price
     FROM stockquote_nn nn, mystocks s
     WHERE nn.ticker=s.ticker )
     DATA INITIALLY DEFERRED REFRESH DEFERRED;
   ```

   Include all of the VARCHAR columns that are used in the join predicate (nn.ticker and s.ticker) in the materialized query table output list to maximize the opportunities that the materialized query table is used by DB2 Universal Database.

   To defer the refresh of the materialized query table, specify the REFRESH DEFERRED keyword. Materialized query tables that are specified with the REFRESH DEFERRED keyword do not reflect changes to the underlying base table. Use the clause DATA INITIALLY DEFERRED so that your data is not inserted into the table as part of the CREATE TABLE statement.
6. Issue a REFRESH TABLE statement to populate the table. The data in the table reflects the result of the query as a snapshot at the time that you issue the REFRESH TABLE statement. The following example populates the stockquote_MQT table, and sets a value for the current refresh age special register.

```
   REFRESH TABLE stockquote_MQT;

   SET CURRENT REFRESH AGE any;
```

The queries that run on the data in the materialized query table are faster than the queries that run on the data in a base table. When you want to use the materialized query table, you refer to the view and not the nickname:

```
SELECT * FROM stock_quote_view
  WHERE  ticker='IBM';
```

If you issue a query to select a value that has not been cached, 0 rows are returned.

Example 2: Issuing joins using the primary and foreign keys

The PRIMARY_KEY and FOREIGN_KEY columns are used to define relationships between the parent and child nicknames. Each parent nickname must have a primary key column option. You define the children of a parent nickname with the foreign key column option that references the primary key column of a parent nickname. A nickname can have multiple children, but a nickname can have only one parent.

Because these columns contain only binary data, the columns are defined with the FOR BIT DATA NOT NULL keywords. The DB2 Control Center generates this definition for you when you create the nickname. You can explicitly define the PRIMARY_KEY and FOREIGN_KEY columns as FOR BIT DATA NOT NULL when you create the nickname.

The following example shows how the Web services wrapper uses the PRIMARY_KEY and FOREIGN_KEY columns to associate parent and child nicknames.

```
CREATE NICKNAME zooport_getzooreport_nn (
   zooid VARCHAR (48) OPTIONS(TEMPLATE '&column'),
   zoo_id VARCHAR (48) OPTIONS(XPATH './ns1:Zoo/@id'),
   report_zoo_zooname VARCHAR (48)
       OPTIONS(XPATH './ns1:Zoo/ns1:ZooName/text()'),
   report_zoo_count VARCHAR (48)
       OPTIONS(XPATH './ns1:Zoo/ns1:Count/text()'),
   report_zoo_lastmodified VARCHAR (48)
       OPTIONS(XPATH './ns1:Zoo/ns1:LastModified/text()'),
   nn_pk VARCHAR (16) NOT NULL OPTIONS(PRIMARY_KEY 'YES'),
   url VARCHAR (256) OPTIONS(URLCOLUMN 'Y'),
   soapaction VARCHAR (256) OPTIONS(SOAPACTIONCOLUMN 'Y')
) FOR SERVER "mytestsrvr"
  OPTIONS(
    URL 'http://localhost:9080/MaelstromTest/services/ZooPort',
    SOAPACTION 'http://myzoo.com/getZooReport' ,
    TEMPLATE '<soapenv:Envelope>
               <soapenv:Body>
                 <zooId>&zooId[1,1]</zooId>
               </soapenv:Body>
              </soapenv:Envelope>',
    XPATH '/soapenv:Envelope/soapenv:Body' ,
    NAMESPACES ' soapenv="http://schemas.xmlsoap.org/soap/envelope/",
                ns1="http://myzoo.com" ');

CREATE NICKNAME zooport_getzooreport_report_zookeeper_nn (
  nn_fk VARCHAR (16) NOT NULL
      OPTIONS(FOREIGN_KEY 'ZOOPORT_GETZOOREPORT_NN'),
  zookeeper_id VARCHAR (48) OPTIONS(XPATH './@id'),
  report_zookeeper_name VARCHAR (48) OPTIONS(XPATH './ns1:Name/text()'),
  zookeeper_numberfingersleft VARCHAR(48)
```

```
        OPTIONS(XPATH './ns1:NumberFingersLeft/text()'),
  nn_pk VARCHAR (16) NOT NULL OPTIONS(PRIMARY_KEY 'YES')
 )
FOR SERVER "MYTESTSRVR" OPTIONS(
  XPATH './ns1:Zoo/ns1:Zookeeper' ,
  NAMESPACES ' soapenv="http://schemas.xmlsoap.org/soap/envelope/",
       ns1="http://myzoo.com" ');
```

The foreign key, nn_fk, in nickname zooport_getzooreport_report_zookeeper_nn, refers to the parent nickname, zooport_getzooreport_nn in the foreign key option. The designated primary and foreign key nickname columns do not correspond to data in your WSDL document because these nickname columns contain keys that are generated by the wrapper. These keys identify a relationship between the parent and child nicknames that is unique only within a query. If the child nickname contains an input column, the parent nickname option template refers to that child nickname in the nickname option template structure.

The following SQL statement joins the parent and child nicknames:

```
SELECT *
FROM   zooport_getzooreport_nn a,
       zooport_getzooreport_report_zookeeper_nn z,

WHERE   a.nn_pk  = z.nn_fk
    AND a.zooid  = 100
    ;
```

The following description explains how the Web services wrapper uses the TEMPLATE and XPATH nickname and column options during query execution. It is not intended as an example of specific implementation.

When you join the primary and foreign key columns, the Web services wrapper sends a message to the Web services provider, and a set of rows is returned from the Web services provider. The wrapper generates a message for the parent nickname by substituting the values of the input column (a.zooid = 100) from the query for the reference in the column option template (ZOOID VARCHAR (48) OPTIONS(TEMPLATE ’&column’)), and then all of the column references in the nickname template option (<zooId>&zooId[1,1]</zooId>). The nickname template option can include column references or child nickname references. The message is then sent to the Web service.

The wrapper generates the rows for a nickname by applying the nickname option XPATH on the document that the Web service returns. If the nickname option XPATH returns multiple XML fragments, then the nickname contains multiple rows. The column XPATH option is applied on the resulting XML fragments that represent the rows to get the column values. If a nickname has one or more indirect parents, all of the parent nickname XPATH expressions are applied in the order from the top of the hierarchy down before the nickname option XPATH and the column option XPATH are applied for this nickname.

Related tasks:

- “Adding Web services data sources to a federated server” on page 387
- “Registering the Web services wrapper” on page 388
- “Registering nicknames for Web services data sources” on page 390

Related reference:

- “CREATE NICKNAME statement – examples for the Web services wrapper” on page 398

- “Messages for the Web services wrapper” on page 411
- “Query restrictions for wrappers for business applications and Web services” on page 151

Web services wrapper security

The Web services wrapper supports HTTPS as a transport protocol for SOAP messages. HTTPS is a standard encryption protocol that is used by many Web service providers. The WSDL document that is generated by the Web services provider contains https:// in the URL. The SOAP messages in the HTTP request or HTTP response are encrypted.

If the Web service uses HTTPS as a transport protocol, the Web service wrapper does not validate the SSL certificates that the server sends for identification. The Web service wrapper can call Web services with self-signed certificates.

The Web services wrapper supports HTTP authentication by using the CREATE USER MAPPING statement. Use the CREATE USER MAPPING statement to map a federated server user ID to a Web services user ID and password. A wrapper developer supplies a user ID and password with a user mapping statement such as the following example:

```
CREATE USER MAPPING
  FOR RSPALTEN SERVER S1
       OPTIONS ( REMOTE_AUTHID 'SYSTEM', REMOTE_PASSWORD 'MANAGER' )
```

When a Web services nickname on the S1 server is accessed, the HTTP request is sent with SYSTEM as the user ID and MANAGER as the password. The user mapping is optional. If you do not specify a user mapping, you might see an error if the Web service provider expects authentication information. Some servers might use authentication to restrict access to a service. The need for authentication is not apparent from the information in the WSDL document.

Related concepts:

- “The Web services wrapper and the Web services description language document” on page 381

Related tasks:

- “Adding Web services data sources to a federated server” on page 387
- “Registering the Web services wrapper” on page 388
- “Registering nicknames for Web services data sources” on page 390

Related reference:

- “Messages for the Web services wrapper” on page 411

Messages for the Web services wrapper

The following table explains some of the typical error messages that you might receive when you use the Web services wrapper.

Table 90. Typical error messages

Error	Description	User response
SQL1822N Unexpected error code ″SOAP-Fault″ received from data source ″wswrap.svl.ibm.″. Associated text and tokens are ″java.lang.Exception: HTTP URI s″. SQLSTATE=560BD	The Web service provider returned a SOAP fault.	• The input might be incorrect. Check that your input arguments are correct. • The Web services provider might have a problem. Contact the owner of the Web service.
SQL30081N A communication error has been detected. Communication protocol being used: ″SOAP″. Communication API being used: ″HTTP″. Location where the error was detected: ″localhos″. Communication function detecting the error: ″connect″. Protocol specific error code(s): ″**38309**″, ″10061″, ″1″. SQLSTATE=08001	The Web service provider did not listen on the port or URL.	• Check to see if the URLs that you use are all valid. • Check to see if the port information is correct. • Ensure that the server is running.
SQL30081N A communication error has been detected. Communication protocol being used: ″SOAP″. Communication API being used: ″HTTP″. Location where the error was detected: ″doesntexist.ibm″. Communication function detecting the error: ″*″. Protocol specific error code(s): ″38308″, ″*″, ″0″.SQLSTATE=08001	The Web service provider host name is not in the domain name server.	Ensure that your host name is in the name server.
SQL30081N A communication error has been detected. Communication protocol being used: ″SOAP″. Communication API being used: ″HTTP″. Location where the error was detected: ″www.ibm.com″. Communication function detecting the error: ″*″. Protocol specific error code(s): ″38312″, ″*″, ″0″.SQLSTATE=08001	HTTP return code, similar to a 404 error that a browser typically reports.	Determine if the server is returning any errors to your application. Run a DB2 UDB trace to determine what response the server is returning.
SQL30081N A communication error has been detected. Communication protocol being used: ″SOAP″. Communication API being used: ″HTTP″. Location where the error was detected: ″″. Communication function detecting the error: ″*″. Protocol specific error code(s): ″38304″, ″*″, ″0″. SQLSTATE=08001	The URL is incorrect. The SQLSTATE 38304 might indicate the protocol is not known. The SQLSTATE 38305 indicates a syntax error in the URL.	Verify that your WSDL document contains a valid URL syntax and protocol.

Table 90. Typical error messages (continued)

Error	Description	User response
SQL1822N Unexpected error code ″SAXException″ received from data source ″wswrap.svl.ibm.″. Associated text and tokens are ″Expected end of tag ′ns0:Mi″. SQLSTATE=560BD	The response includes a parse error on the XML output.	Verify that the server returned correct XML output. Run a DB2 UDB trace to determine what the Web services provider returns. You can also invoke the Web service with a different tool to ensure that the Web service response is valid.

Related concepts:

- "The Web services wrapper and the Web services description language document" on page 381

Related tasks:

- "Adding Web services data sources to a federated server" on page 387
- "Registering the Web services wrapper" on page 388

Related reference:

- "CREATE NICKNAME statement – examples for the Web services wrapper" on page 398
- "Web services data sources – example queries" on page 408
- "Query restrictions for wrappers for business applications and Web services" on page 151

Chapter 24. Configuring access to XML data sources

This chapter explains how to configure your federated server to access data that is stored in XML data sources. You can configure access to XML data sources by using the DB2 Control Center or by issuing SQL statements.

This chapter:
- Explains what XML is
- Lists the tasks that you need to perform
- Contains examples of the SQL statements that you need
- Lists the error messages associated with the XML wrapper

What is XML?

The Extensible Markup Language (XML) is a universal format for structured documents and data. XML files have a file extension of `xml`. Like HTML, XML uses tags (words bracketed by > and <) for structuring data in the document. A sample XML document is shown in Figure 41.

```
<?xml version="1.0" encoding=UTF-8"?>
<doc>
   <customer id='123'>
      <name>...</name>
      <address>...</address>
       ...
      <order>
         <amount>...</amount>
            <date>...</date>
         <item quant='12'>
            <name>...</name>
         </item>
         <item quant='4'>...</item>
          ...
      </order>
      <order>...</order>
       ...
      <payment>
         <number>...</number>
         <date>...</date>
      </payment>
      <payment>>...</payment>
       ...
   </customer>
   <customer id='124'>...</customer>
</doc>
```

Figure 41. Sample XML document

How the XML wrapper works

The XML wrapper enables the use of SQL to query the following types of data:
- External XML documents that are stored in a single file
- Multiple XML files in a directory path
- Remote XML files that are referenced with a Uniform Reference Identifier (URI)

|

- XML documents stored in relational columns

Figure 42 shows how the XML wrapper works with your federated system.

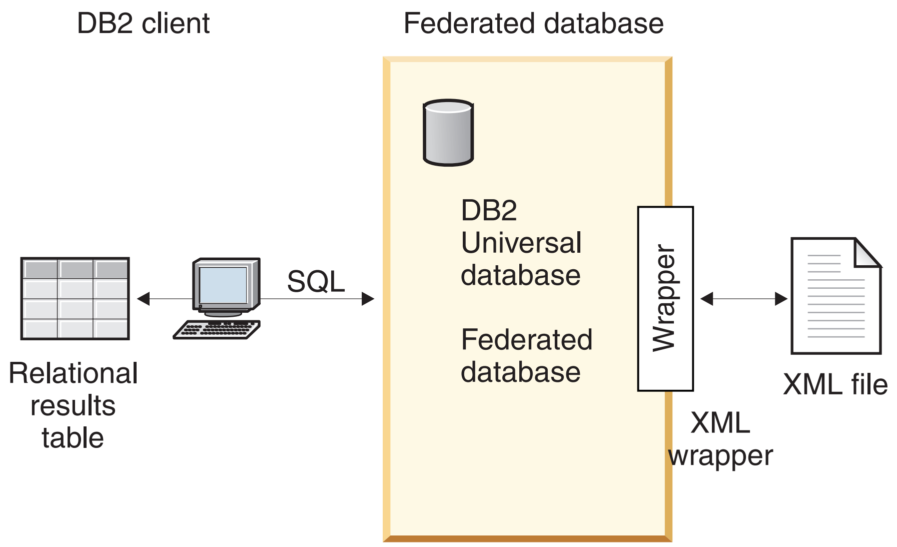


Figure 42. How the XML wrapper works

With the XML wrapper, you can map XML data from an external data source into a relational schema that is composed of a set of nicknames. The structure of an XML document is logically equivalent to a relational schema in which the nested and repeating elements are modeled as separate tables with foreign keys.

The nicknames that correspond to an XML document are organized into a tree structure in which the child nicknames map to elements that are nested within the element that corresponds to the parent nickname.

When nested elements are repeated or have distinct identities with complex structures, you can provide separate nicknames for each nested element.

Child and parent nicknames are connected by primary and foreign keys that are generated by the wrapper.

XPath expressions are used to map an XML document into a relational schema that is composed of a set of nicknames. XPath is an addressing mechanism for identifying the parts of an XML file (for example, the groups of nodes and attributes within an XML document tree). The basic XPath syntax is similar to file system addressing.

Each nickname is defined by an XPath expression that identifies the XML elements representing individual tuples, and a set of XPath expressions that specifies how to extract the column values from each element.

An example of XML document mapping:

The following example illustrates how the sample XML document, shown in Figure 41 on page 415, is mapped into a set of nicknames, how parent and child relationships are established by using primary and foreign keys, how XPath expressions are used to define individual tuples and columns within each element of the document, and how a query can run on the XML document after the document is registered to your federated system.

The sample XML document contains a set of customer elements. Each element encloses several order and payment elements.

The order elements enclose several item elements.

The relationship among the elements is shown in Figure 43.

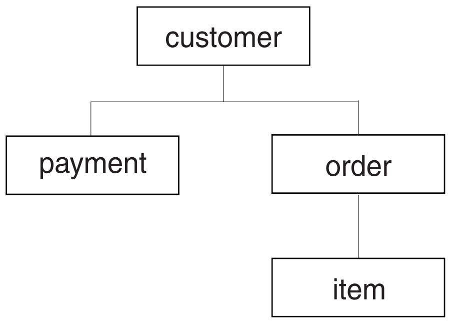


Figure 43. Tree structure of the sample XML document

From this structure, you can use the CREATE NICKNAME statement to map the XML document into a relational schema that includes four nicknames:
- customers
- orders
- payments
- items

You define relationships between the nicknames by specifying each nickname as a parent nickname or a child nickname by using special primary and foreign key nickname column options. Each parent nickname must have a special column that is designated with a primary key column option. You define the children of a parent nickname with the foreign key column option that references the primary key column of a parent nickname. The designated primary and foreign nickname columns do not correspond to data in your XML document because these nickname columns will contain keys that are generated by the wrapper. A nickname can have multiple children, but a nickname can have only one parent. The root nickname has no parent.

For the sample XML document, the customers nickname has a defined primary key, and the orders, payments, and items nicknames have defined foreign keys that point to the parent nickname. The foreign keys of the orders and payments nicknames point to the customers nickname, and the foreign key of the items nickname points to the orders nickname.

To identify the XML elements representing individual tuples, you create one XPath expression. In this example, all the customer elements are referenced by using the `'/doc/customer'` XPath expression, and all the order elements are referenced by using the `'./order'` XPath expression. The period in the `'./order'` XPath expression indicates that the tuples of each order element are nested within the tuples of the corresponding customer element.

You create a set of XPath expressions to specify how to extract the column values from each element. In this example, the `id` attribute of the customer elements, now a column defined in the nickname, is referenced by using the `'./@id'` XPath expression. The name element of the customer elements is referenced by using the `'./name'` XPath expression, and the address element of the customer elements is referenced by using the `'./address/@street'` XPath expression.

After you map the XML document into a set of nicknames by using the CREATE NICKNAME statement, you define each nickname as a parent or child by using primary and foreign keys, with XPath expressions that define individual tuples and columns within each element of the document. You can then run SQL queries on the XML document.

Related concepts:

- "Data associations between nicknames and XML documents" on page 422

Related tasks:

- "Adding XML to a federated system" on page 418

Adding XML to a federated system

Adding XML to a federated system

To configure the federated server to access XML data sources, you must provide the federated server with information about the data sources and objects that you want to access.

| You can configure the federated server to access XML data sources by using the
| DB2 Control Center or the DB2 command line. The DB2 Control Center includes a
| wizard to guide you through the steps required to configure the federated server.

Prerequisites:

- DB2 Information Integrator must be installed on a server that will act as the federated server
- A federated database must exist on the federated server

Procedure:

To add an XML data source to a federated server:

1. Register the XML wrapper.
2. Register the XML server definition.
3. Register nicknames for the XML data sources.
4. Create federated views for non-root nicknames.

   | A root nickname is a nickname at the top level of a nickname hierarchy. A
   | nonroot nickname is a nickname that has a parent nickname in a nickname
   | hierarchy. You can have root nicknames that are not the top level element in an
   | XML document.

You can run the statements from the DB2 Control Center or from a DB2 command line processor. After you add the XML wrapper to your federated system, you can run queries on an XML data source.

Related concepts:

- "DB2 Information Integrator installation process - overview" in the *IBM DB2 Information Integrator Installation Guide for Linux, UNIX, and Windows*

Related tasks:
- "Creating a federated database" on page 51
- "Registering the XML wrapper" on page 419
- "Registering the server for an XML data source" on page 420
- "Registering nicknames for XML data sources" on page 424
- "Creating federated views for nonroot nicknames (XML wrapper)" on page 430

Related reference:
- "Supported operating systems for DB2 Information Integrator (32-bit)" in the *IBM DB2 Information Integrator Installation Guide for Linux, UNIX, and Windows*
- "Supported operating systems for DB2 Information Integrator (64-bit)" in the *IBM DB2 Information Integrator Installation Guide for Linux, UNIX, and Windows*

Registering the XML wrapper

Registering the XML wrapper is part of the larger task of adding XML data sources to a federated server.

You must register a wrapper to access HMMER data sources. Wrappers are used by federated servers to communicate with and retrieve data from data sources. Wrappers are implemented as a set of library files.

Procedure:

To register a wrapper, issue the CREATE WRAPPER statement with the name of the wrapper and the name of the wrapper library file.

For example, to register a wrapper with the name `xml_wrapper` on the federated server that uses the AIX operating system, issue the following statement:

```
CREATE WRAPPER xml_wrapper LIBRARY 'libdb2lsxml.a';
```

The name of the wrapper library file that you specify depends on the operating system of the federated server. See the list of XML wrapper library files for the correct name to specify in the CREATE WRAPPER statement.

The next task in this sequence of tasks is registering the server definitions for the XML wrapper.

Related reference:
- "XML wrapper library files" on page 419
- "CREATE WRAPPER statement" in the *SQL Reference, Volume 2*

| ## XML wrapper library files

| The following table lists the directory paths and library file names for the XML
| wrapper.

When you install DB2 Information Integrator, 3 library files are added to the directory path listed in the table. For example, if the federated server is running on AIX, the wrapper library files added to the directory path are `libdb2lsxml.a`, `libdb2lsxmlF.a`, and `libdb2lsxmlU.a`.

When you register a wrapper, specify only the library file name that is listed in the table.

Table 91. XML wrapper library locations and file names

Operating system	Directory path	Wrapper library file
AIX	/usr/opt/db2_08_01/lib/	libdb2lsxml.a
HP-UX	/opt/IBM/db2/V8.1/lib	libdb2lsxml.sl
Linux	/opt/IBM/db2/V8.1/lib	libdb2lsxml.so
Solaris	/opt/IBM/db2/V8.1/lib	libdb2lsxml.so
Windows	%DB2PATH%\bin	db2lsxml.dll

%DB2PATH% is the environment variable that is used to specify the directory path where DB2 Information Integrator is installed on Windows. The default Windows directory path is `C:\Program Files\IBM\SQLLIB`.

Related tasks:

- "Registering the XML wrapper" on page 419

Registering the server for an XML data source

Registering the server for an XML data source is part of the larger task of adding XML to a federated system. After you register the wrapper, you must register a corresponding server.

Restrictions:

The XML wrapper does not use the TYPE and VERSION keywords. An error occurs if these keywords are used in the CREATE SERVER statement.

The XML wrapper does not support pass-through sessions to the federated system.

Procedure:

To register the XML server to the federated system, issue the CREATE SERVER statement.

For example:

```
CREATE SERVER xml_server WRAPPER my_xml;
```

The next task in this sequence of tasks is registering nicknames for XML data sources.

Accessing XML documents through a proxy server

To access XML documents through a proxy server, you must specify options when you create the server definition. The options that you specify depend on the type of proxy server that you want to access.

You must use the proxy options in the CREATE SERVER statement if all of the following conditions are true:

- You want to retrieve data using a URI
- The URI used will retrieve data from behind a firewall, through a proxy
- The firewall or proxy used is HTTP, SOCKS4, or SOCKS5

Check with your Network administrator for information about the type of proxy that you use, and the settings that you should specify in the proxy options.

Example of registering a server definition for an HTTP proxy server:

To register a server definition and specify an HTTP proxy server, use the following statement:

```
CREATE SERVER xml_server_h
    WRAPPER xml_wrapper
    OPTIONS (PROXY_TYPE 'HTTP', PROXY_SERVER_NAME 'proxy_h',
        PROXY_SERVER_PORT '8080');
```

Example of registering a server definition for a SOCKS4 proxy server:

To register a server definition and specify a SOCKS4 proxy server, use the following statement:

```
CREATE SERVER xml_server_s4
    WRAPPER xml_wrapper
    OPTIONS (PROXY_TYPE 'SOCKS4', PROXY_SERVER_NAME 'proxy_4',
        PROXY_SERVER_PORT '1080');
```

Example of registering a server definition for a SOCKS5 proxy server without authentication information:

To register a server definition and specify a SOCKS5 proxy server without authentication information, use the following statement:

```
CREATE SERVER xml_server_s5
    WRAPPER xml_wrapper
    OPTIONS (PROXY_TYPE 'SOCKS5', PROXY_SERVER_NAME 'proxy_5',
        PROXY_SERVER_PORT '1081');
```

Example of registering a server definition for a SOCKS5 proxy server with authentication information:

To register a server definition and specify a SOCKS5 proxy server with authentication information, use the following statement:

```
CREATE SERVER xml_server_s5a
    WRAPPER xml_wrapper
    OPTIONS (PROXY_TYPE 'SOCKS5', PROXY_SERVER_NAME 'proxy_5',
        PROXY_SERVER_PORT '1081', PROXY_AUTHID 'Martin',
        PROXY_PASSWORD 'aaa', );
```

The XML validation feature might have some limitations when it is used with the proxy feature. The conditions in which you will see this limitation are:

- You are using the proxy feature, at the server level, you have set the various proxy options.
- The XML instance document contains a reference to an external XML schema located outside the firewall

If you have one of these conditions, try to change the location of your XML schema to a location inside the firewall. If you change the XML schema location, you must update the XML instance document with the new location of the XML schema.

Related tasks:
- "Registering nicknames for XML data sources" on page 424

Related reference:
- "CREATE SERVER statement" in the *SQL Reference, Volume 2*

Registering nicknames for XML data sources

Data associations between nicknames and XML documents

Nicknames correspond to the tree structure of your XML document data. Parent nicknames and child nicknames correspond to the root structure and nested elements of the data tree structure. These parent and child nicknames are connected by primary and foreign keys that are specified with the CREATE NICKNAME statement.

Each nickname is defined by XPath expressions that perform the following functions:
- Identifies the XML elements that represent individual tuples
- Specifies how to extract the column values from each element

The XML wrapper uses XPath expressions to establish a correspondence between the data in the XML document and the rows in a relational table. These XPath expressions identify the values within the XML document and determine how these values correspond to the columns of each row. The XML wrapper reads the XML document data only. The XML wrapper does not update this data.

When you create a nickname, you choose options that specify the association between the nickname and the XML document. Nicknames are associated with your XML documents either in a fixed manner or with source names that you specify.

With a fixed association, the nickname represents data from specific XML documents. These XML documents include:

One local file
You specify one XML file as your XML document.

Multiple local files in a directory path
You specify a directory path in which multiple XML files reside. The XML files in this directory path provide the XML document data to the nickname. All of the XML files must have the same configuration. If any XML file in the directory has a configuration that is different from the configuration of the nickname, the XML wrapper returns null values when it processes that XML data file. The directory must be either local to the federated server or accessible from a shared file system.

Note: When scanning the directory, the XML wrapper retains and parses only those files with a `.xml` extension. The XML wrapper ignores all other files, including files with a `.txt` extension, files with a `.xsd` extension, and files without extensions.

Use the FILE_PATH option of the CREATE NICKNAME statement to specify fixed data from a file. Use the DIRECTORY_PATH option to specify fixed data from a directory.

When the source data is specified while the query is running, you can use the nickname to represent data from any XML document source whose schema matches the nickname definition. These XML documents include:

Uniform Reference Identifiers
A remote XML file that a URI refers to supplies the XML document data to the nickname. (Specify this document source by using the DOCUMENT 'URI' nickname column option.)

Relational columns
Columns from a relational table, view, or nickname are used as input to your XML document. (Specify this document source by using the DOCUMENT 'COLUMN' nickname column option.)

File
A single file that contains XML data is supplied as input while the query runs. (Specify this document source by using the DOCUMENT 'FILE' nickname column option.)

Directory
Multiple XML files under a specified directory path supply the data while the query runs. (Specify this document source by using the DOCUMENT 'DIRECTORY' nickname column option.)

You specify the DOCUMENT column option to indicate that the source data is supplied at query time. Specify either URI, COLUMN, FILE, or DIRECTORY with the DOCUMENT column to indicate the type of XML document source.

You cannot specify a FILE_PATH option or a DIRECTORY_PATH option with a DOCUMENT column option.

Regardless of the type of data that you are using (data in a fixed format or data from source names that are specified at query time), you can specify the STREAMING option so that the XML wrapper separates the XML document data into fragments. The XML wrapper processes the resulting stream of XML data and extracts the information that is requested by a query fragment. The XML wrapper parses one fragment at a time. Because fragments are parsed one at a time, total memory use decreases but the processing time required to run the entire query increases depending on the memory capacity of your server. Therefore, use the STREAMING option to parse large XML documents (documents of 50 megabytes or more) only.

You can also choose nickname option values that help you optimize queries that retrieve large amounts of XML data or data that contains multiple nested elements. These options include:

- INSTANCE_PARSE_TIME
- XPATH_EVAL_TIME
- NEXT_TIME

You can set values for these options to test and optimize the XML query. These option values control the processing time that is needed to locate elements and to parse the data in the rows of the XML document.

Related concepts:

- “What is XML?” on page 415
- “The cost model facility for the XML wrapper” on page 424
- “Optimization tips for the XML cost model facility” on page 430

Related tasks:
- “Registering nicknames for XML data sources” on page 424

Related reference:
- “CREATE NICKNAME statement syntax - XML wrapper” on page 557
- “CREATE NICKNAME statement - Examples for XML wrapper” on page 425

The cost model facility for the XML wrapper

The XML wrapper provides a cost model facility to optimize queries on nicknames that correspond to your XML source documents.

When you create a nickname by using the CREATE NICKNAME statement, you can specify the following parameters as nickname option values to support the cost model facility:
- INSTANCE_PARSE_TIME
- XPATH_EVAL_TIME

You can use the default values for these parameters. Or you can set the values for these parameters to optimize queries on the root and nonroot nicknames that you create.

The INSTANCE_PARSE_TIME parameter is the amount of time (in milliseconds) that is required to read and parse one row-producing root element of the root nickname (for example, customers), including all contained row-producing nonroot elements (for example, all elements that correspond to the orders, payments, and items of each customer). The XML wrapper builds a structure in memory to represent these row-producing root and nonroot elements.

The XPATH_EVAL_TIME parameter is the amount of time (in milliseconds) that is required to evaluate the XPath expressions that locate the data corresponding to a row of the nickname. The XPath expressions that are evaluated include the XPath expressions that locate the actual rows and the XPath expressions that locate column values within these rows.

Related concepts:
- “What is XML?” on page 415
- “Data associations between nicknames and XML documents” on page 422
- “Optimization tips for the XML cost model facility” on page 430

Related reference:
- “CREATE NICKNAME statement syntax - XML wrapper” on page 557
- “CREATE NICKNAME statement - Examples for XML wrapper” on page 425

Registering nicknames for XML data sources

Registering nicknames for XML data sources is part of the larger task of adding XML to a federated system. You must create nicknames that correspond to the tree structure of your XML data source. Parent nicknames correspond to the root structure of the tree. Child nicknames correspond to the elements that are nested within the element for the parent nickname.

Prerequisite:

The database code page must match the character set of the XML source files.

Restriction:

Namespaces are not supported.

Procedure:

To register nicknames for XML data sources, issue a CREATE NICKNAME statement.

The next task in this sequence of tasks is creating federated views for nonroot nicknames (XML wrapper).

Related concepts:
- "Data associations between nicknames and XML documents" on page 422
- "The cost model facility for the XML wrapper" on page 424

Related tasks:
- "Adding XML to a federated system" on page 418
- "Creating federated views for nonroot nicknames (XML wrapper)" on page 430
- "Specifying nickname columns for a nonrelational data source" on page 65

Related reference:
- "CREATE NICKNAME statement syntax - XML wrapper" on page 557
- "CREATE NICKNAME statement - Examples for XML wrapper" on page 425

CREATE NICKNAME statement - Examples for XML wrapper

This topic provides several examples that show you how to use the CREATE NICKNAME statement to register nicknames for the XML wrapper. This topic includes a complete example, which shows how to create parent and child nicknames, examples for specific column options, and examples that show the use of views.

| **Recommendation**: Do not use the self or descendant operator // when you specify
| XPATH columns and nickname options in your queries. The self or descendant
| operator is an XPath operator, and using it can decrease federated server
| performance.

Complete example:

The following example shows how to create nicknames for XML data sources by using the sample XML file shown in Figure 44 on page 426.

```
<?xml version="1.0" encoding="UTF-8"?>
<doc>
   <customer id='123'>
      <name>...</name>
      <address>...</address>
       ...
      <order>
         <amount>...</amount>
            <date>...</date>
         <item quant='12'>
            <name>...</name>
         </item>
         <item quant='4'>...</item>
          ...
      </order>
      <order>...</order>
       ...
      <payment>
         <number>...</number>
         <date>...</date>
      </payment>
      <payment>>...</payment>
       ...
   </customer>
   <customer id='124'>...</customer>
</doc>
```

Figure 44. Sample XML file

The parent nickname:

The first step is to create the parent nickname, `customers`. To create the nickname, issue the following statement:

```
CREATE NICKNAME customers
(
   id        VARCHAR(5)    OPTIONS(XPATH './@id')
   name      VARCHAR(16)   OPTIONS(XPATH './name'),
   address   VARCHAR(30)   OPTIONS(XPATH './address/@street'),
   cid       VARCHAR(16) FOR BIT DATA NOT NULL OPTIONS(PRIMARY_KEY 'YES'))
   FOR SERVER xml_server
   OPTIONS(DIRECTORY_PATH '/home/db2user',
         XPATH '/doc/customer', STREAMING 'YES');
```

This statement creates the `customers` nickname over multiple XML files under the specified directory path, `/home/db2user`. The STREAMING nickname option indicates that the XML source data is separated and processed by node (in this example, by customer record). When the STREAMING nickname option is used, the wrapper does not storing the entire XML document into memory. Instead, the XML wrapper divides the document into multiple sections which are parsed individually and sequentially. The STREAMING nickname option should be used only with large XML documents. The performance of your queries is impacted when you use this option.

The child nicknames:

The next step is to create the child nicknames for the `orders`, `payments`, and `items` elements.

Issue the following statement to create the `orders` child nickname.

```
CREATE NICKNAME orders
(
   amount INTEGER       OPTIONS(XPATH './amount'),
   date   VARCHAR(10)   OPTIONS(XPATH './date'),
   oid    VARCHAR(16)   OPTIONS(PRIMARY_KEY 'YES'),
   cid    VARCHAR(16) FOR BIT DATA NOT NULL OPTIONS(FOREIGN_KEY 'CUSTOMERS'))
   FOR SERVER xml_server
   OPTIONS( XPATH './order');
```

Issue the following statement to create the payments child nickname.

```
CREATE NICKNAME payments
(
   number INTEGER       OPTIONS(XPATH './number'),
   date   VARCHAR(10)   OPTIONS(XPATH './date'),
   cid    VARCHAR(16) FOR BIT DATA NOT NULL OPTIONS(FOREIGN_KEY 'CUSTOMERS'))
   FOR SERVER xml_server
   OPTIONS( XPATH './payment');
```

Issue the following statement to create the items child nickname.

```
CREATE NICKNAME items
(
   name      VARCHAR(20)   OPTIONS(XPATH './name'),
   quantity  INTEGER       OPTIONS(XPATH './@quant'),
   oid       VARCHAR(16) FOR BIT DATA NOT NULL OPTIONS(FOREIGN_KEY 'ORDERS'))
   FOR SERVER xml_server
   OPTIONS( XPATH './item');
```

Column option examples:

The following examples show you how to include the DOCUMENT column options when you create nicknames. The examples also show you how those options are used in queries.

The following CREATE NICKNAME example shows the use of the DOCUMENT 'FILE' column option:

```
CREATE NICKNAME customers
(
   doc      VARCHAR(100)   OPTIONS(DOCUMENT 'FILE'),
   name     VARCHAR(16)    OPTIONS(XPATH './name'),
   address  VARCHAR(30)    OPTIONS(XPATH './address/@street'),
   cid      VARCHAR(16) FOR BIT DATA NOT NULL OPTIONS(PRIMARY_KEY 'YES'))
   FOR SERVER xml_server
   OPTIONS(XPATH '/doc/customer');
```

You can then run the following query on the customers nickname, specifying the location of the XML document in the WHERE clause:

```
SELECT * FROM customers WHERE doc = '/home/db2user/Customers.xml';
```

The following CREATE NICKNAME example shows the use of the DOCUMENT 'DIRECTORY' column option:

```
CREATE NICKNAME customers
(
   doc      VARCHAR(100)   OPTIONS(DOCUMENT 'DIRECTORY'),
   name     VARCHAR(16)    OPTIONS(XPATH './name'),
   address  VARCHAR(30)    OPTIONS(XPATH './address/@street'),
   cid      VARCHAR(16) FOR BIT DATA NOT NULL OPTIONS(PRIMARY_KEY 'YES'))
   FOR SERVER xml_server
   OPTIONS(XPATH '/doc/customer');
```

You can then run the following query on the customers nickname:

```
SELECT name FROM customers WHERE doc = '/home/data/xml';
```

This query retrieves the XML documents that are located under the directory path /home/data/xml, which is specified in the WHERE clause.

The following CREATE NICKNAME example shows the use of the DOCUMENT 'URI' nickname column option:

```
CREATE NICKNAME customers
(
   doc      VARCHAR(100)   OPTIONS(DOCUMENT 'URI'),
   name     VARCHAR(16)    OPTIONS(XPATH './name'),
   address  VARCHAR(30)    OPTIONS(XPATH './address/@street'),
   cid      VARCHAR(16) FOR BIT DATA NOT NULL OPTIONS(PRIMARY_KEY 'YES'))
   FOR SERVER xml_server
   OPTIONS(XPATH '/doc/customer');
```

You can then run the following query on the customers nickname to retrieve the XML data from the remote location:

```
SELECT * FROM customers WHERE doc = 'http://www.lg-mv.org/foo.xml';
```

The following CREATE NICKNAME example shows the use of the DOCUMENT 'COLUMN' nickname column option:

```
CREATE NICKNAME emp
(
   doc      VARCHAR(500)  OPTIONS(DOCUMENT 'COLUMN')
   fname    VARCHAR(16)   OPTIONS(XPATH '@first'),
   lname    VARCHAR(16)   OPTIONS(XPATH '@last'))
   FOR SERVER xml_server
   OPTIONS(XPATH '/doc/name');
```

You can then run one of the following queries on the emp nickname to retrieve the XML data:

```
SELECT * FROM emp WHERE doc = '<?xml version="1.0" encoding="UTF-8"?>
        <doc>
        <title>  employees </title>
        <name first="David"  last="Marston"/>
        <name first="Donald" last="Leslie"/>
        <name first="Emily"  last="Farmer"/>
        <name first="Myriam" last="Midy"/>
        <name first="Lee"    last="Tran"/>
        <name first="Lili"   last="Farmer"/>
        <name first="Sanjay" last="Kumar"/>
        </doc>';
```

or

```
SELECT * FROM emp WHERE doc = (SELECT * FROM xml_tab);
```

The xml_tab table contains one column that is populated with the XML data.

View examples:

The following examples show you how to create views for nonroot nicknames to describe XML source documents. In these examples, assume that the nicknames of the sample file shown in Figure 45 on page 429 were previously created as customers, orders, payments, and items.

```
<?xml version="1.0" encoding="UTF-8"?>
<doc>
   <customer id='123'>
      <name>...</name>
      <address>...</address>
       ...
      <order>
         <amount>...</amount>
            <date>...</date>
         <item quant='12'>
            <name>...</name>
         </item>
         <item quant='4'>...</item>
          ...
      </order>
      <order>...</order>
       ...
      <payment>
         <number>...</number>
         <date>...</date>
      </payment>
      <payment>...</payment>
       ...
   </customer>
   <customer id='124'>...</customer>
</doc>
```

Figure 45. Sample XML file

The following example shows how to create a view for the nonroot nickname `order`:

```
CREATE VIEW order_view AS
   SELECT o.amount, o.date, o.oid, c.cid
   FROM customers c, orders o
   WHERE c.cid = o.cid;
```

The following example shows how to create a view for the nonroot nickname `payment`:

```
CREATE VIEW payment_view AS
   SELECT p.number, p.date, c.cid
   FROM customers c, payments p
   WHERE c.cid = p.cid;
```

The following example shows how to create a view for the nonroot nickname `item`:

```
CREATE VIEW item_view AS
   SELECT i.quantity, i.name, o.oid
   FROM customers c, orders o, items i
   WHERE c.cid = o.cid AND o.oid = i.oid;
```

Queries that are submitted to these views are processed correctly because the join path to the root directory is present.

For example, the following query pairs the amounts of the orders and payments from the same date for customers:

```
SELECT o.amount, p.amount
FROM order_view o, payment_view p
WHERE p.date = o.date AND
   p.cid = o.cid;
```

Related tasks:

- “Registering nicknames for XML data sources” on page 424

Related reference:

- Appendix G, “Nickname column options for federated systems,” on page 603
- Appendix F, “Nickname options for federated systems,” on page 593
- “CREATE NICKNAME statement syntax - XML wrapper” on page 557

Creating federated views for nonroot nicknames (XML wrapper)

Creating federated views for nonroot nicknames (XML wrapper) is part of the larger task of adding XML to a federated system.

You can define federated views over the hierarchy of nicknames that describe an XML document. Defining federated views ensures that the queries that join pieces of an XML nickname hierarchy (not including the root nickname and queries that join columns other than the special PRIMARY_KEY and FOREIGN_KEY columns) run properly.

Procedure:

To define federated views that include all required predicates and a full path to the root directory, follow these steps:

1. Define a view for each nonroot nickname as a join of all the nicknames on the path to the root.
2. In the WHERE clause, make the join predicates over the PRIMARY_KEY and FOREIGN_KEY columns.
3. In the SELECT list, include all the columns of the nonroot nickname except the column that is designated with the FOREIGN_KEY nickname column option.
4. In the SELECT list, include the column of the parent nickname designated with the PRIMARY_KEY option.

Related reference:

- “CREATE NICKNAME statement - Examples for XML wrapper” on page 425

Optimization tips for the XML cost model facility

The cost model facility for the XML wrapper helps optimize queries on the nicknames that you create.

The cost model facility uses the following parameters of the CREATE NICKNAME statement:

- INSTANCE_PARSE_TIME
- XPATH_EVAL_TIME

You can specify values for these parameters when you issue a CREATE NICKNAME statement to register a nickname for an XML data source.

The cost model facility uses these parameter values when determining the amount of time required to parse data in each row of an XML source document and to evaluate the nickname’s XPath expression.

You can use the default values for these parameters. However, if you want to optimize queries on large or complex XML source structures for the nicknames that you create, use the following example as a guide.

An example of optimizing a large query:

Assume that your XML document has a relational schema with four nicknames:
- customers
- orders
- payments
- items

Also, assume that the customers nickname is the root nickname.

Run queries on each nickname. Run each query on a sample of the XML data that is typical for your environment.

For example:

```
SELECT * from customers;
SELECT * from orders;
SELECT * from payments;
SELECT * from items;
```

Note the amount of time (in milliseconds) that is required to run each query by using the **db2batch** command or equivalent command or utility. (You can use the **db2batch** command to obtain an output file that contains the time required to run queries.) Also, note the number of tuples that are returned.

For each nickname, use the following formulas to determine the optimal values for the INSTANCE_PARSE_TIME and XPATH_EVAL_TIME parameters:

```
INSTANCE_PARSE_TIME = (75%  X  run time of SELECT * query) ÷ number of tuples returned
XPATH_EVAL_TIME     = (25%  X  run time of SELECT * query) ÷ number of tuples returned
```

For the root nickname (in the example, `customers`), use the calculated values for the INSTANCE_PARSE_TIME and XPATH_EVAL_TIME parameters.

For nonroot nicknames, (in the example, `orders`, `payments`, and `items`), use only the calculated value for the XPATH_EVAL_TIME parameter. The INSTANCE_PARSE_TIME parameter value is not applicable for nonroot nicknames.

You can use these formulas as a guide for tuning your queries. The optimal values for these parameters also depend on the complexity of your XML source documents and on the speed of the processor that you are using.

Related concepts:
- "What is XML?" on page 415
- "Data associations between nicknames and XML documents" on page 422
- "The cost model facility for the XML wrapper" on page 424

Related reference:
- "db2batch - Benchmark Tool Command" in the *Command Reference*

XML data source - Example queries

This topic provides several sample queries that use the nicknames customers, orders, and items. These nicknames were previously registered by using CREATE NICKNAME statements.

The following query displays all customer names:

```
SELECT name FROM customers;
```

The following query displays all records in which the customer name is Chang:

```
SELECT * FROM customers
    WHERE name='Chang';
```

The following query displays the customer names and amounts for each order of each customer:

```
SELECT c.name, o.amount FROM customers c, orders o
    WHERE c.cid=o.cid;
```

You must specify the join, c.cid=o.cid, to indicate the parent-child relationship between the customers nickname and the orders nickname.

The following query selects the customer addresses, order amounts, and item names for each order and item of each customer:

```
SELECT c.address, o.amount, i.name FROM customers c, orders o, items i
    WHERE c.cid=o.cid AND o.oid=i.oid;
```

You must specify the two joins to maintain the parent-child relationships.

The following examples show how to write queries by using a nickname that specifies a DOCUMENT column option rather than a FILE_PATH nickname option. The corresponding CREATE NICKNAME statement that is used to create the customers nickname is shown here:

```
CREATE NICKNAME customers
(
   doc      VARCHAR(100)   OPTIONS(DOCUMENT 'FILE'),
   name     VARCHAR(16)    OPTIONS(XPATH './name'),
   address  VARCHAR(30)    OPTIONS(XPATH './address/@street'),
   cid      VARCHAR(16)    OPTIONS(PRIMARY_KEY 'YES'))
   FOR SERVER xml_server
   OPTIONS(XPATH '/doc/customer');
```

The following query selects all the data from the XML file `Customers.xml` with a file path of `/home/db2user/Customers.xml`:

```
SELECT * FROM customers
   WHERE doc='/home/db2user/Customers.xml';
```

The following query selects names of customers and dates of their orders from the `Customers.xml` file for each order with an amount over 1000:

```
SELECT c.name, o.date FROM customers c, orders o
   WHERE c.doc='/home/db2user/Customers.xml' AND o.amount > 1000;
```

The file path of `/home/db2user/Customers.xml` specifies the location of the `Customers.xml` file.

Related reference:

- "CREATE NICKNAME statement syntax - XML wrapper" on page 557

- “CREATE NICKNAME statement - Examples for XML wrapper” on page 425

Messages for the XML wrapper

This topic describes messages that you might encounter when working with the wrapper for XML. For more information about messages, see the *DB2 Message Reference*.

Table 92. Messages issued by the wrapper for XML

Error Code	Message	Explanation
SQL0405N	The numeric literal "<column_name>" is not valid because its value is out of range.	The specified numeric literal is not within the acceptable range. Check the data type of the column in the CREATE NICKNAME statement.
SQL0408N	A value is not compatible with the data type of its assignment target. Target name is "<column_name>."	The data type of the value that is being assigned to the column is not compatible with the declared data type of the assignment target. Check the data type of the column in the CREATE NICKNAME statement.
SQL0901N	The SQL statement failed because of a non-severe system error. Subsequent SQL statements can be processed. (Reason "Error creating wrapper object.")	An error occurred when creating a new wrapper object. Contact IBM Software Support.
SQL0901N	The SQL statement failed because of a non-severe system error. Subsequent SQL statements can be processed. (Reason "<xerces_xalan_error_message>.")	An error occurred during a call to a Xerces or a Xalan function. Check the XML document. If the document is well structured, refer to the Xalan documentation for more information about the error message.
SQL0901N	The SQL statement failed because of a non-severe system error. Subsequent SQL statements can be processed. (Reason "XalanDOMException: exception code is <exception_code>.")	A XalanDOMException exception occurred. Refer to the Xalan documentation for more information about the exception code.
SQL0901N	The SQL statement failed because of a non-severe system error. Subsequent SQL statements can be processed. (Reason "XMLException: <exception_error_message>.")	An XMLException exception occurred. Refer to the Xalan documentation for more information about the exception code.
SQL0901N	The SQL statement failed because of a non-severe system error. Subsequent SQL statements can be processed. (Reason "XSLException: <exception_error_message>.")	An XSLException exception occurred. Refer to the Xalan documentation for more information about the exception code.
SQL0901N	The SQL statement failed because of a non-severe system error. Subsequent SQL statements can be processed. (Reason "SAXParseException: <exception_error_message>.")	A SAXParseException exception occurred. Refer to the Xalan documentation for more information about the exception code.

Table 92. Messages issued by the wrapper for XML (continued)

Error Code	Message	Explanation
SQL0901N	The SQL statement failed because of a non-severe system error. Subsequent SQL statements can be processed. (Reason "Error getting node value.")	Xalan tried to access a node that is not valid. Contact IBM Software Support.
SQL0901N	The SQL statement failed because of a non-severe system error. Subsequent SQL statements can be processed. (Reason "Error parsing XML document.")	An error occurred when parsing the XML document. Check the XML document.
SQL0901N	The SQL statement failed because of a non-severe system error. Subsequent SQL statements can be processed. (Reason "Error getting root element of XML document.")	After parsing the XML document, Xalan tried to retrieve the root element but failed. Check the XML document.
SQL0901N	The SQL statement failed because of a non-severe system error. Subsequent SQL statements can be processed. (Reason "Unspecified exception while evaluating XPath expression.")	Xalan generated an unspecified exception when evaluating an XPath expression. Check the XML document, and refer to the Xalan documentation.
SQL0901N	The SQL statement failed because of a non-severe system error. Subsequent SQL statements can be processed. (Reason "Unspecified exception while getting node value.")	Xalan generated an unspecified exception when retrieving a node value. Check the XML document, and refer to the Xalan documentation.
SQL0901N	The SQL statement failed because of a non-severe system error. Subsequent SQL statements can be processed. (Reason "Unspecified exception while parsing input document.")	Xalan generated an unspecified exception when parsing the XML document. Check the XML document, and refer to the Xalan documentation.
SQL0901N	The SQL statement failed because of a non-severe system error. Subsequent SQL statements can be processed. (Reason "Error when evaluating cardinality.")	Contact IBM Software Support.
SQL0901N	The SQL statement failed because of a non-severe system error. Subsequent SQL statements can be processed. (Reason "<SOAP_error_message>.")	The SOAP library issued an error. If you cannot resolve the SQL statement error, contact IBM Software Support.
SQL0901N	The SQL statement failed because of a non-severe system error. Subsequent SQL statements can be processed. (Reason "Invalid URI.")	The wrapper cannot access the specified URL. Verify that the URL is accessible.
SQL0901N	The SQL statement failed because of a non-severe system error. Subsequent SQL statements can be processed. (Reason "Invalid XML document content.")	The content of the XML document is not valid. Verify that the document is well structured.

Table 92. Messages issued by the wrapper for XML (continued)

Error Code	Message	Explanation
SQL0901N	The SQL statement failed because of a non-severe system error. Subsequent SQL statements can be processed. (Reason "Invalid SOAP envelope.")	The SOAP envelope is not valid. Check its syntax and content.
SQL0901N	The SQL statement failed because of a non-severe system error. Subsequent SQL statements can be processed. (Reason "Memory allocation error.")	An error occurred when allocating memory.
SQL1822N	Unexpected error code "<trace_point>" received from data source "XML wrapper." Associated text and tokens are "Incorrect DATE format."	A date value in the XML document does not have the correct format. The valid format for date values is yyyy-mm-dd. Check the XML document.
SQL1822N	Unexpected error code "<trace_point>" received from data source "XML wrapper." Associated text and tokens are "Column data type not supported."	A nickname column has an unsupported data type. Check the CREATE NICKNAME statement.
SQL1822N	Unexpected error code "<trace_point>" received from data source "XML wrapper." Associated text and tokens are "TYPE clause not supported."	The CREATE SERVER statement contains a TYPE clause. This clause is not supported by the XML wrapper. Remove the clause.
SQL1822N	Unexpected error code "<trace_point>" received from data source "XML wrapper." Associated text and tokens are "VERSION clause not supported."	The CREATE SERVER statement contains a VERSION clause. This clause is not supported by the XML wrapper. Remove the clause.
SQL1822N	Unexpected error code "<trace_point>" received from data source "XML wrapper." Associated text and tokens are "Invalid use of predicate with DOCUMENT column."	The query contains a predicate with incorrect operands. Check the predicates in the query.
SQL1822N	Unexpected error code "<trace_point>" received from data source "XML wrapper." Associated text and tokens are "Invalid use of predicate with FOREIGN_KEY column."	The query contains a predicate with incorrect operands. Check the predicates in the query.
SQL1822N	Unexpected error code "<trace_point>" received from data source "XML wrapper." Associated text and tokens are "Invalid use of predicate with PRIMARY_KEY column."	The query contains a predicate with incorrect operands. Check the predicates in the query.

Table 92. Messages issued by the wrapper for XML (continued)

Error Code	Message	Explanation
SQL1822N	Unexpected error code "<trace_point>" received from data source "XML wrapper." Associated text and tokens are "XPATH and DOCUMENT options not compatible."	The CREATE NICKNAME statement is not correct as specified. Check the syntax of the statement.
SQL1822N	Unexpected error code "<trace_point>" received from data source "XML wrapper." Associated text and tokens are "XPATH and FOREIGN_KEY options not compatible."	The CREATE NICKNAME statement is not correct as specified. Check the syntax of the statement.
SQL1822N	Unexpected error code "<trace_point>" received from data source "XML wrapper." Associated text and tokens are "XPATH and PRIMARY_KEY options not compatible."	The CREATE NICKNAME statement is not correct as specified. Check the syntax of the statement.
SQL1822N	Unexpected error code "<trace_point>" received from data source "XML wrapper." Associated text and tokens are "DOCUMENT and FOREIGN_KEY options not compatible."	The CREATE NICKNAME statement is not correct as specified. Check the syntax of the statement.
SQL1822N	Unexpected error code "<trace_point>" received from data source "XML wrapper." Associated text and tokens are "DOCUMENT and PRIMARY_KEY options not compatible."	The CREATE NICKNAME statement is not correct as specified. Check the syntax of the statement.
SQL1822N	Unexpected error code "<trace_point>" received from data source "XML wrapper." Associated text and tokens are "FOREIGN_KEY and PRIMARY_KEY options not compatible."	The CREATE NICKNAME statement is not correct as specified. Check the syntax of the statement.
SQL1822N	Unexpected error code "<trace_point>" received from data source "XML wrapper." Associated text and tokens are "Column option missing."	The CREATE NICKNAME statement is not correct as specified. Check the syntax of the statement.
SQL1822N	Unexpected error code "<trace_point>" received from data source "XML wrapper." Associated text and tokens are "DOCUMENT column option not unique."	The CREATE NICKNAME statement is not correct as specified. Check the syntax of the statement.
SQL1822N	Unexpected error code "<trace_point>" received from data source "XML wrapper." Associated text and tokens are "FOREIGN_KEY column option not unique."	The CREATE NICKNAME statement is not correct as specified. Check the syntax of the statement.

Table 92. Messages issued by the wrapper for XML (continued)

Error Code	Message	Explanation
SQL1822N	Unexpected error code "<trace_point>" received from data source "XML wrapper." Associated text and tokens are "PRIMARY_KEY column option not unique."	The CREATE NICKNAME statement is not correct as specified. Check the syntax of the statement.
SQL1822N	Unexpected error code "<trace_point>" received from data source "XML wrapper." Associated text and tokens are "Invalid DOCUMENT option value."	The value of the DOCUMENT option that is specified in the CREATE NICKNAME statement is not valid. The value must be FILE. Check the CREATE NICKNAME statement.
SQL1822N	Unexpected error code "<trace_point>" received from data source "XML wrapper." Associated text and tokens are "Invalid PRIMARY_KEY option value."	The value of the PRIMARY_KEY option that is specified in the CREATE NICKNAME statement is not valid. The value must be YES. Check the CREATE NICKNAME statement.
SQL1822N	Unexpected error code "<trace_point>" received from data source "XML wrapper." Associated text and tokens are "Invalid FOREIGN_KEY option value."	The value of the FOREIGN_KEY option that is specified in the CREATE NICKNAME statement is not valid. The value does not match any parent nickname. Check the CREATE NICKNAME statement.
SQL1822N	Unexpected error code "<trace_point>" received from data source "XML wrapper." Associated text and tokens are "FILE_PATH and DOCUMENT options not compatible."	The CREATE NICKNAME statement is not correct as specified. The FILE_PATH and DOCUMENT options cannot be specified at the same time. Check the syntax of the CREATE NICKNAME statement.
SQL1822N	Unexpected error code "<trace_point>" received from data source "XML wrapper." Associated text and tokens are "FILE_PATH and SOAP options not compatible."	The CREATE NICKNAME statement is not correct as specified. The FILE_PATH and SOAP options cannot be specified at the same time. Check the syntax of the CREATE NICKNAME statement.
SQL1822N	Unexpected error code "<trace_point>" received from data source "XML wrapper." Associated text and tokens are "DIRECTORY_PATH and SOAP options not compatible."	The CREATE NICKNAME statement is not correct as specified. The DIRECTORY_PATH and SOAP options cannot be specified at the same time. Check the syntax of the CREATE NICKNAME statement.
SQL1822N	Unexpected error code "<trace_point>" received from data source "XML wrapper." Associated text and tokens are "FILE_PATH and DIRECTORY_PATH options not compatible."	The CREATE NICKNAME statement is not correct as specified. The FILE_PATH and DIRECTORY_PATH options cannot be specified at the same time. Check the syntax of the CREATE NICKNAME statement.
SQL1822N	Unexpected error code "<trace_point>" received from data source "XML wrapper." Associated text and tokens are "VALIDATE and STREAMING options not compatible."	The CREATE NICKNAME statement is not correct as specified. The VALIDATE and STREAMING options cannot be specified at the same time. Check the syntax of the CREATE NICKNAME statement.

Table 92. Messages issued by the wrapper for XML (continued)

Error Code	Message	Explanation
SQL1822N	Unexpected error code "<trace_point>" received from data source "XML wrapper." Associated text and tokens are "FILE_PATH and FOREIGN_KEY options not compatible."	The CREATE NICKNAME statement is not correct as specified. The FILE_PATH and FOREIGN_KEY options cannot be specified at the same time. Check the syntax of the CREATE NICKNAME statement.
SQL1822N	Unexpected error code "<trace_point>" received from data source "XML wrapper." Associated text and tokens are "DIRECTORY_PATH and FOREIGN_KEY options not compatible."	The CREATE NICKNAME statement is not correct as specified. The DIRECTORY_PATH and FOREIGN_KEY options cannot be specified at the same time. Check the syntax of the CREATE NICKNAME statement.
SQL1822N	Unexpected error code "<trace_point>" received from data source "XML wrapper." Associated text and tokens are "XPATH option value not valid with STREAMING enabled."	The nickname XPATH expression is not valid when you enable the STREAMING feature. Check the XPATH option for values that are not valid such as `/`, `./`, and `//`.
SQL1822N	Unexpected error code "<trace_point>" received from data source "XML wrapper." Associated text and tokens are "Unable to read XML file."	The file path that is specified in the CREATE NICKNAME statement or in the query is not valid. The specified file does not exist. Check the CREATE NICKNAME statement and the query.
SQL1822N	Unexpected error code "<trace_point>" received from data source "XML wrapper." Associated text and tokens are "Unable to open directory."	The directory path that is specified in the CREATE NICKNAME statement or in the query is not valid. The specified directory does not exist. Check the CREATE NICKNAME statement and the query.
SQL1822N	Unexpected error code "<trace_point>" received from data source "XML wrapper." Associated text and tokens are "Reference to XML data missing."	The CREATE NICKNAME statement must contain a reference to the XML data. Check the CREATE NICKNAME statement.
SQL1822N	Unexpected error code "<trace_point>" received from data source "XML wrapper." Associated text and tokens are "DOCUMENT column option with value 'SOAP' missing."	The CREATE NICKNAME statement is not correct as specified. Check the value of the DOCUMENT option. The value must be SOAP.
SQL1822N	Unexpected error code "<trace_point>" received from data source "XML wrapper." Associated text and tokens are "SOAP option missing."	The CREATE NICKNAME statement is not correct as specified. You must specify the SOAP option.
SQL1822N	Unexpected error code "<trace_point>" received from data source "XML wrapper." Associated text and tokens are "INSTANCE_PARSE_TIME only for root nicknames."	The CREATE NICKNAME statement is not correct as specified. You can specify an INSTANCE_PARSE_TIME value only for root nicknames. Check the CREATE NICKNAME syntax.

Table 92. Messages issued by the wrapper for XML (continued)

Error Code	Message	Explanation
SQL1822N	Unexpected error code "<trace_point>" received from data source "XML wrapper." Associated text and tokens are "VALIDATE option only for root nicknames."	The CREATE NICKNAME statement is not correct as specified. You can set the VALIDATE option to YES only if the specified nickname is a root nickname. Check the CREATE NICKNAME syntax.
SQL1822N	Unexpected error code "<trace_point>" received from data source "XML wrapper." Associated text and tokens are "STEAMING option only for root nicknames."	The CREATE NICKNAME statement is not correct as specified. You can set the STREAMING option to YES only if the specified nickname is a root nickname. Check the CREATE NICKNAME syntax.
SQL1823N	No data type mapping exists for data type "<data_type_name>" from server "<server_name>."	The CREATE NICKNAME statement is not correct as specified. A column data type is not valid. Check the CREATE NICKNAME syntax.
SQL1881N	"<option_name>" is not a valid "<option_type>" option for "<object_name>."	The specified option might not exist or might not be valid for this data source. Check the CREATE NICKNAME statement.
SQL1881N	"DIRECTORY_PATH" is not a valid "NICKNAME" option for "<object_name>."	The value of the DIRECTORY_PATH option that is specified in the CREATE NICKNAME statement is not valid. The specified directory must be a root directory. Check the CREATE NICKNAME statement.
SQL1882N	The "nickname" option "VALIDATE" cannot be set to "<option_value>" for "<object_name>."	The value of the VALIDATE option that is specified in the CREATE NICKNAME statement is not valid. This value must be either YES or NO. Check the CREATE NICKNAME statement.
SQL1882N	The "nickname" option "STREAMING" cannot be set to "<option_value>" for v<object_name>."	The value of the STREAMING option that is specified in the CREATE NICKNAME statement is not valid. This value must be either YES or NO. Check the CREATE NICKNAME statement.
SQL1883N	"<option_name>" is a required "<option_type>" option for "<object_name>."	A required DB2 option was not specified. Check the CREATE NICKNAME statement.

Related concepts:

- "Introduction to messages" in the *Message Reference Volume 1*

Related reference:

- "SQLSTATE messages" in the *Message Reference Volume 2*

Part 4. User-defined functions

Chapter 25. Life sciences user-defined functions

This chapter explains what the life sciences user-defined functions are, how to add them to your federated system, and how to use them in your queries.

Life sciences user-defined functions - overview

Life sciences user-defined functions - overview

The life sciences user-defined functions provide you with algorithms that you commonly use to analyze data.

The life sciences user-defined functions use the standard single-letter codes and the IUPAC-IUB ambiguity codes to represent amino acids and nucleotides.

| The life sciences user-defined functions are installed with the Life Sciences
| User-Defined Functions component of the nonrelational wrappers. After the life
| sciences user-defined functions are installed, you must register the functions.

To avoid conflicts with namespaces, all of the life sciences user-defined functions are registered in the DB2LS schema.

Related concepts:

- "DB2 Information Integrator Nonrelational Wrappers" in the *IBM DB2 Information Integrator Installation Guide for Linux, UNIX, and Windows*
- "DB2 Information Integrator Relational Wrappers" in the *IBM DB2 Information Integrator Installation Guide for Linux, UNIX, and Windows*

Related tasks:

- "Registering life sciences user-defined functions" on page 445
- "Disabling the life sciences user-defined functions" on page 446

Related reference:

- "Life sciences user-defined function library files" on page 443
- "Life sciences user-defined functions by functional category" on page 444

Life sciences user-defined function library files

Some of the user-defined functions that are included with DB2 Information Integrator require library files. These library files are required when you register the user-defined functions on the federated server.

When you install DB2 Information Integrator for nonrelational data sources, the following user-defined library files are installed on your federated server.

Life sciences user-defined function libraries:

Table 93. Life sciences user-defined function library locations and file names

Function type	Operating system	Directory path	Library file name
Life sciences user-defined functions	AIX	/SQLLIB/function	libdb2lsudfs.a
Life sciences user-defined functions	HP-UX	/SQLLIB/function	libdb2lsudfs.sl
Life sciences user-defined functions	Linux	/SQLLIB/function	libdb2lsudfs.so
Life sciences user-defined functions	Solaris	/SQLLIB/function	libdb2lsudfs.so
Life sciences user-defined functions	Windows	%DB2PATH%\bin	db2lsudfs.dll

The default Windows directory path is `C:\Program Files\IBM\SQLLIB`. %DB2PATH% is the environment variable that is used to specify the directory path where DB2 Information Integrator is installed on Windows.

LSGeneWise user-defined function library:

The LSGeneWise user-defined function requires a separate library.

Table 94. LSGeneWise function library location and file name

Function type	Operating system	Directory path	Library file name
LSGeneWise function	UNIX	/SQLLIB/lib	libdb2lsSTgenewise.a

Related concepts:
- "Life sciences user-defined functions - overview" on page 443

Life sciences user-defined functions by functional category

Table 95 lists the life sciences user-defined functions by functional category. It also provides a brief description of each category.

Table 95. Life sciences user-defined functions

Functional category	User-defined functions	Description
Back translate	LSPep2AmbNuc, LSPep2ProbNuc	Converts an amino acid sequence into a nucleotide sequence.
Defline parsing	LSDeflineParse	Parses elements of a definition line, such as that returned by the BLAST wrapper or present in a FASTA format data file.
Generalized pattern matching	LSPatternMatch, LSPrositePattern	Identifies areas of interest in a given string, such as a nucleotide or peptide sequence.
GeneWise	LSGeneWise	Aligns a protein sequence to a genomic sequence.
Motifs	LSMultiMatch, LSMultiMatch3, LSBarCode	Matches patterns in nucleotide or amino acid sequences.

Table 95. Life sciences user-defined functions (continued)

Functional category	User-defined functions	Description
Reverse	LSRevNuc, LSRevPep, LSRevComp	Reverses a nucleotide or amino acid sequence.
Translate	LSNuc2Pep, LSTransAllFrames	Converts a nucleotide sequence into a peptide sequence.

Related concepts:
- "Life sciences user-defined functions - overview" on page 443

Related tasks:
- "Registering life sciences user-defined functions" on page 445
- "Disabling the life sciences user-defined functions" on page 446

Registering life sciences user-defined functions

Before you can use the life sciences user-defined functions, you must register the functions.

Prerequisites:

| The life sciences user-defined functions component of the nonrelational wrappers
| must be installed with DB2 Information Integrator.

Procedure:

To register the life sciences user-defined functions, use the enable_LSFunctions command.
- On federated servers that run Windows NT, this command is in the `sqllib\bin` directory
- On federated servers that run AIX, this command is in the `sqllib/bin` directory

The syntax for the enable_LSFunctions command is:

```
enable_LSFunctions -n dbName -u userID -p password [-force]
```

dbName
: The name of the federated database that you are registering the functions in.

userID
: A valid user ID for the federated database.

password
: A valid password for the user ID.

force
: A flag that you can use to remove the functions and register them again. Use this flag to register the functions again if the functions get corrupted or dropped accidentally.

The enable_LSFunctions command registers the all of the life sciences user-defined functions in the federated database. The functions are registered with the schema name DB2LS.

The following example shows the output that is returned when you issue the enable_LSFunctions command:

```
C:> enable_LSFunctions -n federateddb -u db2admin -p db2admin

(0) Life Sciences Functions were found
    -- Create Life Sciences Functions ...
    Create Life Sciences Functions Successfully.

*** Please allow a few seconds to clean up the system ......
```

The following example shows the output that is returned when you issue the enable_LSFunctions command when you use the force flag and the functions are already registered:

```
C:> enable_LSFunctions -n federateddb -u db2admin -p db2admin -force

(21) Life Sciences Functions were found

Life Sciences functions already exist ...
Reinstall Life Sciences functions ...
    -- Drop Life Sciences Functions ...
    Drop Life Sciences Functions Successfully.
    -- Create Life Sciences Functions ...
    Create Life Sciences Functions Successfully.


*** Please allow a few seconds to clean up the system ......
```

Related concepts:

- “Life sciences user-defined functions - overview” on page 443

Related tasks:

- “Adding relational wrappers, nonrelational wrappers, and user-defined functions to your DB2 Information Integrator system” in the *IBM DB2 Information Integrator Installation Guide for Linux, UNIX, and Windows*
- “Disabling the life sciences user-defined functions” on page 446

Related reference:

- “Life sciences user-defined functions by functional category” on page 444

Disabling the life sciences user-defined functions

If you no longer want to use the life sciences user-defined functions, you can temporarily disable the functions or permanently remove the functions from your federated database.

Procedure:

To disable the life sciences user-defined functions, use the **disable_LSFunctions** command.

- On federated servers that run Windows NT, this command is in the `sqllib\bin` directory
- On federated servers that run AIX, this command is in the `sqllib/bin` directory

The syntax for the **disable_LSFunctions** command is:

```
disable_LSFunctions -n dbName -u userID -p password
```

dbName
The name of the federated database that you want to disable the functions from.

userID
A valid user ID for the federated database.

password
A valid password for the user ID.

Example of disabling the life sciences user-defined functions:

The following example shows the output that is returned when you issue the **disable_LSFunctions** command:

```
C:>disable_LSFunctions -n federateddb -u db2admin -p db2admin

(21) Life Sciences Functions were found
    -- Drop Life Sciences Functions ...
    Drop Life Sciences Functions Successfully.


*** Please allow a few seconds to clean up the system ......
```

You must uninstall the functions to permanently remove the functions from the federated database.

Related tasks:
- "Removing relational wrappers, nonrelational wrappers, and life sciences user-defined functions (Windows)" in the *IBM DB2 Information Integrator Installation Guide for Linux, UNIX, and Windows*
- "Removing wrappers, user-defined functions, and the wrapper development kits (UNIX)" in the *IBM DB2 Information Integrator Installation Guide for Linux, UNIX, and Windows*
- "Registering life sciences user-defined functions" on page 445

Back translation user-defined functions

Back translation user-defined functions - overview

Use the back–translation user-defined functions to convert a peptide sequence to a nucleotide sequence. Back translation is the inverse of translation.

Because the mapping from amino acids to nucleotide triplet codons is one-to-many, a back translation produces two results:

most ambiguous
Simple text conversion and lookup. Use the LSPep2AmbNuc user-defined function to do a most ambiguous translation.

most probable
Requires additional information from a codon frequency table. Use the LSPep2ProbNuc user-defined function to do a most probable translation.

Related concepts:
- "Life sciences user-defined functions - overview" on page 443

Related reference:

- "LSPep2AmbNuc user-defined function" on page 448
- "LSPep2AmbNuc user-defined function - error messages" on page 450
- "LSPep2ProbNuc user-defined function" on page 451
- "LSPep2ProbNuc user-defined function - error messages" on page 452
- "LSPep2AmbNuc user-defined function - example" on page 449
- "LSPep2ProbNuc user-defined function - example" on page 451

LSPep2AmbNuc user-defined function

```
►►──DB2LS.LSPep2AmbNuc──(input peptide sequence──┬──────────────────────────────────────┬──)──────►◄
                                                 └─,filepath to external translation table─┘
```

input peptide sequence
: A valid character string representation describing a peptide sequence. A character string representation must have a data type of VARCHAR and an actual length that is no greater than 10890 bytes. The input data uses the standard amino acid symbols and ambiguity codes.

filepath to external translation table
: If you use a customized translation table, include the file path information to find the translation table. The string value of the path must be no greater than 255 characters.

The schema name is DB2LS.

Use the LSPep2AmbNuc function to produce the most ambiguous nucleotide sequence, according to a translation table, from a peptide sequence.

The result of the function is a character string with a data type of VARCHAR and an actual length that is not greater than 32672 bytes. The result represents the most ambiguous nucleotide sequence, according to a translation table, either built-in or specified by you.

If you do not specify a translation table, the function uses Table 96 by default.

Table 96. Default translation table

Amino acid symbol	Abbreviation	Codon
A	Ala	GCX
B	Asx	RAY
C	Cys	TGY
D	Asp	GAY
E	Glu	GAR
F	Phe	TTY
G	Gly	GGX
H	His	CAY
I	Ile	ATH
K	Lys	AAR
L	Leu	YTX
M	Met	ATG
N	Asn	AAY

Table 96. Default translation table (continued)

Amino acid symbol	Abbreviation	Codon
P	Pro	CCX
Q	Gln	CAR
R	Arg	MGX
S	Ser	WSX
T	Thr	ACX
V	Val	GTX
W	Trp	TGG
X	Xxx	XXX
Y	Tyr	TAY
Z	Glx	SAR
*	End	TRR

Related reference:
- "LSPep2AmbNuc user-defined function - error messages" on page 450
- "LSPep2ProbNuc user-defined function" on page 451
- "LSPep2AmbNuc user-defined function - example" on page 449

LSPep2AmbNuc user-defined function - example

You can invoke the function with a values statement. The single input is a peptide sequence, as in the following example:

```
values db2ls.LSPep2AmbNuc('HR');
```

The above example transforms a peptide into a nucleotide using the ambiguous translations and the built-in translation table. The result of the above statement is a nucleotide sequence created from the standard amino acid symbols:

```
CAYMGX
```

The following example transforms a peptide into a nucleotide using the ambiguous translations and the built-in table:

```
values db2ls.LSPep2AmbNuc('SRGFGFITYSHSSMIDEAQKSRPHKIDGRVVEPKRA');
```

The result of this values statement is the following nucleotide sequence. (The sequence has been split to fit on the page.)

```
WSXMGXGGXTTYGGXTTYATHACXTAYWSXCAYWSXWSXATGATHGAYGARGCXCARA
ARWSXMGXCCXCAYAARATHGAYGGXMGXGTXGTXGARCCXAARMGXGCX
```

The next example shows the function applied to a set of values extracted from a table or nickname:

```
SELECT DB2LS.LsPep2AmbNuc(peptide_seq) FROM table protein_table;
```

The data in column peptide_seq of table protein_table looks like the following:

Table 97. Data in the peptide_seq column

peptide_seq
GIKEDTEEHHLRDYFE
QKYHTVNGHNCEVRKA
.....

The result of the select statement is:

```
GGXATHAARGARGAYACXGARGARCAYCAYYTXMGXGAYTAYTTYGAR
CARAARTAYCAYACXGTXAAYGGXCAYAAYTGYGARGTXMGXAARGCX
...
```

The following example transforms a peptide into a nucleotide using the ambiguous translations and a user-defined table. Usually, the differences between translation tables are small. There might be just one or two symbols that are unique. They might occur because some species have more codons or some species have fewer codons. For example, the codon AGG is absent in Drosophila.

```
values db2ls.LSPep2AmbNuc('RGNMGGGNYGNQNGGGNWNNG',
                          '\data\transl_table_06.txt')
```

Assuming the input translation table is for Drosophila, the result of the values statement is shown in the following example:

```
MGRGGXAAYATGGGXGGXGGXAAYTAYGGXAAYTARAAYGGXGGXGGXAAYTGGAAYAAYGGX
```

Related reference:

- "LSPep2AmbNuc user-defined function" on page 448
- "LSNuc2Pep user-defined function – example" on page 476

LSPep2AmbNuc user-defined function - error messages

Table 98. Messages issued by the LSPep2AmbNuc user-defined function

Error code	Message	Explanation
SQL0443N	Routine ″DB2LS.LSPEP2AMBNUC″ (specific name ″LSPEP2AMBNUC″) has returned an error SQLSTATE with diagnostic text ″Sequence not valid″. SQLSTATE=38608	The sequence given is invalid.
SQL0443N	Routine ″DB2LS.LSPEP2AMBNUC″ (specific name ″LSPEP2AMBNUCUT″) has returned an error SQLSTATE with diagnostic text ″No translation found″. SQLSTATE=38610	The translation table file is empty.
SQL0443N	Routine ″LSPEP2AMBNUC″ (specific name ″LSPEP2AMBNUCUT″) has returned an error SQLSTATE with diagnostic text ″Can not open the translation table file″. SQLSTATE=38612	The translation table file specified does not exist.
SQL0443N	Routine ″DB2LS.LSPEP2AMBNUC″ (specific name ″LSPEP2AMBNUCUT″) has returned an error SQLSTATE with diagnostic text ″Line too long reading from file″. SQLSTATE=38614	The file contained a line that was longer than is allowed.

Table 98. Messages issued by the LSPep2AmbNuc user-defined function (continued)

Error code	Message	Explanation
SQL0443N	Routine ″DB2LS.LSPEP2AMBNUC″ (specific name ″LSPEP2AMBNUCUT″) has returned an error SQLSTATE with diagnostic text ″Invalid data file″. SQLSTATE=38615	The file format is invalid.
SQL0443N	Routine ″LSPEP2AMBNUC″ (specific name ″LSPEP2AMBNUCUT″) has returned an error SQLSTATE with diagnostic text ″Can't construct the translation table″. SQLSTATE=38611	Invalid symbols were found in the file.

Related reference:

- "LSPep2AmbNuc user-defined function" on page 448

LSPep2ProbNuc user-defined function

```
►►──DB2LS.LSPep2ProbNuc──(input peptide sequence──┬─────────────────────────────────────┬──)──────────►◄
                                                  └─,filepath to codon frequency table──┘
```

input peptide sequence
A valid character string representation describing a peptide sequence. The character string representation must have a data type of VARCHAR and an actual length that is no greater than 10890 bytes. The input data uses the standard amino acid symbols.

filepath to codon frequency table
This is the codon frequency table. Include the file path information to find the frequency table. The string value of the path must be no greater than 255 characters.

The schema name is DB2LS.

Use the LSPep2ProbNuc function to generate the most probable nucleotide sequence, from a peptide sequence, based on the codon frequency table specified in the second argument.

The result of the function is a character string with a data type of VARCHAR and an actual length that is not greater than 32672 bytes representing the most probable nucleotide sequence using the codon frequency table.

Related reference:

- "LSPep2AmbNuc user-defined function" on page 448
- "LSPep2ProbNuc user-defined function - error messages" on page 452
- "LSPep2ProbNuc user-defined function - example" on page 451

LSPep2ProbNuc user-defined function - example

The following example shows how you can transform a peptide sequence into a nucleotide sequence using the most probable translations defined in the `yeast_high.cod` frequency table.

```
values db2ls.LSPep2ProbNuc('RDNNDDDN', '\data\yeast_high.cod')
```

The result of the above values statement is:

```
AGAGACAATAACGACGATGATAAC
```

A second execution of the same statement produces the following string:

AGA**GAT**AATAACGACGAT**GAC**AAC

A third execution of the same statement produces the following string with random values:

AGAGAT**AAC**AACGAC**GACGATAAT**

Codons in bold codons highlight the differences between the current and previous transformations.

The results from the single values statement shows that function LSPep2ProbNuc chooses one of the possible symbols based on pervious statistics. This is different from function LSPep2AmbNuc which uses ambiguous symbols where there are more possible translations.

Function LSPep2ProbNuc picks up the most probable translations for each symbol and then replaces every symbol with a random translation from the set previously picked. Assume that you have the following data in a frequency table:

Table 99. Sample frequency table data

Amino acid	Codon	Frequency
Ala	GCG	0.17
Ala	GCA	0.13
Ala	GCT	0.17
Ala	GCC	0.53

Assume that the peptide sequence contains four “A” symbols (Ala). The function translates A twice to GCC; once to GCG and once to GCT. However, the order that the function produces the translations is random. The query could translate the first A to each of translations from the set {GCC, GCC, GCG, GCT}. The result is always two occurrences of GCC, one occurrence of GCG and one occurrence of GCT in the output DNA sequence. Multiple executions of the function on the same sequence might return DNA sequences with the values interchanged.

Related reference:
- “LSPep2ProbNuc user-defined function” on page 451
- “LSPep2ProbNuc user-defined function - error messages” on page 452
- “LSPep2AmbNuc user-defined function - example” on page 449

LSPep2ProbNuc user-defined function - error messages

Table 100. Messages issued by the LSPep2ProbNuc user-defined function

Error code	Message	Explanation
SQL0443N	Routine ″DB2LS.LSPEP2PROBNUC″ (specific name ″LSPEP2PROBNUC″) has returned an error SQLSTATE with diagnostic text ″Sequence not valid″. SQLSTATE=38608	The input sequence is invalid.

Table 100. Messages issued by the LSPep2ProbNuc user-defined function (continued)

Error code	Message	Explanation
SQL0443N	Routine ″DB2LS.LSPEP2PROBNUC″ (specific name ″LSPEP2PROBNUC″) has returned an error SQLSTATE with diagnostic text ″No translation found″. SQLSTATE=38610	The codon frequency table file is empty.
SQL0443N	Routine ″LSPEP2PROBNUC″ (specific name ″LSPEP2PROBNUC″) has returned an error SQLSTATE with diagnostic text ″Can't open the translation table file″. SQLSTATE=38612	The file does not exist.
SQL0443N	Routine ″DB2LS.LSPEP2PROBNUC″ (specific name ″LSPEP2PROBNUC″) has returned an error SQLSTATE with diagnostic text ″Line too long reading from file″. SQLSTATE=38614	The file contains lines that are longer than allowed.
SQL0443N	Routine ″DB2LS.LSPEP2PROBNUC″ (specific name ″LSPEP2PROBNUC″) has returned an error SQLSTATE with diagnostic text ″Invalid data file″. SQLSTATE=38615	The file format is invalid.
SQL0443N	Routine ″LSPEP2PROBNUC″ (specific name ″LSPEP2PROBNUC″) has returned an error SQLSTATE with diagnostic text ″Can't construct the translation table″. SQLSTATE=38611	The file contains invalid symbols.

Related reference:
- "LSPep2ProbNuc user-defined function" on page 451
- "LSPep2ProbNuc user-defined function - example" on page 451

Defline parsing user-defined functions

Defline parsing user-defined functions - overview

Defline parsing user-defined functions parse elements of a definition line. For example, the defline parsing user-defined functions:
- Enable joins with other data sources on sequence identifiers parsed out of the defline
- Evaluate predicates on portions of the defline, such as `'species = "human"'`

The defline parsing functions cover the most common defline formats. Examples include definition line elements that the BLAST wrapper returns or that are present in a FASTA format data file.

Related concepts:
- "Definition line parsing" on page 108

Related reference:
- "LSDeflineParse user-defined function — examples" on page 456
- "LSDeflineParse user-defined functions" on page 454

LSDeflineParse user-defined functions

Each LSDeflineParse function parses out the fields of the NCBI standard FASTA sequence identifier (NSID) and the description into columns in a table. Definition lines that are compound definitions are output on multiple rows, with each row containing a single component definition.

DB2LS is the schema name that you use with defline parsing user-defined functions.

The defline parsing user-defined functions are:

```
►►──DB2LS.LSDeflineParse2──(definition line)──────────────────────────►◄
```

```
►►──DB2LS.LSDeflineParse3──(definition line)──────────────────────────►◄
```

```
►►──DB2LS.LSDeflineParse2_2──(definition line)────────────────────────►◄
```

```
►►──DB2LS.LSDeflineParse2_3──(definition line)────────────────────────►◄
```

```
►►──DB2LS.LSDeflineParse3_3──(definition line)────────────────────────►◄
```

definition line
A valid string representation of a definition line in FASTA format. The string must have a data type of VARCHAR and an actual length that is no greater than 1024 bytes.

The LSDeflineParse2 user-defined function:

LSDeflineParse2 parses a defline that has a two-field NSID. The result of the function is a table with four columns:

Table 101. LSDeflineParse2 user-defined function result table column descriptions

Column name	Description
ROWID	An integer that numbers the rows returned from the function.
TAG	A VARCHAR of up to three characters that represents the NSID tag.
IDENTIFIER	A VARCHAR of up to 20 characters and represents the second identifier field in the NSID.
DESCRIPTION	A VARCHAR of up to 1019 characters.

The LSDeflineParse3 user-defined function:

LSDeflineParse3 parses a defline that has a three-field NSID. The result of the function is a table with five columns:

Table 102. LSDeflineParse3 user-defined function result table column descriptions

Column name	Description
ROWID	An integer that numbers the rows returned from the function.

Table 102. LSDeflineParse3 user-defined function result table column descriptions (continued)

Column name	Description
TAG	A VARCHAR of up to three characters that represents the NSID tag.
ACCESSION	A VARCHAR of up to 20 characters and represents the second identifier field in the NSID.
LOCUS	A VARCHAR of up to 20 characters and represents the third identifier field in the NSID.
DESCRIPTION	A VARCHAR of up to 1017 characters.

The LSDeflineParse2_2 user-defined function:

LSDeflineParse2_2 parses a defline that has a compound identifier consisting of a pair of concatenated two-field NSIDs. The result of the function is a table with six columns:

Table 103. LSDeflineParse2_2 user-defined function result table column descriptions

Column name	Description
ROWID	An integer that numbers the rows returned from the function.
TAG1	A VARCHAR of up to three characters that represents the NSID tag of the first identifier.
IDENTIFIER1	A VARCHAR of up to 20 characters and represents the second identifier field of the first NSID.
TAG2	A VARCHAR of up to three characters that represents the NSID tag of the first identifier.
IDENTIFIER2	A VARCHAR of up to 20 characters and represents the second identifier field of the second NSID.
DESCRIPTION	A VARCHAR of up to 1015 characters.

The LSDeflineParse2_3 user-defined function:

LSDeflineParse2_3 parses a defline that has a compound identifier consisting of a two-field NSID concatenated with a three-field NSID. The order of concatenation in the input defline--whether the two-field NSID comes before the three-field NSID, or vice versa--is not important. The result of the function is a table with seven columns:

Table 104. LSDeflineParse2_3 user-defined function result table column descriptions

Column name	Description
ROWID	An integer that numbers the rows returned from the function.
TAG1	VARCHAR of up to three characters that represents the NSID tag of the two-field identifier.
IDENTIFIER	A VARCHAR of up to 20 characters and represents the second identifier field of the two-field NSID.
TAG2	A VARCHAR of up to three characters that represents the NSID tag of the three-field identifier.
ACCESSION	A VARCHAR of up to 20 characters and represents the second identifier field of the three-field NSID.

Table 104. LSDeflineParse2_3 user-defined function result table column descriptions (continued)

Column name	Description
LOCUS	A VARCHAR of up to 20 characters and represents the third identifier field of the three-field NSID.
DESCRIPTION	A VARCHAR of up to 1013 characters.

The LSDeflineParse3_3 user-defined function:

LSDeflineParse3_3 parses a defline that has a compound identifier consisting of a pair of three-field NSIDs. The result of the function is a table with eight columns:

Table 105. LSDeflineParse3_3 user-defined function result table column descriptions

Column name	Description
ROWID	An integer that numbers the rows returned from the function.
TAG1	A VARCHAR of up to three characters that represents the NSID tag of the first identifier.
ACCESSION1	A VARCHAR of up to 20 characters and represents the second identifier field of the first NSID.
LOCUS1	A VARCHAR of up to 20 characters and represents the third identifier field of the first NSID.
TAG2	A VARCHAR of up to three characters that represents the NSID tag of the first identifier.
ACCESSION2	A VARCHAR of up to 20 characters and represents the second identifier field of the second NSID.
LOCUS2	A VARCHAR of up to 20 characters and represents the third identifier field of the second NSID.
DESCRIPTION	A VARCHAR of up to 1014 characters.

Related reference:
- "LSDeflineParse user-defined function — examples" on page 456

LSDeflineParse user-defined function — examples

This topic contains seven examples that show how the LSDeflineParse user-defined functions parse definition lines into result tables.

The following example query and results table shows how the LSDeflineParse2 user-defined function parses a definition line containing a two-field NSID:

```
select *
from table(DB2LS.LSDeflineParse2(
          '>gi|12346 hypothetical protein 185 —wheat chloroplast')) as t
```

The result table contains the following data:

Table 106. LSDeflineParse2 user-defined function result data

Column name	Data
ROWID	1
TAG	gi
IDENTIFIER	12346

Table 106. LSDeflineParse2 user-defined function result data (continued)

Column name	Data
DESCRIPTION	hypothetical protein 185 – wheat chloroplast

The following example query and results table shows how the LSDeflineParse3 user-defined function parses a definition line containing a three-field NSID:

```
select *
from table(DB2LS.LSDeflineParse3('
          >gb|U37104|APU37104 Aethia pusilla cytochrome b gene')) as t
```

The result table contains the following data:

Table 107. LSDeflineParse3 user-defined function result data

Column name	Data
ROWID	1
TAG	gb
ACCESSION	U37104
LOCUS	APU37104
DESCRIPTION	Aethia pusilla cytochrome b gene

The following example query and results table shows how the LSDeflineParse2_2 user-defined function parses a definition line containing a compound identifier that consists of a pair of 2-field NSIDs:

```
select *
from table(DB2LS.LSDeflineParse2_2(
          '>gb|U37104|gim|73401A Aethia pusilla cytochrome b gene')) as t
```

The result table contains the following data:

Table 108. LSDeflineParse2_2 user-defined function result data

Column name	Data
ROWID	1
TAG1	gb
IDENTIFIER1	U37104
TAG2	gim
IDENTIFIER2	73401A
DESCRIPTION	Aethia pusilla cytochrome b gene

The following example query contains a definition line with a compound identifier that consists of a 2-field NSID concatenated with a 3-field NSID. The example shows how the LSDeflineParse2_3 function parses the definition line.

```
select *
from table(DB2LS.LSDeflineParse2_3('
          >gi|12346|gp|CAA44030.1|CHTAHSRA_4
          hypothetical protein 185 – wheat chloroplast')) as t
```

The result table contains the following data:

Table 109. LSDeflineParse2_3 user-defined function result data

Column name	Data
ROWID	1
TAG1	gi
IDENTIFIER	12346
TAG2	gp
ACCESSION	CAA44030.1
LOCUS	CHTAHSRA_4
DESCRIPTION	hypothetical protein 185 – wheat chloroplast

The following example query contains a definition line with a compound identifier that consists of a 3-field NSID concatenated with a 2-field NSID. The example shows how the LSDeflineParse2_3 function parses the definition line.

```
select *
from table(DB2LS.LSDeflineParse2_3('
          >gp|CAA44030.1|CHTAHSRA_4|gi|12346
          hypothetical protein 185 - wheat chloroplast')) as t
```

The result table contains the following data:

Table 110. LSDeflineParse2_3 user-defined function result data

Column name	Data
ROWID	1
TAG1	gi
IDENTIFIER	12346
TAG2	gp
ACCESSION	CAA44030.1
LOCUS	CHTAHSRA_4
DESCRIPTION	hypothetical protein 185 – wheat chloroplast

The following example query and results table shows how the LSDeflineParse3_3 user-defined function parses a definition line that contains a compound identifier with a pair of 3-field NSIDs:

```
select * from table(DB2LS.LSDeflineParse3_3('
                  >dbj|AAD55586.1|AF055084_1|gp|CAA44030.1|CHTAHSRA_4
                  hypothetical protein 185 – wheat chloroplast')) as t
```

The result table contains the following data:

Table 111. LSDeflineParse3_3 user-defined function result data

Column name	Data
ROWID	1
TAG1	dbj
ACCESSION1	AAD55586.1
LOCUS1	AF055084_1
TAG2	gp

Table 111. LSDeflineParse3_3 user-defined function result data (continued)

Column name	Data
ACCESSION2	CAA44030.1
LOCUS2	CHTAHSRA_4
DESCRIPTION	hypothetical protein 185 – wheat chloroplast

You can use any of the defline user-defined functions to parse a compound definition line. The following example query contains a compound definition line with multiple definitions that are separated by a Control-A character. You can find this type of definition line in NCBI's non-redundant protein database nr. The example shows how the LSDeflineParse2_3 function parses the definition line.

```
select *
from table(DB2LS.LSDeflineParse2_3('
          >gi|12346|gp|CAA44030.1|CHTAHSRA_4
          hypothetical protein 185 - wheat chloroplast
     ^Agp|CAA44030.1|CHTAHSRA_4|gi|12346
          hypothetical protein 185 - wheat chloroplast')) as t
```

The result table contains the following data:

Table 112. LSDeflineParse2_3 user-defined function result data

Column name	Data	Data
ROWID	1	2
TAG1	gi	gi
IDENTIFIER	12346	12346
TAG2	gp	gp
ACCESSION	CAA44030.1	CAA44030.1
LOCUS	CHTAHSRA_4	CHTAHSRA_4
DESCRIPTION	hypothetical protein 185 – wheat chloroplast	hypothetical protein 185 – wheat chloroplast

Related reference:

- "LSDeflineParse user-defined functions" on page 454

Generalized pattern matching user-defined functions

The generalized pattern matching user-defined functions identify areas of interest in a given string, such as a nucleotide or peptide sequence.

LSPatternMatch user-defined function

```
►►—DB2LS.LSPatternMatch—(input character sequence, pattern)——————————————►◄
```

input character sequence
: The character string representation must have a data type of VARCHAR and an actual length that is no greater than 32672 bytes.

pattern
: The pattern as specified in any valid Perl regular expression. The character string representation must have a data type of VARCHAR and an actual length that is no greater than 32672 bytes.

The schema name is DB2LS.

You can use the LSPatternMatch user-defined function to search the input nucleotide or peptide sequence for a pattern you specify.

The result of the function is an integer representing the position of the first match of the pattern in the sequence. The function returns a value of zero if there is no match.

If you have patterns written with the PROSITE syntax, you can covert them to Perl syntax with the LSPrositePattern user-defined function. You can then use the converted syntax with the LSPatternMatch user-defined function.

Related reference:
- "LSPatternMatch user-defined function – example" on page 460
- "LSPrositePattern user-defined function" on page 462

LSPatternMatch user-defined function – example

In the following example, look for the beginning position of the string that matches "coward", "cowage", "cowboy", or "cowl".

```
values DB2LS.LSPatternMatch('joe the cowboy is next', 'cow(ard|age|boy|l)')
```

The function searches by characters, and in this example, returns a value of nine. The string "cowboy" begins at position nine, assuming that the first position is one.

In the next example, look for the beginning position of the string that matches "not " or "non ":

```
values DB2LS.LSPatternMatch('match not and non but
   no match for no or none', 'no[tn] ')
```

The function searches by characters, and in this example, returns a value of seven. The string "not " begins at position seven, assuming that the first position is one.

LSPatternMatch is useful in select statements to filter the results using the PERL syntax, which is a more powerful syntax than the SQL LIKE statement. In the following example, use LSPatternMatch on a blast output to filter the genes that match a certain pattern:

```
SELECT BlastOutput.*
   FROM BlastOutput
   WHERE db2ls.LSPatternMatch(HSP_H_Seq, 'F[GSTV]PRL') > 0;
```

If you are more familiar with the PROSITE syntax, you can use the LSPrositePattern function with the above query. Change the query to the following:

```
SELECT BlastOutput.*
   FROM BlastOutput
   WHERE db2ls.LSPatternMatch(HSP_H_Seq,
         db2ls.LSPrositePattern('F-[GSTV]-P-R-L.') ) > 0;
```

The pattern matching functions are useful for searching other types of text, as well as the nucleotide or peptide sequences. Consider using the SQL LIKE statement when performance might be a concern.

The following example shows a query that filters BLAST hsp alignments based on protein motifs found in the subject or target line of the alignment. The example is adapted from Zhang,Z., Schaffer,A.A., Miller,W., Madden,T.L., Lipman,D.J., Koonin,E.V. and Altschul,S.F. (1998) Protein sequence similarity searches using patterns as seeds. *Nucl. Acids Res.*, **26**, 3896-3990.

The following query returns only alignments in which the subject sequence includes the P-loop ATPase domain [GA]xxxxGK[ST]. The query uses CED4, the *Caenorhabditis elegans* regulator of cell death, as a query sequence against NCBI's non-redundant protein sequence database. The database retrieves the blast query sequence from the translation of the CDS feature of GenBank entry X69016.

```
SELECT HSP_Q_Seq, HSP_Midline, HSP_H_Seq
FROM BlastP b, GBseq gs, gbfeat gf, gbqual gq
WHERE gs.PRIMARYACCESSION = 'X69016' and
      gs.sequencekey = gf.sequencekey and
      gf.featurejoinkey = gq.featurejoinkey and
      gf.FeatureKey = 'CDS' and
      gq.QualifierName = 'translation' and
      gq.QualifierValue = b.BlastSeq and
      db2ls.LSPatternMatch(HSP_H_Seq,
      db2ls.LSPrositePattern('[GA]-x(4)-G-K-[ST].') ) > 0;
```

You can use the next example query to find HSPs in a genomic sequence that contain putative single nucleotide polymorphisms (SNPs) with respect to a canonical query sequence. It is adapted from Extending traditional query-based integration approaches for functional characterization of post-genomic data. (2001) Barbara A Eckman, Anthony S Kosky, and Leonardo A Laroco Jr.   *Bioinformatics* 17(7), 587-601.

The query uses the pattern matching on the blast hsp midline to find a pattern of ≥20 perfect matches followed by a single mismatch followed by ≥20 perfect matches. That is, 20 "|" characters, a single space, and then 20 "|" characters in the midline of the alignment.

This example also shows the usage of the LSPatternMatch user-defined function on strings that are not nucleotide or peptide sequences.

```
SELECT HSP_Info, HSP_Midline, HSP_H_Seq
   FROM BlastOutput
   WHERE db2ls.LSPatternMatch(HSP_Midline, '\|{20} \|{20}') > 0;
```

You can rewrite the previous query as:

```
SELECT HSP_Info, HSP_Midline, HSP_H_Seq, func.Position, func.Match
FROM BlastOutput,
   TABLE(SELECT * AS c FROM TABLE(
   LSMultiMatch(HSP_Midline, '\|{20} \|{20}') )
   AS f) AS func
```

This second query will return the blast rows that have a match together with the matched string and their position in the sequence.

BlastOutput is a view over a BlastN nickname.

Related reference:

- "LSPrositePattern user-defined function - example" on page 462
- "LSPatternMatch user-defined function" on page 459
- "LSPrositePattern user-defined function" on page 462

LSPrositePattern user-defined function

```
►►—DB2LS.LSPrositePattern—(pattern)——————————————————————————————►◄
```

pattern
The pattern matching syntax specified by the Prosite syntax. The character string representation must have a data type of VARCHAR and an actual length that is no greater than 32672 bytes.

The schema name is DB2LS.

Use the LSPrositePattern user-defined function to convert from the PROSITE syntax to the PERL syntax. You can then use the converted syntax with the LSPatternMatch, LSMultiMatch, and LSMultiMatch3 user-defined functions.

The result of the function is a character string representing a regular expression in the Perl syntax. The character string representation must have a data type of VARCHAR and an actual length that is no greater than 32672 bytes.

Related reference:
- "LSPrositePattern user-defined function - example" on page 462
- "LSPatternMatch user-defined function" on page 459

LSPrositePattern user-defined function - example

In the following example, convert a pattern from PROSITE syntax into PERL syntax.

```
values db2ls.LSPrositePattern('[AC]-x-V-x(4)-{ED}.');
```

The function converts the input pattern in PROSITE syntax into an equivalent pattern in Perl syntax, as shown in the following example:

```
[AC].V.{4}[^ED]
```

The next example converts another syntax pattern from PROSITE into the PERL syntax:

```
values db2ls.LSPrositePattern('<A-x-[ST](2)-x(0,1)-V.');
```

The function translate the string from the PROSITE syntax based on the input pattern and returns the following:

```
\AA.[ST]{2}.{0,1}V
```

The next example converts the pattern corresponding to the PROSITE database entry with the ID number of PS01205 into a PERL pattern that is used as input by the pattern matching functions.

```
values db2ls.LSPrositePattern('R-P-L-[IV]-x-[NS]-F-G-S-[CA]-T-C-P-x-F.')
```

The result of this query is:

```
RPL[IV].[NS]FGS[CA]TCP.F
```

The next example shows how you can use the function in a query. The query prints out only sequences that match the PROSITE pattern specified.

```
SELECT H_Accession, HSP_Info, HSP_H_Seq
FROM BlastOutput
WHERE db2ls.LSPatternMatch( HSP_H_Seq,
  db2ls.LSPrositePattern('R-P-L-[IV]-x-[NS]-F-G-S-[CA]-T-C-P-x-F.') ) > 0;
```

The next example converts the pattern corresponding to the PROSITE entry whose ID is PS00261:

```
values db2ls.LSPrositePattern('C-[STAGM]-G-[HFYL]-C-x-[ST].')
```

The result of this query is:

```
C[STAGM]G[HFYL]C.[ST]
```

Related reference:

- "LSPatternMatch user-defined function – example" on page 460
- "LSPrositePattern user-defined function" on page 462

Regular expression support

Regular expression support is provided by the PCRE library package, which is open source software, written by Philip Hazel, and copyright by the University of Cambridge, England.

The source can be found at ftp://ftp.csx.cam.ac.uk/pub/software/programming/pcre/.

Related concepts:

- "Life sciences user-defined functions - overview" on page 443

GeneWise user-defined functions

The GeneWise user-defined function aligns a protein sequence with a genomic sequence.

GeneWise is a commonly-used component which aligns a protein sequence with a genomic DNA sequence, allowing for introns and frameshifting errors.

Linking to GeneWise

This topic describes the procedure for linking to the GeneWise library.

Procedure:

To link to the GeneWise library:

| 1. Download the Wise2 package version 2.1.20c from www.ebi.ac.uk/Wise2.
2. Expand the archive into a folder of your preference.
3. Compile the package with pthread support. For more information on this step, see the Wise2 documentation.

| On HP-UX federated servers, you need to add the option +z to the compiler
| options before you compile the source code. To add this option, open the file
| makefile from the root directory of the Wise2 package and change the line as
| shown in this example:

```
| CFLAGS = -c -O -DPTHREAD to: CFLAGS = -c -O -DPTHREAD  +z
```

4. Run **make api** in its root directory.

5. Set the WISE2_HOME environment variable to point to the Wise2 package root directory.
6. Set the WISECONFIGDIR variable in the `sqllib/cfg/db2dj.ini` file to point to the wisecfg subdirectory. For example, if Wise2 package is installed in `/usr/wise2.1.20c/`, then add `WISECONFIGDIR=/usr/wise2.1.20c/wisecfg/` into the `db2dj.ini` file.
7. Run the **djxlinkLSGeneWise** script which is located in the `sqllib/bin` directory.
8. Check the output from the **djxlinkLSGeneWise** script. This output file, `djxlinkLSGeneWise.out`, is located in the `sqllib/function` directory.
9. If no errors were reported, then the library was successfully built.

Related reference:
- "LSGeneWise user-defined function" on page 464

LSGeneWise user-defined function

```
►►──DB2LS.LSGeneWise──(protein sequence, DNA_sequence)──────────────────────►◄
```

protein sequence
A valid character string representation describing a peptide sequence. The character string representation must have a data type of VARCHAR and an actual length that is no greater than 32672 bytes.

DNA_sequence
A valid character string representation describing a nucleotide sequence. The character string representation must have a data type of VARCHAR and an actual length that is no greater than 32672 bytes.

Table 113 shows the one row output table the LSGeneWise function returns.

Table 113. Column names, types, and descriptions for the output table from the LSGeneWise function

Column name	Type	Description
PROTEIN_OFFSET	INTEGER	Represents the starting offset in the input protein sequence at which an alignment was found.
DNA_OFFSET	INTEGER	Represents the starting offset in the input DNA sequence at which an alignment was found.
PROTEIN	VARCHAR(32672)	A fragment from the input sequence representing the aligned sequence.
SIMILARITY	VARCHAR(32672)	Shows the matching between the protein and the dna sequences. Perfect matches are marked with the corresponding symbol letter. Non-perfect matches with a positive score are indicated with the "+" sign, and mismatches are indicated with a space.
TRANSLATED_DNA	VARCHAR(32672)	The translated DNA sequence. The sequence might contain dashes and special symbols like deletions and introns.
DNA	VARCHAR(32672)	The DNA sequence with special markers like frame-shifting and introns.

The correspondence between the output of the GeneWise program and the output of the LSGeneWise UDF is the following:

- protein and dna offsets printed by the GeneWise program match PROTEIN_OFFSET and DNA_OFFSET columns.
- protein sequence printed on the first line by GeneWise matches PROTEIN column.
- similarity line, the second line in GeneWise output matches SIMILARITY column.
- the third line in the GeneWise output matches TRANSLATED_DNA column.
- the fourth, fifth and sixth lines of the GeneWise output are combined, by reading them vertically, into the DNA column.

Use the LSGeneWise user-defined function to align a protein sequence with a genomic DNA sequence, allowing for introns and frameshifting errors.

For more information on the LSGeneWise user-defined function output, see http://www.ebi.ac.uk/Wise2.

Related tasks:

- "Linking to GeneWise" on page 463

Related reference:

- "LSGeneWise user-defined function – example" on page 465

LSGeneWise user-defined function – example

The following example shows a query using the LSGeneWise user-defined function and the resulting data.

```
select protein_offset, dna_offset, protein, similarity, translated_dna, dna
from table( db2ls.LSGeneWise( '
             VEPKRAVPRQDIDSPNAGATVKKLFVGALKDDHDEQSIRDYFQHFGNIVDINIVIDKETGK
             KRGFAFVEFDDYDPVDKVVLQKQHQLNGKMVDVKKALPKQNDQQGGGGGRGGPGGRAGGNR
             GNMGGGNYGNQNGGGNWNNGGNNWGNNR',
            'CACTTAACTGTGAAAGATATTTGTTGGTGGCATTAAAGAAGACACTGAAGAACATCACCTAAG
             AGATTATTTTGAACAGTATGGAAAAATTGAAGTGATTGAAATCATGACTGACCGAGGCAGTGG
             CAAGAAAAGGGGCTTTGCCTTRGTAACCTTTGACGACCATGACTCCGTGGATAAGATTGTCAT
             TCAGAAATACCATACTGTGAATGGCCACAACTGTGAAGTTAGAAAAGCCCTGTCAAAGCAAGA
             GATGGCTAGTGCTTCATCCAGCCAAAGAGGTCGAAGTGGTTCTGGAAACTTTGGTGGTGGTCG
             TGGAGGTGGTTTCGGTGGGAATGACAACTTCGGTCGTGGAGGAAACTTCAGTGGTCGTGGTYG
             CTTTGGTGGCAGCCGTGGTGGTGGTGGATATGGTGGC' ) ) as f;
```

Table 114. Results table

Column	Data
PROTEIN_OFFSET	23
DNA_OFFSET	14
PROTEIN	KLFVGALKDDHDEQSIRDYFQHFGNIVDINIVIDKET GKKRGFAFVEFDDYDPVDKVVLQKQHQLNGKMVD VKKALPKQNDQQGGGGGRGGPGGRAGGNRGNMGG GNYGNQNGGGNWNNGGN
SIMILARITY	K+FVG +K+D +E +RDYF+ +G I I I+ D+ +GKKRGFA+V FDD+D VDK+V+QK H +NG +V+KAL KQ RG G GN+GGG G G N+ GGN

Table 114. Results table (continued)

Column	Data
TRANSLATED_DNA	KIFVGGIKEDTEEHHLRDYFEQYGKIEVIEIMTDRGSGK KRGFAxVTFDDHDSVDKIVIQKYHTVNGHNCEVRKAL SKQEMASASSSQRGRSGS------ GNFGGGRGGGFGGNDNFGRGGN
DNA	aagatatttgttggtggcattaaagaagacactgaagaacatcacctaagagat...

Related tasks:

- "Linking to GeneWise" on page 463

Related reference:

- "LSGeneWise user-defined function" on page 464

Motifs user-defined functions

Motif user-defined functions match patterns in nucleotide or amino acid sequences.

LSBarCode user-defined function

```
►►──DB2LS.LSBarCode──(input string sequence)──────────────────────────────►◄
```

input string sequence
A valid character string representing an HSP alignment between two sequence fragments. The character string representation must have a data type of VARCHAR and an actual length that is no greater than 32672 bytes.

The schema name is DB2LS.

Use the LSBarCode user-defined function to use a sequence as input and generate another sequence by replacing every character except spaces and plus signs with the vertical bar symbol (|).

The result of the function is a variable character sequence representing a barcode sequence.

Related reference:

- "LSBarCode user-defined function — example" on page 466
- "LSMultiMatch user-defined function" on page 468
- "LSMultiMatch3 user-defined function" on page 469

LSBarCode user-defined function — example

This example creates a barcode from a string sequence:

```
values db2ls.LSBarCode(
   'MDY +G++L  GN  ++ +PASLTK+MT YVV +A+ + +I   D+VTVG+DAWA  NP ')
```

The result of the above values statement is:

```
||| +|++|  ||  ++ +||||||+|| ||| +|+ + +|   |+||||+||||  ||
```

The next example shows a more realistic use of this function. Suppose that a researcher running a BLAST search wants to return only HSP alignments that contain fewer than 25% prolines among their perfect matches. This example uses the function to compute the percentage of prolines (symbol 'P') among the perfect matches in an alignment returned by BLAST. Notice that this example also invokes the LSMultiMatch3 user-defined function. The query uses match function to find perfect matches. It is used in conjunction with the LSBarCode function in this query because Blast does not always return a sequence of bars ("|") in an alignment. The following example shows this:

```
Query:       MDYTTGQILTAGNEHQQRNPASLTKLMTGYVVDRAIDSHRITPDDIVTVGRDAWAKDNPV
Alignment:   MDY +G++L  GN  ++ +PASLTK+MT YVV +A+ + +I   D+VTVG+DAWA  NP
Target:      MDYASGKVLAEGNADEKLDPASLTKIMTSYVVGQALKADKIKLTDMVTVGKDAWATGNPA
```

To ensure that the output is aligned with the correct sequence of bars, use the LSBarCode function. The function replaces all characters except spaces and plus signs with a vertical bar.

```
SELECT BlastOutput.* , float( p )/ float( m ) AS percent_prolines
  FROM
  BlastOutput b,
  table(SELECT COUNT(*) AS p FROM table(
          db2ls.LSMultiMatch3(
            b.HSP_Q_Seq, 'P',
            db2ls.LSBarCode(b.HSP_Midline), '\|',
            b.HSP_H_Seq, 'P')
          ) AS f
        ) AS y,
  table(SELECT COUNT(*) AS m FROM table(
          db2ls.LSMultiMatch3(
            b.HSP_Q_Seq, '.',
            db2ls.LSBarCode(b.HSP_Midline), '\|',
            b.HSP_H_Seq, '.')
          ) AS f
        ) AS z
WHERE float(p) / float(m) < 0.25;
```

In this query, BlastOutput is actually a view over a Blast nickname. The query uses the LSMultiMatch3 function to return the perfect matches on alignment. The first usage returns the perfect matches for symbol "P", the second one returns all the perfect matches. A row from the result table show in Table 115.

Table 115. Sample results row

HSP_Q_SEQ	HSP_H_SEQ	HSP_INFO	PERCENT_PROLINES
NIWDFMQGN...	NIWDFMQGN...	Identities = 80/80 (100%), Positives = 80/80 (100%), Gaps = 0/80 (0%)	+2.50000000000000E-002

The previous query was adapted from `Extending traditional query-based integration approaches for functional characterization of post-genomic data. (2001) Barbara A Eckman, Anthony S Kosky and Leonardo A Laroco Jr.` *`Bioinformatics`* `17(7), 587-601.`

Related reference:

- "LSMultiMatch3 user-defined function – example" on page 470
- "LSBarCode user-defined function" on page 466

LSMultiMatch user-defined function

```
►►──DB2LS.LSMultiMatch──(input nucleotide or peptide sequence, pattern)──────────►◄
```

input nucleotide or peptide sequence
: A valid character string representation describing a nucleotide or peptide sequence. The character string representation must have a data type of VARCHAR and an actual length that is no greater than 32672 bytes.

pattern
: The pattern matching grammar specified by the Perl language. The character string representation must have a data type of VARCHAR and an actual length that is no greater than 32672 bytes.

The schema name is DB2LS.

Use the LSMultiMatch user-defined function to return a table for each match that does not overlap in the input sequence. Each table consists of a start position and the matching sequence fragment.

The result of the function is a table with two columns. The first column is an integer representing the start position of a match of the pattern in the sequence. The second column is the matching sequence fragment.

Related reference:
- "LSMultiMatch user-defined function - example" on page 468
- "LSBarCode user-defined function" on page 466
- "LSMultiMatch3 user-defined function" on page 469

LSMultiMatch user-defined function - example

This example looks for the position and the matching fragments for all the non-overlapping matches taken from the input.

```
SELECT position, match FROM table
  (LSMultiMatch('match not and non but no match for no or none',
      'no[tn] ')) as f
```

The query returns a table that is based on this select statement that shows the results of the matches:

Table 116. Result of LSMultiMatch returning multiple rows

POSITION	MATCH
7	not
15	non

LSMultiMatch returns the position and the matched string for all matches. The following example searches Entrez Nucleotide for sequence entries that contain a certain motif. The query prints the sequence identifiers and the matched sequences. The sub-patterns ".{0,9}" at the beginning and at the end have to match up to nine characters before and after the sequence. The query also prints these characters.

```
select SequenceKey, Position, Match from GBSeq,
  table(db2ls.LSMultiMatch(Sequence, '.{0,9}(ATG|CGC)ACGGGC.{0,9}') )
  as fmatch
  WHERE entrez.contains(KeywordList,
        'Na/K/2Cl cotransporter AND nkcc1 gene') = 1;
```

The result of this query is as follows:

Table 117. Search Entrez data

SEQUENCEKEY	POSITION	MATCH
N02B59AE0.04DD4E84	1	TGCTTGGTGATGACGGGCTACCCCAAC
N02B59AE0.04DD4E84	91	GGCCATGTTCGCACGGGCTCCAGAAGG
N02B59AE0.04DC5EF4	1	TGCTTGGTGATGACGGGCTACCCCAAC
N02B59AE0.04DC5EF4	91	GGCCATGTTCGCACGGGCTCCAGAAGG

Related reference:
- "LSMultiMatch user-defined function" on page 468
- "LSBarCode user-defined function" on page 466
- "LSMultiMatch3 user-defined function" on page 469

LSMultiMatch3 user-defined function

```
►►──DB2LS.LSMultiMatch3──(input string1, pattern1, input string2, pattern2, input string3, pattern3)──────────►◄
```

input strings
: A valid character string representation describing a nucleotide or peptide sequence, or an HSP_Midline string from a blast alignment. The character string representation must have a data type of VARCHAR and an actual length that is no greater than 32672 bytes.

pattern
: The pattern matching grammar specified by the Perl language. The character string representation must have a data type of VARCHAR and an actual length that is no greater than 32672 bytes.

The schema name is DB2LS.

Use the LSMultiMatch3 user-defined function to input three patterns and three strings and return any positions where all three strings match their respective patterns. You can use this user-defined function to perform a pattern match on an alignment.

The result of the function is a table with four columns. The first column is an integer representing the start position of a match of the pattern in all the sequences. The function anchors all the strings together at the first position. The second, third, and fourth columns are the matching sequence fragments.

Related reference:
- "LSMultiMatch3 user-defined function – example" on page 470
- "LSMultiMatch user-defined function" on page 468
- "LSBarCode user-defined function" on page 466

LSMultiMatch3 user-defined function – example

The following example uses the function to compute the percentage of a particular amino-acid symbol among the perfect matches returned by Blast. Notice that this example also invokes the LSBarCode user-defined function. The query needs this because Blast does not always return a sequence of bars ("|") in an alignment. The following example illustrates this:

```
Query:       MDYTTGQILTAGNEHQQRNPASLTKLMTGYVVDRAIDSHRITPDDIVTVGRDAWAKDNPV
Alignment:   MDY +G++L  GN  ++ +PASLTK+MT YVV +A+ + +I   D+VTVG+DAWA  NP
Target:      MDYASGKVLAEGNADEKLDPASLTKIMTSYVVGQALKADKIKLTDMVTVGKDAWATGNPA
```

To ensure that the output is aligned with the correct sequence of bars, use the LSBarCode function to convert the sequence. The function replaces all non-space and non-"+" characters with a vertical bar.

```
SELECT BlastOutput.* , float( p )/ float( m ) AS percent_prolines
  FROM
  BlastOutput b,
  table(SELECT COUNT(*) AS p FROM table(
          db2ls.LSMultiMatch3(
            b.HSP_Q_Seq, 'P',
            db2ls.LSBarCode(b.HSP_Midline), '\|',
            b.HSP_H_Seq, 'P')
          ) AS f
        ) AS y,
  table(SELECT COUNT(*) AS m FROM table(
          db2ls.LSMultiMatch3(
            b.HSP_Q_Seq, '.',
            db2ls.LSBarCode(b.HSP_Midline), '\|',
            b.HSP_H_Seq, '.')
          ) AS f
        ) AS z
WHERE float(p) / float(m) < 0.25;
```

In this query, BlastOutput is a view over a Blast select. The query uses the LSMultiMatch3 function to return the perfect matches on alignment. The first usage returns the perfect matches for symbol "P", the second one returns all the perfect matches. A row from the result table show in Table 118.

Table 118. Sample results row

HSP_Q_SEQ	HSP_H_SEQ	HSP_INFO	PERCENT_PROLINES
NIWDFMQG...	NIWDFMQG...	Identities = 80/80 (100%), Positives = 80/80 (100%), Gaps = 0/80 (0%)	+2.50000000000000E-002

The previous query was adapted from `Extending traditional query-based integration approaches for functional characterization of post-genomic data. (2001)  Barbara A Eckman, Anthony S Kosky and Leonardo A Laroco Jr.` *`Bioinformatics`* `17(7), 587-601.`

The following example looks for three separate patterns in three separate string fragments:

```
SELECT position, match_1, match_2, match_3
  FROM table(db2ls.LSMultiMatch3('zaza', 'a', 'abab',
             'b', 'bcbc', 'c')) as f
```

It returns the positions and the matching strings for all of the matches, as shown in the following table:

Table 119. Result of a multi-match using three inputs

POSITION	MATCH_1	MATCH_2	MATCH_3
2	a	b	c
4	a	b	c

The next example finds three separate patterns within three separate string fragments:

```
SELECT position, match_1, match_2, match_3
  FROM table
  (LSMultiMatch3('cbccbbcccbbbccccbbbbccccc','c{1,3}b{1,3}c{1,3}',
   'abcdefghijklmnopqrstuvwxyzabcdefghijklmnopqrstuvwxyz',
   '.','0123456789012345678901234567890123456789','\d')) as f
```

The results are in the following table:

Table 120. Result of a multi-match using three inputs

POSITION	MATCH_1	MATCH_2	MATCH_3
1	cbcc	a	0
7	cccbbbccc	g	6

Related reference:

- “LSBarCode user-defined function — example” on page 466
- “LSBarCode user-defined function” on page 466
- “LSMultiMatch3 user-defined function” on page 469

Reverse user-defined functions

Reverse user-defined functions reverse a nucleotide or amino acid sequence.

LSRevComp user-defined function

```
►►──DB2LS.LSRevComp──(input nucleotide sequence)──────────────────────────►◄
```

input nucleotide sequence
: A valid character string representation describing a nucleotide sequence. The sequence can contain IUPAC ambiguity codes. A character string representation must have a data type of VARCHAR and an actual length that is no greater than 32672 bytes.

The schema name is DB2LS.

The result of the function is a character string with a data type of VARCHAR and an actual length that is not greater than 32672 bytes representing the reverse complement of the nucleotide sequence.

Related reference:

- “LSRevComp user-defined function—example” on page 472
- “LSRevNuc user-defined function” on page 473

- “LSRevPep user-defined function” on page 474

LSRevComp user-defined function—example

You can use the LSRevComp function in an SQL statement wherever you would use any built-in function that accepts a nucleotide sequence. For example:

```
SELECT DB2LS.LSRevComp(:NucSeq) FROM SYSDUMMY1;
```

This example uses the function to return the reverse complement of the input sequence that comes from a host variable.

If you use an invalid string, or invalid data type, you get the following error message:

```
SQL0443N Routine "DB2LS.LSREVCOMP" (specific name "LSREVCOMP") has returned
an error SQLSTATE with diagnostic text "Sequence not valid". SQLSTATE=38608
```

An exception is raised if the input alphabet is not correct.

The following example shows how the LSRevComp user-defined function works in a query:

```
SELECT HSP_H_Seq, db2ls.LSRevComp(HSP_H_Seq) as REV_HSP_H_Seq
FROM BlastN
WHERE BlastSeq='ccgctagtattggtcaatcttttgatatccaccgaa'
```

The results of the query are shown below:

```
HSP_H_SEQ                      REV_HSP_H_SEQ
------------------------------ ----------------------------

AGTATTGGTCAATCTTTTGAT          ATCAAAAGATTGACCAATACT

TGGTCAATCTTTTGATA              TATCAAAAGATTGACCA

TTGGCCAATCTTTTGATATCC          GGATATCAAAAGATTGGCCAA

TCAATCTTTTGATATCC              GGATATCAAAAGATTGA

GGATATCAAAAGATTGA              TCAATCTTTTGATATCC


  5 record(s) selected.
```

You can use the reverse function along with other life sciences user-defined functions to translate the reverse complement of a nucleotide sequence, as in the following example:

```
values db2ls.LSNuc2Pep(
       db2ls.LSRevComp('TTTTTCTTATTGTCTTCCTCATCGTATTTCTTATGTTGCTGATGT'))
```

The query returns the following:

```
TSAT*EIR*GRQ*EK
```

Related reference:

- “LSRevComp user-defined function” on page 471

LSRevNuc user-defined function

```
►►—DB2LS.LSRevNuc—(input nucleotide sequence)————————————————————————————►◄
```

input nucleotide sequence
A valid character string representation describing a nucleotide sequence. A character string representation must have a data type of VARCHAR and an actual length that is no greater than 32672 bytes. The nucleotide sequence must be part or all of the DNA alphabet.

The schema name is DB2LS.

The result of the function is a character string with a data type of VARCHAR and an actual length that is not greater than 32672 bytes representing the reverse order of the nucleotide sequence.

Related reference:
- "LSRevNuc user-defined function - example" on page 473
- "LSRevComp user-defined function" on page 471
- "LSRevPep user-defined function" on page 474

LSRevNuc user-defined function - example

You can use the LSRevNuc function in an SQL statement wherever you would use any built-in function that accepts a nucleotide sequence. For example:

```
SELECT DB2LS.LSRevNuc(:NucSeq) FROM SYSDUMMY1;
```

This example uses the function to reverse input data that comes from a host variable.

If you use an invalid string, or invalid data type, you get the following error message:

```
SQL0443N Routine "DB2LS.LSREVNUC" (specific name "LSREVNUC") has returned
an error SQLSTATE with diagnostic text "Sequence not valid". SQLSTATE=38608
```

The following example shows the use of the LSRevNuc user-defined function in a query.

```
SELECT HSP_H_Seq, db2ls.LSRevNuc(HSP_H_Seq) as REV_HSP_H_Seq
FROM BlastN
WHERE BlastSeq='gtaatacgtagggggctagcgcgggcaaactgaagataaagc'
```

The following results table shows the reversed nucleotide sequences the query returns:

```
HSP_H_SEQ                                          REV_HSP_H_SEQ

-------------------------------------------------- -------------------------
CGCGGGCAAACTGAAGATAAAGC                            CGAAATAGAAGTCAAACGGGCGC

GCGCTAGCCCCCTACGTATTAC                             CATTATGCATCCCCCGATCGCG

GTAATACGTAGGGGGCTAGCG                              GCGATCGGGGGATGCATAATG

GTAATACGTAGGGGGCTAGCG                              GCGATCGGGGGATGCATAATG

GTAATACGTAGGGGGCTAGCG                              GCGATCGGGGGATGCATAATG
```

```
  5 record(s) selected.
```

Related reference:
- "LSRevNuc user-defined function" on page 473

LSRevPep user-defined function

```
►►──DB2LS.LSRevPep──(input peptide sequence)──────────────────────────────►◄
```

input peptide sequence
: A valid character string representation describing a peptide sequence. A character string representation must have a data type of VARCHAR and an actual length that is no greater than 32672 bytes. The input sequence must be part of the protein alphabet.

The schema name is DB2LS.

The result of the function is a character string with a data type of VARCHAR and an actual length that is not greater than 32672 bytes representing the reverse order of the peptide sequence.

Related reference:
- "LSRevPep user-defined function - example" on page 474
- "LSRevComp user-defined function" on page 471
- "LSRevNuc user-defined function" on page 473

LSRevPep user-defined function - example

You can use the LSRevPep function in an SQL statement wherever you would use any built-in function that accepts a peptide sequence. For example:

```
SELECT DB2LS.LSRevPep(:NucSeq) FROM SYSDUMMY1;
```

This example uses the function to reverse input data that comes from a host variable.

If you use an invalid string, or invalid data type, you get the following error message:

```
SQL0443N Routine "DB2LS.LSREVPEP" (specific name "LSREVPEP") has returned
an error SQLSTATE with diagnostic text "Sequence not valid". SQLSTATE=38608
```

The following example shows how the LSRevPep user-defined function is used in a query:

```
SELECT  HSP_H_Seq, db2ls.LSRevPep(HSP_H_Seq) as REV_HSP_H_Seq
FROM BlastP
WHERE BlastSeq='MLCEIECRALSTAHTRLIHDFEPRDALTYLEGKNIFTEDH'
```

The following table shows the reversed peptide sequences the query returns.

```
HSP_H_SEQ                           REV_HSP_H_SEQ
----------------------------------- -----------------------------------

MLCEIECRALSTAHTRLIHDFEPRDALTYL...   HDETFINKGELYTLADRPEFDHILRTHATS...
```

```
RVVSTEHTRLVTDAYPEFSISFTATKN        NKTATFSISFEPYADTVLRTHETSVVR

STAHIRVLRDMVPGDEITCFYGSEFF         FFESGYFCTIEDGPVMDRLVRIHATS

AHTRRCPDHEPRGVITYL                 LYTIVGRPEHDPCRRTHA


  4 record(s) selected.
```

Related reference:
- "LSRevPep user-defined function" on page 474

Translate

The translation user-defined functions convert a nucleotide sequence into a peptide sequence.

LSNuc2Pep user-defined function

```
►►──DB2LS.LSNuc2Pep──(input nucleotide sequence──┬──────────────────────────────────────────┬──)──────────►◄
                                                 └─,filepath to external translation table──┘
```

input nucleotide sequence
: A valid character string representation describing a nucleotide sequence. A character string representation must have a data type of VARCHAR and an actual length that is no greater than 32672 bytes.

filepath to external translation table
: If you use a customized translation table, include the file path information to find the translation table. The string value of the path must be no greater than 255 characters.

The schema name is DB2LS.

The result of the function is a character string with a data type of VARCHAR and an actual length that is not greater than 10890 bytes representing the peptide sequence.

The input is a nucleotide sequence using the IUB character set. The functions assume that the first codon begins at the first character of the nucleotide sequence. If the first codon does not begin at the first character of the nucleotide sequence, use a SUBSTR function on the input sequence.

The result of the function is a peptide sequence, using the standard amino acid symbols.

The function:
- Excises spaces in input sequences.
- Ignores extraneous nucleotides outside of a reading frame.
- Returns null output if you input a null nucleotide sequence.

Related reference:
- "LSNuc2Pep user-defined function – example" on page 476
- "LSTransAllFrames user-defined function" on page 477

LSNuc2Pep user-defined function – example

Assume that you want to translate your nucleotide sequence data into a peptide sequence. This example assumes that the first codon begins at the first character of the nucleotide sequence.

You can invoke the function with a values statement. The single input is a nucleotide sequence, as in the following example:

```
values db2ls.LSNuc2Pep('TTTTTCTTATTGTCTTCCTCATCGTATTTCTTATGTTGCTGATGT')
```

The result of the above statement is a peptide sequence using the standard amino acid symbols:

```
FFLLSSSSYFLCC*C
```

If you want the translation in the +2 reading frame, then use the following example:

```
values LSNuc2Pep(SUBSTR('TTTTTCTTATTGTCTTCCTCATCGTATTTCTTATGTTGCTGATGT',2))
```

The integer in the statement indicates the starting position of the search for the codon.

Here is an example of using this function as a predicate in a query.

```
SELECT  *
  FROM proteindata
  WHERE peptideseq=DB2LS.LSNuc2Pep('TTTTTCTTATTGTCTTCCTCATCG
                                   TATTTCTTATGTTGCTGATGT');
```

The result is shown in Table 121.

Table 121. Results using the LSNuc2Pep function as a predicate

ID	PROTEINNAME	PEPTIDESEQ
1	proteinA	FSYCLPHRISYVAD

The following example translates a nucleotide sequence into a peptide sequence using an external translation table. The first parameter is the nucleotide sequence, and the second parameter is the path to the external translation table.

```
values db2ls.LSNuc2Pep('TTTTCTTATTGTCTTCCTCATCGTATTTCTTATGTTGCTGATGT',
                       'C:\translation.txt')
```

The result of the above statement using this particular translation table is the following string:

```
FSYCLPHRISYVAD
```

The following example combines two of the user-defined functions to demonstrate the additional uses of the functions:

```
values DB2LS.LSNuc2Pep(DB2LS.LSRevCompNuc('TTT..')
```

Notice that the previous example returns the same result as the following query:

```
select * from table (DB2LS.LSTransAllFrames ('TTT..')) as t
where t.readframe = -1
```

Related reference:

- "LSRevNuc user-defined function - example" on page 473
- "LSTransAllFrames user-defined function - example" on page 477

- “LSNuc2Pep user-defined function” on page 475

LSTransAllFrames user-defined function

```
►►──DB2LS.LSTransAllFrames──(input nucleotide sequence──┬────────────────────────────────────┬──)──────────►◄
                                                         └─,filepath to external translation table─┘
```

input nucleotide sequence
A valid character string representation describing a nucleotide sequence. The input sequence can contain IUPAC ambiguity codes. A character string representation must have a data type of VARCHAR and an actual length that is no greater than 32672 bytes.

filepath to external translation table
If you use a customized translation table, include the file path information to find the translation table. The string value of the path must be no greater than 255 characters.

The schema name is DB2LS.

Use the LSTransAllFrames user-defined function to produce a set of peptide sequences from a given nucleotide sequence. These peptide sequences represent possible translations of the input nucleotide sequence, in each of 6 frames. This function is useful when the input contains errors or the reading frame is not known.

The result of the function is a table with two columns. The first column is labelled READFRAME and represents the frame that is used for the translation. This column has an integer value that represents the start position of translation. A negative integer indicates a translation of the opposite strand. The second column, called PEPTIDE, is a character string with a data type of VARCHAR and an actual length that is not greater than 10890 bytes representing the peptide sequence.

The function:
- Excises spaces in input sequences.
- Ignores extraneous nucleotides outside of a reading frame.
- Returns null output if you input a null nucleotide sequence.

Related reference:
- “LSTransAllFrames user-defined function - example” on page 477
- “LSNuc2Pep user-defined function” on page 475

LSTransAllFrames user-defined function - example

Assume that you want to translate a nucleotide sequence in all six reading frames using the built-in translation table. The following example shows how to do this:

```
SELECT * FROM table(DB2LS.LSTransAllFrames('TTTTTCTTATTGTCTTCCTCATCG
                                            TATTTCTTATGTTGCTGATGT')) as t;
```

The query returns the peptides a table, as in the following example:

Table 122. Result of translating a nucleotide sequence

READFRAME	PEPTIDE
1	FFLLSSSSYFLCC*C

Table 122. Result of translating a nucleotide sequence (continued)

READFRAME	PEPTIDE
2	FSYCLPHRISYVAD
3	FLIVFLIVFLMLLM
–1	TSAT*EIR*GRQ*EK
–2	HQQHKKYDEEDNKK
–3	ISNIRNTMRKTIRK

The next example uses a customized translation table to translate a nucleotide sequence in all six reading frames.

```
SELECT * FROM table
  (DB2LS.LSTransAllFrames
   ('TTTTTCTTATTGTCTTCCTCATCGTATTTCTTATGTTGCTGATGT',
    'C:\msvs6\MyProjects\alin_udf\test\files\translation.txt')) as t;
```

The resulting table is the same as the previous example because the input sequence is the same and translation table is the same as the one built into the function.

The following example combines two of the user-defined functions to demonstrate the additional uses of the functions:

```
values DB2LS.LSNuc2Pep(DB2LS.LSRevCompNuc('TTT..')
```

Notice that the previous example returns the same result as the following query:

```
select * from table (DB2LS.LSTransAllFrames ('TTT..')) as t
where t.readframe = -1
```

The following example selects a specific reading frame from the output produced by the LSTransAllFrames function.

```
SELECT * FROM
  TABLE(db2ls.LSTransAllFrames('TTTTTCTTATTGTCTTCCTCATCG
                               TATTTCTTATGTTGCTGATGT')) AS t
  WHERE t.readframe=-2
```

The result of this query is:

Table 123. Readframe function usage

READFRAME	PEPTIDE
–2	HQQHKKYDEEDNKK

Related reference:
- "LSNuc2Pep user-defined function – example" on page 476
- "LSRevNuc user-defined function - example" on page 473
- "LSTransAllFrames user-defined function" on page 477

Codon frequency table format

A codon frequency table shows the frequency to which the amino acids are back translated into a particular codon. The LSPep2ProbNuc user-defined function uses the codon frequency table to determine a nucleotide sequence from a given peptide sequence.

The following list describes the format of the codon frequency table file:

- Two adjacent periods mark the beginning of the table. Any text that comes before is commentary. The two adjacent periods are required even if there is no commentary before them.
- The table contains the following columns:
  1. Am-Acid: a three letter code for the amino acid symbol.
  2. Codon: the codon for that amino acid symbol.
  3. Number: the number of occurrences of that codon in the genes from which the table is compiled.
  4. x/1000: the expected number of occurrences of the amino acid, codon pair per 1000 translations in genes.
  5. Fraction: the fraction of occurrences of the codon in its synonymous codon family.

The product provides sample codon frequency tables in the sqllib/samples/lifesci/ls_udfs subdirectory.

Related reference:

- "LSPep2ProbNuc user-defined function" on page 451
- "Codon frequency table - example" on page 479

Codon frequency table - example

Figure 46 shows the format of a sample codon frequency table.

```
Am-Acid Codon    Number       x/1000   Fraction   ..

Gly     GGG      198.00       18.34        0.23
Gly     GGA       71.00        6.58        0.08
Gly     GGT       66.00        6.11        0.08
Gly     GGC      527.00       48.81        0.61

Glu     GAG      534.00       49.46        0.88
Glu     GAA       71.00        6.58        0.12
Asp     GAT       31.00        2.87        0.06
Asp     GAC      481.00       44.55        0.94

Val     GTG      396.00       36.68        0.47
Val     GTA       22.00        2.04        0.03
Val     GTT       44.00        4.08        0.05
Val     GTC      384.00       35.57        0.45

Ala     GCG      446.00       41.31        0.39
Ala     GCA       71.00        6.58        0.06
Ala     GCT      116.00       10.74        0.10
Ala     GCC      503.00       46.59        0.44
... (truncated)
```

Figure 46. Sample codon frequency table

Related reference:

- "LSPep2ProbNuc user-defined function" on page 451
- "Codon frequency table format" on page 478

Translation table format

This topic describes the format of a translation table that are used by the LSPep2AmbNuc, LSTransAllFrames, and LSNuc2Pep life sciences user-defined functions.

The following list describes the format of the codon frequency table file:

- Two adjacent periods mark the beginning of the table. Any text that comes before is commentary.
- Each line of the table consists of a single-letter amino acid symbol, the three-letter amino acid name, the unambiguous codons, an exclamation mark, and the ambiguous codons. White space separates each word in the line.
- Each codon and amino acid symbol must appear only once in the file.
- Stop codons translate to the symbol '*'.
- Codons made up of lowercase letters are start codons.
- All other codons are uppercase.
- Codons that do not have a translation to a corresponding amino acid symbol are translated to the symbol 'X'.

The product provides sample translation tables in the sqllib/samples/lifesci/ls_udfs subdirectory.

Related concepts:

- "Life sciences user-defined functions - overview" on page 443

Related reference:

- "Translation table - example" on page 480

Translation table - example

Figure 47 on page 481 shows the format of a sample translation table.

```
                Standard Translation Table

Symbol   3-letter    Codons                    ! IUPAC     ..

 A        Ala         GCT GCC GCA GCG           ! GCX
 B        Asx                                   ! RAY
 C        Cys         TGT TGC                   ! TGY
 D        Asp         GAT GAC                   ! GAY
 E        Glu         GAA GAG                   ! GAR
 F        Phe         TTT TTC                   ! TTY
 G        Gly         GGT GGC GGA GGG           ! GGX
 H        His         CAT CAC                   ! CAY
 I        Ile         ATT ATC ATA               ! ATH
 K        Lys         AAA AAG                   ! AAR
 L        Leu         TTG TTA CTT CTC CTA CTG   ! TTR CTX YTR     ; YTX
 M        Met         atg                       ! ATG
 N        Asn         AAT AAC                   ! AAY
 P        Pro         CCT CCC CCA CCG           ! CCX
 Q        Gln         CAA CAG                   ! CAR
 R        Arg         CGT CGC CGA CGG AGA AGG   ! CGX AGR MGR     ; MGX
 S        Ser         TCT TCC TCA TCG AGT AGC   ! TCX AGY         ; WSX
 T        Thr         ACT ACC ACA ACG           ! ACX
 V        Val         GTT GTC GTA GTG           ! GTX
 W        Trp         TGG                       ! TGG
 X        Xxx                                   ! XXX
 Y        Tyr         TAT TAC                   ! TAY
 Z        Glx                                   ! SAR
 *        End         TAA TAG TGA               ! TAR TRA         ; TRR
```

Figure 47. Sample translation table

Related concepts:

- “Life sciences user-defined functions - overview” on page 443

Related reference:

- “Translation table format” on page 480

Chapter 26. KEGG user-defined functions

This chapter explains what the KEGG user-defined functions are, how to add them to your federated system, and how to use them in your queries.

KEGG user-defined functions - overview

The Kyoto Encyclopedia of Genes and Genomes (KEGG) is a suite of databases that contain genomic information. The KEGG user-defined functions are a set of functions provided with DB2® Information Integrator to access the genomic information in the KEGG databases.

The Pathway database and Sequence Similarity Database (SSDB) are the only two databases in the KEGG suite that DB2 Information Integrator can access through the KEGG web services interface. The Pathway database is a collection of data about molecular interaction networks in biological processes, including metabolic pathways, regulatory pathways, and molecular. The SSDB is a collection of data about protein-coding genes in the complete genomes complexes.

The KEGG user-defined functions use the KEGG API to access these databases.

Many of the KEGG methods return lists of values, such as genes or pathways. Some of these methods also require lists of values as input. To facilitate the composition of complex operations from multiple methods, most of the KEGG user-defined function exist in both table and scalar formats. The table functions return a table of single values. The scalar functions return values as a space-delimited list.

The KEGG user-defined functions are installed with the life sciences user-defined functions component of the nonrelational wrappers. After the KEGG user-defined functions are installed, you must register the functions.

To avoid conflicts with namespaces, all of the KEGG user-defined functions are registered in the DB2LS schema.

Related tasks:

- "Registering the KEGG user-defined functions" on page 486
- "Disabling the KEGG user-defined functions" on page 513

Related reference:

- "Function arguments for the KEGG user-defined functions" on page 485
- "KEGG user-defined functions by functional category" on page 483

KEGG user-defined functions by functional category

DB2 Information Integrator includes KEGG user-defined functions to access data in the Pathway database and Sequence Similarity (SSDB) database.

The following table lists the user-defined functions that you can use to retrieve data from the Pathway database.

Table 124. Pathway user-defined functions

Description	Function types	Function names
Compounds by pathway	scalar table	CompoundsByPathwyS CompoundsByPathwyT
Enzymes by pathways	scalar table	EnzymesByPathwyS EnzymesByPathwyT
Genes by pathways	scalar table	GenesByPathwyS GenesByPathwyT
Pathways by compound	scalar table	PathwysByCompndsS PathwysByCompndsT
Pathways by enzymes	scalar table	PathwysByEnzymesS PathwysByEnzymesT
Pathways by genes	scalar table	PathwysByGenesS PathwysByGenesT

The following table lists the user-defined functions that you can use to retrieve data from the SSDB database.

Table 125. SSDB user-defined functions

Description	Function types	Function names
All neighbors by gene	scalar table	AllNbrsByGeneS AllNbrsByGeneT
Best neighbors by gene	scalar table	BestNbrsByGeneS BestNbrsByGeneT
Best-best neighbors by gene	scalar table	BstBstNbrsByGeneS BstBstNbrsByGeneT
Reverse best neighbors by gene	scalar table	RevBestNbrsByGeneS RevBestNbrsByGeneT
Homologs by gene	scalar table	BestHmlgsByGenesS BestHmlgsByGenesT
Best-best homologs by gene	scalar table	BstBstHmlgByGenesS BstBstHmlgByGenesT
Paralogs by gene	scalar table	ParalogsByGeneS ParalogsByGeneT
Definitions by gene	scalar	DefinitionsByGeneS
Genes by motifs	table	GenesByMotifsT
Smith-Waterman score between genes	scalar	ScoreBetweenGenesS

All of the table user-defined functions that are for the SSDB database return a fixed set of output columns, except for the GetGenesByMotifsT function. The GetGenesByMotifsT function returns the *keggid* VARCHAR(100) and the *definition* VARCHAR(1000) for each gene.

Related concepts:

- "KEGG user-defined functions - overview" on page 483
- "Columns that are returned from SSDB database queries (table functions)" on page 497

Related tasks:

- “Registering the KEGG user-defined functions” on page 486

Function arguments for the KEGG user-defined functions

The KEGG user-defined functions use a set of common arguments. The arguments for the KEGG user-defined function are described in the following list.

cpdlist
A list of compounds. The list must have a data type of VARCHAR and an actual length that is no greater than 1630 bytes. The format of each entry in the list must be:

```
cpd:compound
```

compound is the compound identifier. Each entry in the list must be delimited by spaces, commas, or semicolons.

enzymelist
A list of enzymes. The list must have a data type of VARCHAR and an actual length that is no greater than 1630 bytes. Each entry in the list must be delimited by spaces, commas, or semicolons.

keggid
A unique KEGG identifier for each organism, expressed as a character string. In the KEGG API, this identifier is called gene_ID.

The identifier must have a data type of VARCHAR and an actual length that is no greater than 100 bytes. The format that you use for the *keggid* argument is `keggorg:gene_name`. The `keggorg` is the 3-letter KEGG organism code and is expressed as a character string. The `gene_name` is the name of the gene.

Each organism in the KEGG databases is assigned an identifier. The list of organisms changes frequently. Check the current list of genomes at http://www.genome.ad.jp/kegg/kegg2.html#genes for the correct identifier.

keggidlist
A list of KEGG identifiers. The list is a character string of delimited values. The list must have a data type of VARCHAR and an actual length that is no greater than 1630 bytes. Each entry in the list must be delimited by spaces, commas, or semicolons.

orglist A list of KEGG organism names. The list is a character string of delimited values. The list must have a data type of VARCHAR and an actual length that is no greater than 1630 bytes. Each entry in the list must be delimited by spaces, commas, or semicolons.

If this parameter is not specified, the entire organism list is searched.

midlist
A list of motif identifiers. The *midlist* is a character string of delimited values. The list must have a data type of VARCHAR and an actual length that is no greater than 1630 bytes. Each motif identifier in the midlist must be delimited by spaces, commas, or semicolons.

The format of each motif identifier is `database:entry`. The database is an identifier for a motif database and the entry is a motif entry in that database. For example, the motif identifier for the DnaJ entry in the pfam database is `pf:DnaJ`.

pathwayid
A 3-part name that identifies a particular pathway. The name is a character string. The name must have a data type of VARCHAR and an actual length that is no greater than 100 bytes. The format of the *pathwayid* must be:

```
path:org map
```

org is a KEGG organism identifier, and *map* is a pathway map identifier.

threshold
A value for a Smith-Waterman score. The value must be equal to or greater than 100. The value must have a data type of INTEGER.

Related concepts:
- "KEGG user-defined functions - overview" on page 483

Related tasks:
- "Registering the KEGG user-defined functions" on page 486

Related reference:
- "KEGG user-defined functions by functional category" on page 483

Registering the KEGG user-defined functions

Before you can use the KEGG user-defined functions, you must register the functions.

The enable_KEGGFunctions command registers all of the KEGG user-defined functions in the federated database. The functions are registered with the schema name DB2LS.

Prerequisites:

The life sciences user-defined functions component of the nonrelational wrappers must be installed with DB2 Information Integrator.

Procedure:

To register the KEGG user-defined functions, run the enable_KEGGFunctions command.
- On federated servers that run Windows, this command is in the `sqllib\bin` directory
- On federated servers that run UNIX, this command is in the `sqllib/bin` directory

Syntax:

```
enable_KEGGFunctions -n dbName -u userID -p password [-force]
```

dbName
The name of the federated database that you are registering the functions in.

userID
A valid user ID for the federated database.

password
A valid password for the user ID.

force An optional flag that you can use to remove the functions and register them again. Use this flag if the functions get corrupted or dropped accidentally.

Example of registering the user-defined functions:

The following example shows the output that is returned when you issue the enable_KEGGFunctions command:

```
C:> enable_KEGGFunctions -n federateddb -u db2admin -p db2admin

(0) KEGG Functions were found
    -- Create KEGG Functions ...
    Create KEGG Functions Successfully.

*** Please allow a few seconds to clean up the system ......
```

Example of using the force flag to drop and register the user-defined functions:

The following example shows the output that is returned when you issue the enable_KEGGFunctions command with the force flag, and the user-defined functions are already registered:

```
C:> enable_KEGGFunctions -n federateddb -u db2admin -p db2admin -force

(37) KEGG Functions were found

KEGG functions already exist ...
Reinstall KEGG functions ...
    -- Drop KEGG Functions ...
    Drop KEGG Functions Successfully.
    -- Create KEGG Functions ...
    Create KEGG Functions Successfully.


*** Please allow a few seconds to clean up the system ......
```

Related concepts:
- "KEGG user-defined functions - overview" on page 483

Related tasks:
- "Adding relational wrappers, nonrelational wrappers, and user-defined functions to your DB2 Information Integrator system" in the *IBM DB2 Information Integrator Installation Guide for Linux, UNIX, and Windows*
- "Disabling the KEGG user-defined functions" on page 513

Related reference:
- "Function arguments for the KEGG user-defined functions" on page 485
- "KEGG user-defined functions by functional category" on page 483

Pathway database functions

The following sections describe the user-defined functions for the Pathway database. These sections contain the syntax and examples for each function.

GenesByPathwyS user-defined function

Use the GenesByPathwyS function to search for all of the genes on a pathway.

The GenesByPathwyS function is a scalar function that returns a space-delimited list of genes. The list is a character string with a data type of VARCHAR and an actual length that is not greater than 32767 bytes.

You can use this function in a SELECT statement.

Syntax:

```
►►──DB2LS.GenesByPathwyS──(pathwayid)──────────────────────────────────────►◄
```

The schema name for this user-defined function is DB2LS.

Example of searching for all of the genes on a pathway:

To search for all of the genes on a pathway, you specify the *pathwayid* argument in the function.

For example, you want to search for all of the E. coli genes on the pathway map 00020.

The clause that you use is:

```
VALUES CAST(DB2LS.GenesByPathwyS
    ('path:eco00020')
    AS VARCHAR(1000));
```

Related concepts:
- "KEGG user-defined functions - overview" on page 483

Related tasks:
- "Registering the KEGG user-defined functions" on page 486

Related reference:
- "Function arguments for the KEGG user-defined functions" on page 485
- "KEGG user-defined functions by functional category" on page 483

GenesByPathwyT user-defined function

Use the GenesByPathwyT function to search for all of the genes on a pathway.

The GenesByPathwyT function is a table function that returns a VARCHAR(100) column with the names of the genes on the pathway. The name of the column that is returned is gene.

You can use this function in a SELECT statement.

Syntax:

```
►►──DB2LS.GenesByPathwyT──(pathwayid)──────────────────────────────────────►◄
```

The schema name for this user-defined function is DB2LS.

Example of searching for all of the genes on a pathway:

To search for all of the genes on a particular pathway, you specify the *pathwayid* argument in the function.

For example, you want to search for all of the E. coli genes on the pathway map 00020.

The SELECT statement is:

```
SELECT * FROM TABLE(DB2LS.GenesByPathwyT
    ('path:eco00020')) AS t;
```

Related concepts:

- “KEGG user-defined functions - overview” on page 483

Related tasks:

- “Registering the KEGG user-defined functions” on page 486

Related reference:

- “Function arguments for the KEGG user-defined functions” on page 485
- “KEGG user-defined functions by functional category” on page 483

CompoundsByPathwyS user-defined function

Use the CompoundsByPathwyS function to search for all of the compounds on a pathway.

The CompoundsByPathwyS function is a scalar function that returns a space-delimited list of compounds. The list is a character string with a data type of VARCHAR and an actual length that is not greater than 32767 bytes.

You can use this function in a SELECT statement.

Syntax:

```
►►──DB2LS.CompoundsByPathwyS──(pathwayid)──────────────────────────────►◄
```

The schema name for this user-defined function is DB2LS.

Example of searching for all of the compounds on a pathway:

To search for the all of the compounds on a pathway, you specify the *pathwayid* argument in the function.

For example, you want to search for all of the compounds on a the pathway map 00020.

The clause that you use is:

```
VALUES CAST(DB2LS.CompoundsByPathwyS
    ('path:00020')
    AS VARCHAR(1000));
```

Related concepts:
- "KEGG user-defined functions - overview" on page 483

Related tasks:
- "Registering the KEGG user-defined functions" on page 486

Related reference:
- "Function arguments for the KEGG user-defined functions" on page 485
- "KEGG user-defined functions by functional category" on page 483

CompoundsByPathwyT user-defined function

Use the CompoundsByPathwyT function to search for all of the compounds on a pathway.

The CompoundsByPathwyT function is a table function that returns a VARCHAR(100) column with the names of the compounds on the pathway. The name of the column that is returned is `compound`.

You can use this function in a SELECT statement.

Syntax:

```
►►──DB2LS.CompoundsByPathwyT──(pathwayid)──────────────────────────────────►◄
```

The schema name for this user-defined function is DB2LS.

Example of searching for all of the compounds on a pathway:

To search for all of the compounds on a pathway, you specify the *pathwayid* argument in the function.

For example, you want to search for all of the compounds on a the pathway map 00020.

The SELECT statement is:

```
SELECT * FROM TABLE(DB2LS.CompoundsByPathwyT
    ('path:00020')) AS t;
```

Related concepts:
- "KEGG user-defined functions - overview" on page 483

Related tasks:
- "Registering the KEGG user-defined functions" on page 486

Related reference:
- "Function arguments for the KEGG user-defined functions" on page 485
- "KEGG user-defined functions by functional category" on page 483

EnzymesByPathwyS user-defined function

Use the EnzymesByPathwyS function to search for all of the enzymes on a pathway.

The EnzymesByPathwyS function is a scalar function that returns a space-delimited list of enzymes. The list is a character string with a data type of VARCHAR and an actual length that is not greater than 32767 bytes.

You can use this function in a SELECT statement.

Syntax:

```
►►──DB2LS.EnzymesByPathwyS──(pathwayid)──────────────────────────────────►◄
```

The schema name for this user-defined function is DB2LS.
Example of searching for all of the enzymes on a pathway:
To search for the all of the enzymes on a pathway, you specify the *pathwayid* argument in the function.

For example, you want to search for all of the enzymes on the pathway map 00020.

The clause that you use is:

```
VALUES CAST(DB2LS.EnzymesByPathwyS
    ('path:00020')
    AS VARCHAR(1000));
```

Related concepts:
- "KEGG user-defined functions - overview" on page 483

Related tasks:
- "Registering the KEGG user-defined functions" on page 486

Related reference:
- "Function arguments for the KEGG user-defined functions" on page 485
- "KEGG user-defined functions by functional category" on page 483

EnzymesByPathwyT user-defined function

Use the EnzymesByPathwyT function to search for all of the enzymes on a pathway.

The EnzymesByPathwyT function is a table function that returns a VARCHAR(100) column with the names of the enzymes on the pathway. The name of the column that is returned is enzyme.

You can use this function in a SELECT statement.

Syntax:

```
►►──DB2LS.EnzymesByPathwyT──(pathwayid)──────────────────────────────────►◄
```

The schema name for this user-defined function is DB2LS.

Example of searching for all of the enzymes on a pathway:

To search for the all of the enzymes on a pathway, you specify the *pathwayid* argument in the function.

For example, you want to search for all of the compounds on a the pathway map 00020.

The SELECT statement is:

```
SELECT * FROM TABLE(DB2LS.EnzymesByPathwyT
    ('path:00020')) AS t;
```

Related concepts:

- "KEGG user-defined functions - overview" on page 483

Related tasks:

- "Registering the KEGG user-defined functions" on page 486

Related reference:

- "Function arguments for the KEGG user-defined functions" on page 485
- "KEGG user-defined functions by functional category" on page 483

PathwysByGenesS user-defined function

Use the PathwysByGenesS function to search for all of the pathways that contain the genes that you specify.

The PathwysByGenesS function is a scalar function that returns a space-delimited list of pathways. The list is a character string with a data type of VARCHAR and an actual length that is not greater than 32767 bytes.

You can use this function in a SELECT statement.

Syntax:

```
►►──DB2LS.PathwysByGenesS──(keggidlist)──────────────────────────────────►◄
```

The schema name for this user-defined function is DB2LS.

Example of searching for all of the pathways that contain specific genes:

To search for the all of the pathways that contain certain genes, you specify the *keggidlist* argument in the function.

For example, you want to search for all of the pathways that contain the E. coli genes b0077 and b0078.

The clause that you use is:

```
VALUES CAST(DB2LS.PathwysByGenesS
    ('eco:b0077 eco:0078')
    AS VARCHAR(1000));
```

Related concepts:

- “KEGG user-defined functions - overview” on page 483

Related tasks:
- “Registering the KEGG user-defined functions” on page 486

Related reference:
- “Function arguments for the KEGG user-defined functions” on page 485
- “KEGG user-defined functions by functional category” on page 483

PathwysByGenesT user-defined function

Use the PathwysByGenesT function to search for all of the pathways that contain the genes that you specify.

The PathwysByGenesT function is a table function that returns a VARCHAR(100) column with the names of the pathways for the gene. The name of the column that is returned is `pathway`.

You can use this function in a SELECT statement.

Syntax:

```
►►—DB2LS.PathwysByGenesT—(keggidlist)———————————————————————————►◄
```

The schema name for this user-defined function is DB2LS.

Example of searching for all of the pathways that contain specific genes:

To search for the all of the pathways that contain certain genes, you specify the *keggidlist* argument in the function.

For example, you want to search for all of the pathways that contain the E. coli genes b0077 and b0078.

The SELECT statement is:

```
SELECT * FROM TABLE(DB2LS.PathwysByGenesT
    ('eco:b0077 eco:0078')) AS t;
```

Related concepts:
- “KEGG user-defined functions - overview” on page 483

Related tasks:
- “Registering the KEGG user-defined functions” on page 486

Related reference:
- “Function arguments for the KEGG user-defined functions” on page 485
- “KEGG user-defined functions by functional category” on page 483

PathwysByCompndsS user-defined function

Use the PathwysByCompndsS function to search for all of the pathways that contain all of compounds that you specify.

The PathwysByCompndsS function is a scalar function that returns a space-delimited list of compounds. The list is a character string with a data type of VARCHAR and an actual length that is not greater than 32767 bytes.

You can use this function in a SELECT statement.

Syntax:

```
►►──DB2LS.PathwysByCompndsS──(cpdlist)─────────────────────────────────►◄
```

The schema name for this user-defined function is DB2LS.

Example of searching for the pathways that contain all of the specified compounds:

To search for all of the pathways that contain all of the compounds in a list, you specify the *cpdlist* argument in the function.

For example, you want to search for all of the pathways that contain the compounds C00033 and C00158.

The clause that you use is:

```
VALUES CAST(DB2LS.PathwysByCompndsS
    ('cpd:C00033 cpd:C00158')
    AS VARCHAR(1000));
```

Related concepts:
- "KEGG user-defined functions - overview" on page 483

Related tasks:
- "Registering the KEGG user-defined functions" on page 486

Related reference:
- "Function arguments for the KEGG user-defined functions" on page 485
- "KEGG user-defined functions by functional category" on page 483

PathwysByCompndsT user-defined function

Use the PathwysByCompndsT function to search for all of the pathways that contain all of compounds that you specify.

The PathwysByCompndsT function is a table function that returns a VARCHAR(100) column with the names of the pathways. The name of the column that is returned is `pathway`.

You can use this function in a SELECT statement.

Syntax:

```
►►──DB2LS.PathwysByCompndsT──(cpdlist)─────────────────────────────────►◄
```

The schema name for this user-defined function is DB2LS.

Example of searching for the pathways that contain all of the specified compounds:

To search for all of the pathways that contain all of the compounds in a list, you specify the *cpdlist* argument in the function.

For example, you want to search for all of the pathways that contain the compounds C00033 and C00158.

The SELECT statement is:

```
SELECT * FROM TABLE(DB2LS.PathwysByCompndsT
    ('cpd:C00033 cpd:C00158')) AS t;
```

Related concepts:

- "KEGG user-defined functions - overview" on page 483

Related tasks:

- "Registering the KEGG user-defined functions" on page 486

Related reference:

- "Function arguments for the KEGG user-defined functions" on page 485
- "KEGG user-defined functions by functional category" on page 483

PathwysByEnzymesS user-defined function

Use the PathwysByEnzymesS function to search for all of the pathways that contain all of the enzymes that you specify.

The PathwysByEnzymesS function is a scalar function that returns a space-delimited list of pathways. The list is a character string with a data type of VARCHAR and an actual length that is not greater than 32767 bytes.

You can use this function in a SELECT statement.

Syntax:

```
►►──DB2LS.PathwysByEnzymesS──(emzymelist)──────────────────────────────►◄
```

The schema name for this user-defined function is DB2LS.

Example of searching for all of the pathways that contain all of the specified enzymes:

To search for all of the pathways that contain all of the enzymes in a list, you specify the *enzymelist* argument in the function.

For example, you want to search for all of the pathways that contain the enzyme 1.3.99.1.

The clause that you use is:

```
VALUES CAST(DB2LS.PathwysByEnzymesS
    ('ec:1.3.99.1')
    AS VARCHAR(1000));
```

Related concepts:
- "KEGG user-defined functions - overview" on page 483

Related tasks:
- "Registering the KEGG user-defined functions" on page 486

Related reference:
- "Function arguments for the KEGG user-defined functions" on page 485
- "KEGG user-defined functions by functional category" on page 483

PathwysByEnzymesT user-defined function

Use the PathwysByEnzymesT function to search for all of the pathways that contain all of the enzymes that you specify.

The PathwysByEnzymesT function is a table function that returns a VARCHAR(100) column with the names of the pathways. The name of the column that is returned is `pathway`.

You can use this function in a SELECT statement.

Syntax:

```
►►──DB2LS.PathwysByEnzymesT──(emzymelist)──────────────────────────────────►◄
```

The schema name for this user-defined function is DB2LS.

Example of searching for all of the pathways that contain all of the specified enzymes:

To search for all of the pathways that contain all of the enzymes in a list, you specify the *enzymelist* argument in the function.

For example, you want to search for all of the pathways that contain the enzyme 1.3.99.1.

The SELECT statement is:

```
SELECT * FROM TABLE(DB2LS.PathwysByEnzymesT
    ('ec:1.3.99.1')) AS t;
```

Related concepts:
- "KEGG user-defined functions - overview" on page 483

Related tasks:
- "Registering the KEGG user-defined functions" on page 486

Related reference:
- "Function arguments for the KEGG user-defined functions" on page 485
- "KEGG user-defined functions by functional category" on page 483

Sequence Similarity Database functions

The following topics describe the user-defined functions for the Sequence Similarity database (SSDB). These sections contain the syntax and examples for each function.

Columns that are returned from SSDB database queries (table functions)

Many of the user-defined table functions for the Sequence Similarity Database (SSDB) return a set of fixed output columns. When you use an SSDB table function, you can specify that only a subset of the columns are returned. The names and data types of the columns that are returned are listed in the following table.

Table 126. Columns in the SSDB database

Column name	Column data type	Description
keggid1	VARCHAR (100)	The identifier for the gene that is specified in the query.
keggid2	VARCHAR (100)	The identifier for the gene that is returned from the query.
swscore	DOUBLE	The Smith-Waterman score between keggid1 and keggid2.
identity	DOUBLE	The identity percent between keggid1 and keggid2.
overlap	INTEGER	The overlap length between keggid1 and keggid2.
s1_start	INTEGER	The start position of the alignment in keggid1.
s1_end	INTEGER	The end position of the alignment in keggid1.
s2_start	INTEGER	The start position of the alignment in keggid2.
s2_end	INTEGER	The end position of the alignment in keggid2.
best1	INTEGER	The flag that indicates the best hit from keggid1 to keggid2. A value of 1 indicates a best hit relationship between keggid1 to keggid2. A value of 0 indicates that there is not a best hit relationship between keggid1 to keggid2.
best2	INTEGER	The flag that indicates the best hit from keggid2 to keggid1. A value of 1 indicates a best hit relationship between keggid2 to keggid1. A value of 0 indicates that there is not a best hit relationship between keggid2 to keggid1.
def1	VARCHAR (1000)	The definition for keggid1.
def2	VARCHAR (1000)	The definition for keggid2.
length1	INTEGER	The length of the amino acid in keggid1.
length2	INTEGER	The length of the amino acid in keggid2.

Related concepts:

- "KEGG user-defined functions - overview" on page 483

Related reference:

- “GenesByPathwyT user-defined function” on page 488
- “AllNbrsByGeneT user-defined function” on page 499
- “BstBstNbrsByGeneT user-defined function” on page 501
- “BestNbrsByGeneT user-defined function” on page 503
- “RevBestNbrsByGeneT user-defined function” on page 505
- “ParalogsByGeneT user-defined function” on page 507
- “BestHmlgsByGenesT user-defined function” on page 509
- “BstBstHmlgByGenesT user-defined function” on page 510

AllNbrsByGeneS user-defined function

Use the AllNbrsByGeneS function to search for all organisms that are homologous neighbors of the KEGG identifier that you specify.

Instead of searching for all organisms, you can specify a list of organisms to narrow the search.

The AllNbrsByGeneS function is a scalar function that returns a space-delimited list of target identifiers. The list is a character string with a data type of VARCHAR and an actual length that is not greater than 32767 bytes.

You can use this function in a SELECT statement.

Syntax:

```
►►──DB2LS.AllNbrsByGeneS──(keggid, threshold──┬──────────┬──)────────────────►◄
                                              └─,orglist─┘
```

The schema name for this user-defined function is DB2LS.

Example of searching the entire organism list:

To search the entire organism list, you specify the *keggid* and *threshold* arguments in the function.

For example, to search for all homologous genes for the E. coli gene with a threshold value greater than 200, the clause that you use is:

```
VALUES CAST(DB2LS.AllNbrsByGeneS
    ('eco:b0002', 200)
    AS VARCHAR(1000));
```

The KEGG code for the E. coli gene is *eco*. The gene name is *b0002*.

Example of searching for a specific set of organisms:

To specify a list of organisms to search for, you must specify the *keggid*, *threshold*, and *orglist* arguments in the function.

For example, to search for all homologous genes with a threshold value greater than 500 and return only the 0157 strain of the E. coli gene and all strains of the H. influenzae gene, the clause that you use is:

```
VALUES CAST(DB2LS.AllNbrsByGeneS
    ('eco:b0002', 500, 'ece hin')
    AS VARCHAR(1000));
```

The KEGG code for the E. coli gene is *eco*. The gene name is *b0002*. The KEGG name for the 0157 strain of the E. coli gene is *ece*. The KEGG name for the H. influenzae gene is *hin*.

Related concepts:

- "KEGG user-defined functions - overview" on page 483

Related tasks:

- "Registering the KEGG user-defined functions" on page 486

Related reference:

- "Function arguments for the KEGG user-defined functions" on page 485
- "KEGG user-defined functions by functional category" on page 483

AllNbrsByGeneT user-defined function

Use the AllNbrsByGeneT function to search for all organisms that are homologous neighbors of the KEGG identifier that you specify.

Instead of searching for all organisms, you can specify a list of organisms to narrow the search.

The AllNbrsByGeneT function is a table function that returns a fixed set of output columns.

You can use this function in a SELECT statement.

Syntax:

```
►►──DB2LS.AllNbrsByGeneT──(keggid, threshold──┬──────────┬──)──────────────────►◄
                                              └─,orglist─┘
```

The schema name for this user-defined function is DB2LS.

Example of searching the entire organism list:

To search the entire organism list, you specify the *keggid* and *threshold* arguments in the function.

For example, to search for all homologous genes for the E. coli gene with a threshold value greater than 200, the SELECT statement is:

```
SELECT * FROM TABLE(DB2LS.AllNbrsByGeneT
    ('eco:b0002', 200)) AS t;
```

The KEGG code for the E. coli gene is *eco*. The gene name is *b0002*.

Example of searching for a specific set of organisms:

To specify a list of organisms to search for, you must specify the *keggid*, *threshold*, and *orglist* arguments in the function.

For example, to search for all homologous genes with a threshold value greater than 500 and return only the 0157 strain of the E. coli gene and all strains of the H. influenzae gene, the SELECT statement is:

```
SELECT * FROM TABLE(DB2LS.AllNbrsByGeneT
    ('eco:b0002', 500, 'ece hin')) AS t;
```

The KEGG code for the E. coli gene is *eco*. The gene name is *b0002*. The KEGG name for the 0157 strain of the E. coli gene is *ece*. The KEGG name for the H. influenzae gene is *hin*.

Related concepts:

- “KEGG user-defined functions - overview” on page 483
- “Columns that are returned from SSDB database queries (table functions)” on page 497

Related tasks:

- “Registering the KEGG user-defined functions” on page 486

Related reference:

- “Function arguments for the KEGG user-defined functions” on page 485

BstBstNbrsByGeneS user-defined function

Use the BstBstNbrsByGeneS function to search for the best-best neighbors of the gene in each organism.

The BstBstNbrsByGeneS function is a scalar function that returns a space-delimited list of target identifiers. The list is a character string with a data type of VARCHAR and an actual length that is not greater than 32767 bytes.

You can use this function in a SELECT statement.

Syntax:

```
►►──DB2LS.BstBstNbrsByGeneS──(keggid, threshold──┬──────────┬──)──────────────►◄
                                                 └─,orglist─┘
```

The schema name for this user-defined function is DB2LS.

Example of searching the entire organism list:

To search the entire organism list, you specify the *keggid* and *threshold* arguments in the function.

For example, to search for the best-best neighbors of the E. coli gene with a threshold value greater than 200, the clause that you use is:

```
VALUES CAST(DB2LS.BstBstNbrsByGeneS
    ('eco:b0002', 200)
    AS VARCHAR(1000));
```

The KEGG code for the E. coli gene is *eco*. The gene name is *b0002*.

Example of searching for a specific set of organisms:

To specify a list of organisms to search for, you must specify the *keggid*, *threshold*, and *orglist* arguments in the function.

For example, to search for the best-best neighbors with a threshold value greater than 500 and return only the 0157 strain of the E. coli gene and all strains of the H. influenzae gene, the clause that you use is:

```
VALUES CAST(DB2LS.BstBstNbrsByGeneS
    ('eco:b0002', 500, 'ece hin')
    AS VARCHAR(1000));
```

The KEGG code for the E. coli gene is *eco*. The gene name is *b0002*. The KEGG name for the 0157 strain of the E. coli gene is *ece*. The KEGG name for the H. influenzae gene is *hin*.

Related concepts:

- “KEGG user-defined functions - overview” on page 483

Related tasks:

- “Registering the KEGG user-defined functions” on page 486

Related reference:

- “Function arguments for the KEGG user-defined functions” on page 485
- “KEGG user-defined functions by functional category” on page 483

BstBstNbrsByGeneT user-defined function

Use the BstBstNbrsByGeneT function to search for the best-best neighbors of the gene in each organism.

The BstBstNbrsByGeneT function is a table function that returns a fixed set of output columns.

You can use this function in a SELECT statement.

Syntax:

```
►►──DB2LS.BstBstNbrsByGeneT──(keggid, threshold──┬───────────┬──)────────────────►◄
                                                 └─,orglist──┘
```

The schema name for this user-defined function is DB2LS.

Example of searching the entire organism list:

To search the entire organism list, you specify the *keggid* and *threshold* arguments in the function.

For example, to search for the best-best neighbors of the E. coli gene with a threshold value greater than 200, the SELECT statement is:

```
SELECT * FROM TABLE(DB2LS.BstBstNbrsByGeneT
    ('eco:b0002', 200)) AS t;
```

The KEGG code for the E. coli gene is *eco*. The gene name is *b0002*.

Example of searching for a specific set of organisms:

To specify a list of organisms to search for, you must specify the *keggid*, *threshold*, and *orglist* arguments in the function.

For example, to search for the best-best neighbors with a threshold value greater than 500 and return only the 0157 strain of the E. coli gene and all strains of the H. influenzae gene, the SELECT statement is:

```
SELECT * FROM TABLE(DB2LS.BstBstNbrsByGeneT
    ('eco:b0002', 500, 'ece hin')) AS t;
```

The KEGG code for the E. coli gene is *eco*. The gene name is *b0002*. The KEGG name for the 0157 strain of the E. coli gene is *ece*. The KEGG name for the H. influenzae gene is *hin*.

Related concepts:

- “KEGG user-defined functions - overview” on page 483
- “Columns that are returned from SSDB database queries (table functions)” on page 497

Related tasks:

- “Registering the KEGG user-defined functions” on page 486

Related reference:

- “Function arguments for the KEGG user-defined functions” on page 485
- “KEGG user-defined functions by functional category” on page 483

BestNbrsByGeneS user-defined function

Use the BestNbrsByGeneS function to search for the best neighbors of the gene in each organism.

The BestNbrsByGeneS function is a scalar function that returns a space-delimited list of target identifiers. The list is a character string with a data type of VARCHAR and an actual length that is not greater than 32767 bytes.

You can use this function in a SELECT statement.

Syntax:

```
►►──DB2LS.BestNbrsByGeneS──(keggid, threshold─┬──────────┬─)──────────────────►◄
                                              └─,orglist─┘
```

The schema name for this user-defined function is DB2LS.

Example of searching the entire organism list:

To search the entire organism list, you specify the *keggid* and *threshold* arguments in the function.

For example, to search for the best neighbors of the E. coli gene with a threshold value greater than 200, the clause that you use is:

```
VALUES CAST(DB2LS.BestNbrsByGeneS
    ('eco:b0002', 200)
    AS VARCHAR(1000));
```

The KEGG code for the E. coli gene is *eco*. The gene name is *b0002*.

Example of searching for a specific set of organism:

To specify a list of organisms to search for, you must specify the *keggid*, *threshold*, and *orglist* arguments in the function.

For example, to search for the best neighbors with a threshold value greater than 500, and return only the 0157 strain of the E. coli gene and all strains of the H. influenzae gene, the clause that you use is:

```
VALUES CAST(DB2LS.BestNbrsByGeneS
    ('eco:b0002', 500, 'ece hin')
    AS VARCHAR(1000));
```

The KEGG code for the E. coli gene is *eco*. The gene name is *b0002*. The KEGG name for the 0157 strain of the E. coli gene is *ece*. The KEGG name for the H. influenzae gene is *hin*.

Related concepts:
- "KEGG user-defined functions - overview" on page 483

Related tasks:
- "Registering the KEGG user-defined functions" on page 486

Related reference:
- "Function arguments for the KEGG user-defined functions" on page 485
- "KEGG user-defined functions by functional category" on page 483

BestNbrsByGeneT user-defined function

Use the BestNbrsByGeneT function to search for the best neighbors of the gene in each organism.

The BestNbrsByGeneT function is a table function that returns a fixed set of output columns.

You can use this function in a SELECT statement.

Syntax:

```
►►──DB2LS.BestNbrsByGeneT──(keggid, threshold──┬──────────┬──)──────────────►◄
                                               └─,orglist─┘
```

The schema name for this user-defined function is DB2LS.

Example of searching the entire organism list:

To search the entire organism list, you specify the *keggid* and *threshold* arguments in the function.

For example, to search for the best neighbors of the E. coli gene with a threshold value greater than 200, the SELECT statement is:

```
SELECT * FROM TABLE(DB2LS.BestNbrsByGeneT
    ('eco:b0002', 200)) AS t;
```

The KEGG code for the E. coli gene is *eco*. The gene name is *b0002*.

Example of searching for a specific set of organisms:

To specify a list of organisms to search for, you must specify the *keggid*, *threshold*, and *orglist* arguments in the function.

For example, to search for the best neighbors with a threshold value greater than 500 and return only the 0157 strain of the E. coli gene and all strains of the H. influenzae gene, the SELECT statement is:

```
SELECT * FROM TABLE(DB2LS.BestNbrsByGeneT
    ('eco:b0002', 500, 'ece hin')) AS t;
```

The KEGG code for the E. coli gene is *eco*. The gene name is *b0002*. The KEGG name for the 0157 strain of the E. coli gene is *ece*. The KEGG name for the H. influenzae gene is *hin*.

Related concepts:
- “KEGG user-defined functions - overview” on page 483
- “Columns that are returned from SSDB database queries (table functions)” on page 497

Related tasks:
- “Registering the KEGG user-defined functions” on page 486

Related reference:
- “Function arguments for the KEGG user-defined functions” on page 485
- “KEGG user-defined functions by functional category” on page 483

RevBestNbrsByGeneS user-defined function

Use the RevBestNbrsByGeneS function to search for the reverse best neighbors of the gene in each organism.

The RevBestNbrsByGeneS function is a scalar function that returns a space-delimited list of target identifiers. The list is a character string with a data type of VARCHAR and an actual length that is not greater than 32767 bytes.

You can use this function in a SELECT statement.

Syntax:

```
►►──DB2LS.RevBestNbrsByGeneS──(keggid, threshold──┬───────────┬──)──────────────►◄
                                                  └─,orglist──┘
```

The schema name for this user-defined function is DB2LS.

Example of searching the entire organism list:

To search the entire organism list, you specify the *keggid* and *threshold* arguments in the function.

For example, to search for the reverse best neighbors of the E. coli gene with a threshold value greater than 200, the clause that you use is:

```
VALUES CAST(DB2LS.RevBestNbrsByGeneS
    ('eco:b0002', 200)
    AS VARCHAR(1000));
```

The KEGG code for the E. coli gene is *eco*. The gene name is *b0002*.

Example of searching for a specific set of organisms:

To specify a list of organisms to search for, you must specify the *keggid*, *threshold*, and *orglist* arguments in the function.

For example, to search for the reverse best neighbors with a threshold value greater than 500 and return only the 0157 strain of the E. coli gene and all strains of the H. influenzae gene, the clause that you use is:

```
VALUES CAST(DB2LS.RevBestNbrsByGeneS
    ('eco:b0002', 500, 'ece hin')
    AS VARCHAR(1000));
```

The KEGG code for the E. coli gene is *eco*. The gene name is *b0002*. The KEGG name for the 0157 strain of the E. coli gene is *ece*. The KEGG name for the H. influenzae gene is *hin*.

Related concepts:
- "KEGG user-defined functions - overview" on page 483

Related tasks:
- "Registering the KEGG user-defined functions" on page 486

Related reference:
- "Function arguments for the KEGG user-defined functions" on page 485
- "KEGG user-defined functions by functional category" on page 483

RevBestNbrsByGeneT user-defined function

Use the RevBestNbrsByGeneT function to search for the reverse best neighbors of the gene in each organism.

The RevBestNbrsByGeneT function is a table function that returns a fixed set of output columns.

You can use this function in a SELECT statement.

Syntax:

```
►►──DB2LS.RevBestNbrsByGeneT──(keggid, threshold──┬──────────┬──)──────────►◄
                                                  └─,orglist─┘
```

The schema name for this user-defined function is DB2LS.

Example of searching the entire organism list:

To search the entire organism list, you specify the *keggid* and *threshold* arguments in the function.

For example, to search for the reverse best neighbors of the E. coli gene with a threshold value greater than 200, the SELECT statement is:

```
SELECT * FROM TABLE(DB2LS.RevBestNbrsByGeneT
    ('eco:b0002', 200)) AS t;
```

The KEGG code for the E. coli gene is *eco*. The gene name is *b0002*.

Example of searching for a specific set of organisms:

To specify a list of organisms to search for , you must specify the *keggid*, *threshold*, and *orglist* arguments in the function.

For example, to search for the reverse best neighbors with a threshold value greater than 500 and return only the 0157 strain of the E. coli gene and all strains of the H. influenzae gene, the SELECT statement is:

```
SELECT * FROM TABLE(DB2LS.RevBestNbrsByGeneT
    ('eco:b0002', 500, 'ece hin')) AS t;
```

The KEGG code for the E. coli gene is *eco*. The gene name is *b0002*. The KEGG name for the 0157 strain of the E. coli gene is *ece*. The KEGG name for the H. influenzae gene is *hin*.

Related concepts:
- "KEGG user-defined functions - overview" on page 483
- "Columns that are returned from SSDB database queries (table functions)" on page 497

Related tasks:
- "Registering the KEGG user-defined functions" on page 486

Related reference:
- "Function arguments for the KEGG user-defined functions" on page 485
- "KEGG user-defined functions by functional category" on page 483

ParalogsByGeneS user-defined function

Use the ParalogsByGeneS function to search for the paralogous genes in an organism.

The ParalogsByGeneS function is a scalar function that returns a space-delimited list of target identifiers. The list is a character string with a data type of VARCHAR and an actual length that is not greater than 32767 bytes.

You can use this function in a SELECT statement.

Syntax:

```
►►──DB2LS.ParalogsByGeneS──(keggid, threshold)──────────────────────────►◄
```

The schema name for this user-defined function is DB2LS.

Example of searching for paralogous genes:

To search for paralogous genes in an organism, you specify only the *keggid* and *threshold* arguments in the function.

For example, to search for the paralogous genes in the E. coli organism with a threshold value greater than 5000, the clause that you use is:

```
VALUES CAST(DB2LS.ParalogsByGeneS
    ('eco:b0002', 5000)
    AS VARCHAR(1000));
```

The KEGG database name for the E. coli organism is *eco*. The organism name is *b0002*.

Related concepts:
- "KEGG user-defined functions - overview" on page 483

Related tasks:
- "Registering the KEGG user-defined functions" on page 486

Related reference:
- "Function arguments for the KEGG user-defined functions" on page 485
- "KEGG user-defined functions by functional category" on page 483

ParalogsByGeneT user-defined function

Use the ParalogsByGeneT function to search for the paralogous genes in an organism.

The ParalogsByGeneT function is a table function that returns a fixed set of output columns.

You can use this function in a SELECT statement.

Syntax:

```
►►──DB2LS.ParalogsByGeneT──(keggid, threshold)──────────────────────────────►◄
```

The schema name for this user-defined function is DB2LS.

Example of searching for paralogous genes:

To search for paralogous genes in an organism, you specify the *keggid* and *threshold* arguments in the function.

For example, to search for the paralogous genes in the E. coli organism with a threshold value greater than 5000, the SELECT statement is:

```
SELECT * FROM TABLE(DB2LS.ParalogsByGeneT
    ('eco:b0002', 5000)) AS t;
```

The KEGG database name for the E. coli organism is *eco*. The organism name is *b0002*.

Related concepts:

- “KEGG user-defined functions - overview” on page 483
- “Columns that are returned from SSDB database queries (table functions)” on page 497

Related tasks:
- “Registering the KEGG user-defined functions” on page 486

Related reference:
- “Function arguments for the KEGG user-defined functions” on page 485
- “KEGG user-defined functions by functional category” on page 483

BestHmlgsByGenesS user-defined function

Use the BestHmlgsByGenesS function to search for the best homologous neighbors of an organism from a list of genes that you specify.

The BestHmlgsByGenesS function is a scalar function that returns a space-delimited list of target identifiers. The list is a character string with a data type of VARCHAR and an actual length that is not greater than 32767 bytes.

You can use this function in a SELECT statement.

Syntax:

```
►►—DB2LS.BestHmlgsByGenesS—(keggorg—, keggidlist)——————————————————————————►◄
```

The schema name for this user-defined function is DB2LS.

For better performance, use the BstBstHmlgByGenesS function.

Example of searching for the best homologous neighbors:

To search for the best homologous neighbors of an organism from a list of genes, you specify the *keggorg* and *keggidlist* arguments in the function.

For example, to search for the H. influenzae organism in the list of E. coli genes b0002, b0003, b0004, and b0005, the clause that you use is:

```
VALUES CAST(DB2LS.BestHmlgsByGenesS
    ('hin', 'eco:b0002 eco:b0003 eco:b0004 eco:b0005')
    VARCHAR(1000));
```

Related concepts:
- “KEGG user-defined functions - overview” on page 483

Related tasks:
- “Registering the KEGG user-defined functions” on page 486

Related reference:
- “Function arguments for the KEGG user-defined functions” on page 485
- “KEGG user-defined functions by functional category” on page 483

BestHmlgsByGenesT user-defined function

Use the BestHmlgsByGenesT function to search for the best homologous neighbors of an organism from a list of genes that you specify.

The BestHmlgsByGenesT function is a table function that returns a fixed set of output columns.

You can use this function in a SELECT statement.

Syntax:

```
►►──DB2LS.BestHmlgsByGenesT──(keggorg, keggidlist)─────────────────────────────►◄
```

The schema name for this user-defined function is DB2LS.

For better performance, use the BstBstHmlgByGenesT function.

Example of searching for the best homologous neighbors:

To search for the best homologous neighbors of an organism from a list of genes, you specify the *keggorg* and *keggidlist* arguments in the function.

For example, to search for the H. influenzae organism in the list of E. coli genes b0002, b0003, b0004, and b0005, the SELECT statement is:

```
SELECT * FROM TABLE(DB2LS.BestHmlgsByGenesT
    ('hin', 'eco:b0002 eco:b0003 eco:b0004 eco:b0005')) AS t;
```

Related concepts:
- "KEGG user-defined functions - overview" on page 483
- "Columns that are returned from SSDB database queries (table functions)" on page 497

Related tasks:
- "Registering the KEGG user-defined functions" on page 486

Related reference:
- "Function arguments for the KEGG user-defined functions" on page 485
- "KEGG user-defined functions by functional category" on page 483

BstBstHmlgByGenesS user-defined function

Use the BstBstHmlgByGenesS function to search for the best-best homologous neighbors of an organism from a list of genes.

The BstBstHmlgByGenesS function is a scalar function that returns a space-delimited list of target identifiers. The list is a character string with a data type of VARCHAR and an actual length that is not greater than 32767 bytes.

You can use this function in a SELECT statement.

Syntax:

```
►►──DB2LS.BstBstHmlgByGenesS──(keggorg, keggidlist)──────────────────────────►◄
```

The schema name for this user-defined function is DB2LS.

Example of searching for only the best-best neighbors:

To search for only the best-best neighbors of an organism from a list of genes, you specify the *keggorg* and *keggidlist* arguments in the function.

For example, to search for the H. influenzae organism in the list of E. coli genes b0002, b0003, b0004, and b0005, the clause that you use is:

```
VALUES CAST(DB2LS.BstBstHmlgByGenesS
    ('hin', 'eco:b0002 eco:b0003 eco:b0004 eco:b0005')
    AS VARCHAR(1000));
```

Related concepts:
- "KEGG user-defined functions - overview" on page 483

Related tasks:
- "Registering the KEGG user-defined functions" on page 486

Related reference:
- "Function arguments for the KEGG user-defined functions" on page 485
- "KEGG user-defined functions by functional category" on page 483

BstBstHmlgByGenesT user-defined function

Use the BstBstHmlgByGenesT function to search for the best-best homologous neighbors of an organism from a list of genes.

The BstBstHmlgByGenesT function is a table function that returns a fixed set of output columns.

You can use this function in a SELECT statement.

Syntax:

```
►►──DB2LS.BstBstHmlgByGenesT──(keggorg, keggidlist)──────────────────────────►◄
```

The schema name for this user-defined function is DB2LS.

Example of searching for only the best-best neighbors:

To search for only the best-best neighbors of an organism from a list of genes, you specify the *keggorg* and *keggidlist* arguments in the function.

For example, to search for the H. influenzae organism in the list of E. coli genes b0002, b0003, b0004, and b0005, the SELECT statement is:

```
SELECT * FROM TABLE(DB2LS.BstBstHmlgByGenesT
    ('hin', 'eco:b0002 eco:b0003 eco:b0004 eco:b0005')) AS t;
```

Related concepts:

- “KEGG user-defined functions - overview” on page 483
- “Columns that are returned from SSDB database queries (table functions)” on page 497

Related tasks:
- “Registering the KEGG user-defined functions” on page 486

Related reference:
- “Function arguments for the KEGG user-defined functions” on page 485
- “KEGG user-defined functions by functional category” on page 483

ScoreBetweenGenesS user-defined function

Use the ScoreBetweenGenesS function to determine the Smith-Waterman score between two genes.

The ScoreBetweenGenesS function is a scalar function that returns a single value that has a data type of DOUBLE.

You can use this function in a SELECT statement.

Syntax:

```
►►──DB2LS.ScoreBetweenGenesS──(keggid1, keggid2)──────────────────────────────►◄
```

The schema name for this user-defined function is DB2LS.

Example of determining the Smith-Waterman score between two genes:

To determine the Smith-Waterman score between two genes, you specify the *keggid* of each gene.

For example, to determine the Smith-Waterman score between the E. Coli genes b0002 and b3940, the clause that you use is:

```
VALUES CAST(DB2LS.ScoreBetweenGenesS
    ('eco:b0002', 'eco:b3940')
    AS DOUBLE);
```

Related concepts:
- “KEGG user-defined functions - overview” on page 483

Related tasks:
- “Registering the KEGG user-defined functions” on page 486

Related reference:
- “Function arguments for the KEGG user-defined functions” on page 485
- “KEGG user-defined functions by functional category” on page 483

DefinitionsByGeneS user-defined function

Use the DefinitionsByGeneS function to search for the definition of a gene.

The DefinitionsByGeneS function is a scalar function that returns a single value that has a data type of VARCHAR(1000).

You can use this function in a SELECT statement.

Syntax:

```
►►──DB2LS.DefinitionsByGeneS──(keggid)──────────────────────────────────►◄
```

The schema name for this user-defined function is DB2LS.

Example of searching for the definition of a gene:

To search for the definition of a gene, you specify the *keggid*.

For example, to return the definition for the E. coli gene b0002, the clause that you use is:

```
VALUES CAST(DB2LS.DefinitionsByGeneS
    ('eco:b0002')) AS t;
    AS VARCHAR(1000));
```

Related concepts:
- "KEGG user-defined functions - overview" on page 483

Related tasks:
- "Registering the KEGG user-defined functions" on page 486

Related reference:
- "Function arguments for the KEGG user-defined functions" on page 485
- "KEGG user-defined functions by functional category" on page 483

GenesByMotifsT user-defined function

Use the GenesByMotifsT function to search for genes that contain all of the motifs in a list of motifs that you specify.

The GenesByMotifsT function is a table function that returns a table. The table contains the *keggid* VARCHAR(100) and the *definition* VARCHAR(1000) for each gene.

You can use this function in a SELECT statement.

Syntax:

```
►►──DB2LS.GenesByMotifsT──(midlist)─────────────────────────────────────►◄
```

The schema name for this user-defined function is DB2LS.

Example of searching for genes that contain all of the motifs in a list:

To search for genes that contain all of the motifs in a list, you specify the *midlist* argument.

For example, to find the genes that contain both the *Pfam 'DnaJ'* and *Prosite 'DNAJ_2'* motifs, the SELECT statement is:

```
SELECT * FROM TABLE(DB2LS.GenesByMotifsT
    ('pf:DnaJ ps:DNAJ_2')) AS t;
```

Related concepts:

- "KEGG user-defined functions - overview" on page 483

Related tasks:

- "Registering the KEGG user-defined functions" on page 486

Related reference:

- "Function arguments for the KEGG user-defined functions" on page 485
- "KEGG user-defined functions by functional category" on page 483

Disabling the KEGG user-defined functions

If you no longer want to use the KEGG user-defined functions, you can temporarily disable the functions or permanently remove the functions from your federated database.

If you disable the KEGG user-defined functions, you can enable the functions again by registering the functions in the federated database.

You must uninstall the functions to permanently remove the functions from the federated database.

Procedure:

To disable the KEGG user-defined functions, run the disable_KEGGFunctions command.

- On federated servers that run Windows, this command is in the `sqllib\bin` directory
- On federated servers that run UNIX, this command is in the `sqllib/bin` directory

Syntax:

```
disable_KEGGFunctions -n dbName -u userID -p password
```

dbName
: The name of the federated database that you want to disable the functions from.

userID
: A valid user ID for the federated database.

password
: A valid password for the user ID.

Example of disabling the KEGG user-defined functions:

The following example shows the output that is returned when you issue the disable_KEGGFunctions command:

```
C:>disable_KEGGFunctions -n federateddb -u db2admin -p db2admin

(37) KEGG Functions were found
    -- Drop KEGG Functions ...
    Drop KEGG Functions Successfully.


*** Please allow a few seconds to clean up the system ......
```

Related tasks:

- "Removing relational wrappers, nonrelational wrappers, and life sciences user-defined functions (Windows)" in the *IBM DB2 Information Integrator Installation Guide for Linux, UNIX, and Windows*
- "Removing wrappers, user-defined functions, and the wrapper development kits (UNIX)" in the *IBM DB2 Information Integrator Installation Guide for Linux, UNIX, and Windows*
- "Registering the KEGG user-defined functions" on page 486

Part 5. Reference

Chapter 27. Data types supported for nonrelational data sources

The following sections list the data types that are supported by the nonrelational wrappers.

Data types supported for nonrelational data sources

For most of the nonrelational data sources, you must specify the column information, including data type, when you create the nicknames to access the data source.

Some of the nonrelational wrappers create all of the columns required to access a data source. These are called *fixed columns*. Other wrappers let you specify some or all of the data types for the columns in the CREATE NICKNAME statement.

The following sections list the wrappers that you can specify the data types for, and the data types that are supported by the wrapper.

Data types supported by the BioRS wrapper

The following table lists the DB2 data types that are supported by the BioRS wrapper.

Table 127. BioRS data types that map to DB2 data types

BioRS data types	DB2 data type
AUTHOR	CHARACTER, CLOB, VARCHAR
DATE	CHARACTER, CLOB, VARCHAR
NUMBER	CHARACTER, CLOB, VARCHAR
REFERENCE	CHARACTER, CLOB, VARCHAR
TEXT	CHARACTER, CLOB, VARCHAR

The maximum length allowed for the CLOB data type is 5 megabytes.

Data types supported by the BLAST wrapper

Some of the data types are automatically set for the fixed columns that the BLAST wrapper creates.

For the definition line fields, you can assign when you create a nickname. If the data in the definition line column is not compatible with the local column data type, you will get an error. For example, if you define a definition line column of type INTEGER and there are values in the column that are not numeric, an error is returned.

The following table lists the DB2 data types that are supported by the BLAST wrapper.

Table 128. BLAST data types that map to DB2 data types

BLAST data types	DB2 data type
definition line	CLOB The maximum length allowed for the CLOB data type is 5 megabytes.
definition line	DOUBLE
definition line	FLOAT
definition line	INTEGER
definition line	VARCHAR

Data types supported by the Documentum wrapper

The following table lists the DB2 data types that are supported by the Documentum wrapper.

Table 129. Documentum data types that map to DB2 data types

Documentum data types	DB2 data type
DOUBLE	DOUBLE, FLOAT, INTEGER, SMALLINT
ID	CHARACTER (16)
INTEGER	DOUBLE, FLOAT, INTEGER, SMALLINT
STRING (up to 255 characters)	CHAR, VARCHAR
TIME	CHAR, DATE, TIMESTAMP, VARCHAR

Data types supported by the Entrez wrapper

The following table lists the DB2 data types that are supported by the Entrez wrapper.

Table 130. Entrez data types that map to DB2 data types

Entrez data types	DB2 data type
character	CHARACTER
character	CLOB The maximum length allowed for the CLOB data type is 5 megabytes.
date	DATE
number	DECIMAL
number	DOUBLE
integer	INTEGER
number	REAL
integer	SMALLINT
time	TIMESTAMP
character	VARCHAR

Data types supported by the Excel wrapper

The following table lists the DB2 data types that are supported by the Excel wrapper.

Table 131. Excel data types that map to DB2 data types

Excel data types	DB2 data type
date	DATE
number	DOUBLE
number	FLOAT (n) where n is >= 25 and ≤= 53
integer	INTEGER
character	VARCHAR

Data types supported by the Extended Search wrapper

The following table lists the DB2 data types that are supported by the Extended Search wrapper.

Table 132. Extended Search data types that map to DB2 data types

Extended Search data types	DB2 data type
Date	DATE
Double	DOUBLE
Integer	INTEGER
String	VARCHAR

Data types supported by the HMMER wrapper

The following table lists the DB2 data types that are supported by the HMMER wrapper.

Table 133. HMMER data types that map to DB2 data types

HMMER data types	DB2 data type
character	CLOB The maximum length allowed for the CLOB data type is 5 megabytes.
character	DOUBLE
character	FLOAT
character	INTEGER
character	VARCHAR

Data types supported by the table-structured file wrapper

The following table lists the DB2 data types that are supported by the table-structured file wrapper.

Table 134. Table-structured file data types that map to DB2 data types

Table-structured file data types	DB2 data type
character	CHARACTER
character	CLOB The maximum length allowed for the CLOB data type is 5 megabytes.
number	DECIMAL

Table 134. Table-structured file data types that map to DB2 data types (continued)

Table-structured file data types	DB2 data type
number	DOUBLE
number	FLOAT
integer	INTEGER
number	REAL
integer	SMALLINT
character	VARCHAR

Data types supported by the Web services wrapper

The following table lists the DB2 data types that are supported by the Web services wrapper. The Web services wrapper uses XML data types.

Table 135. XML data types that map to DB2 data types for the Web services wrapper

XML data types	DB2 data type
character	CHARACTER
character	CHARACTER FOR BIT DATA
character	CLOB
date	DATE
number	DECIMAL
number	DOUBLE
number	FLOAT
integer	INTEGER
number	REAL
integer	SMALLINT
character	VARCHAR
character	VARCHAR FOR BIT DATA

Data types supported by the WebSphere Business Integration wrapper

The following table lists the DB2 data types that are supported by the WebSphere Business Integration wrapper. The WebSphere Business Integration wrapper uses XML data types.

Table 136. XML data types that map to DB2 data types for the WebSphere Business Integration wrapper

XML data types	DB2 data type
character	CHARACTER
character	CHARACTER FOR BIT DATA
character	CLOB
date	DATE
number	DECIMAL
number	DOUBLE
number	FLOAT

Table 136. XML data types that map to DB2 data types for the WebSphere Business Integration wrapper (continued)

XML data types	DB2 data type
integer	INTEGER
number	REAL
integer	SMALLINT
character	VARCHAR
character	VARCHAR FOR BIT DATA

Data types supported by the XML wrapper

The following table lists the DB2 data types that are supported by the XML wrapper

Table 137. XML data types that map to DB2 data types for the XML wrapper

XML data types	DB2 data type
character	CHARACTER
character	CHARACTER FOR BIT DATA
character	CLOB The maximum length allowed for the CLOB data type is 5 megabytes.
date	DATE
number	DECIMAL
number	DOUBLE
number	FLOAT
integer	INTEGER
number	REAL
integer	SMALLINT
character	VARCHAR
character	VARCHAR FOR BIT DATA

Related concepts:

- "Data type mappings in a federated system" in the *Federated Systems Guide*
- "Data type mappings and the federated database global catalog" in the *Federated Systems Guide*
- "Data type mappings for nonrelational data sources" in the *Federated Systems Guide*

Chapter 28. Altering nicknames

This chapter explains how to alter previously registered nicknames.

Altering a nickname

Nicknames are identifiers that are used to reference an object that you want to access at a data source.

You might want to alter a nickname to:

- Alter the local column names for the columns of the data source object
- Alter the local data types for the columns of the data source object
- Add, set, or drop nickname and column options
- Add or drop a primary key
- Add or drop one or more unique, referential, or check constraints
- Alter one or more referential, check, or functional dependency constraint attributes

Prerequisites:

The privileges held by the authorization ID of the statement must include at least one of the following:

- SYSADM or DBADM authority
- ALTER privilege on the nickname specified in the statement
- CONTROL privilege on the nickname specified in the statement
- ALTERIN privilege on the schema, if the schema name of the nickname exists
- Definer of the nickname as recorded in the DEFINER column of the catalog view for the nickname

Restrictions:

See the topic on restrictions to altering nicknames.

Procedure:

You can alter a nickname from the DB2 Control Center or the DB2 command line.

To do this task from the DB2 Control Center:

1. Select the **Nicknames** folder.
2. Right-click on the nickname that you want to change and click **Alter**. The Alter Nickname notebook opens.
3. On the Nicknames page, change the local column names, local data types, or column options for the columns that are stored in the global catalog.
4. On the Keys page, set the referential integrity constraints for the nickname. You can set a primary key, unique key, or foreign key constraint.
5. On the Check Constraints page, set the check constraints or functional dependency constraints for the nickname.
6. On the Settings page, set the nickname options for the nickname.

7. Click **OK** to alter the nickname and close the notebook.

   Some nickname options are required and cannot be dropped. Other nickname options cannot be added if specific nickname options are already set. See the nickname options for federated systems and the nickname column options for federated systems for a list of descriptions for each of the options.

To do this task from the DB2 command line, issue the ALTER NICKNAME statement with the appropriate parameters set.

When the data source object structure or content changes significantly, you should update the nickname statistics. Significant changes include the addition or removal of multiple rows.

Related concepts:

- "Informational constraints on nicknames" in the *Federated Systems Guide*
- "Nickname statistics update facility - overview" in the *Federated Systems Guide*

Related tasks:

- "Altering nickname options" on page 527
- "Altering a local type for a data source object" on page 530
- "Altering nickname column names" on page 526
- "Altering nickname column options" on page 528

Related reference:

- "Restrictions on altering nicknames" on page 524
- Appendix F, "Nickname options for federated systems," on page 593
- Appendix G, "Nickname column options for federated systems," on page 603
- "ALTER NICKNAME statement" in the *SQL Reference, Volume 2*

Restrictions on altering nicknames

The following restrictions apply when you alter a nickname.

Column names

The ALTER NICKNAME statement cannot be used to alter column names for the following data sources. You must drop the nickname and create the nickname again with the correct column names.

- BLAST
- Documentum
- HMMER

Column options

If one of the following options is set on a column, you cannot add any other options to that column:

- SOAPACTIONCOLUMN
- URLCOLUMN
- PRIMARY_KEY
- FOREIGN_KEY

For BioRS

- If you change the element name of a column by using the ELEMENT_NAME option, the new name is not checked to ensure that it is correct. An incorrect option might result in errors when the column is referenced in a query.
- If you make changes to the IS_INDEXED column option, the changes are not verified with the BioRS server. An incorrect option might result in errors when the column is referenced in a query.

Data types

- If you change the data type of a column, the new data type must be compatible with the data type of the corresponding data source column or element. Changing the local data type to a data type that is incompatible with the remote data type might cause unpredictable errors.
- The *local_data_type* cannot be long VARCHAR, LONG VARGRAPHIC, DATALINK or a user-defined data type.
- The *data_source_data_type* cannot be a user-defined type.
- You cannot override the existing local types or create new local types for some of the nonrelational data sources. Check the documentation for the specific data source wrapper for more information on this restriction.
- When the local specification of a column's data type is changed, the federated database manager invalidates any statistics (for example, HIGH2KEY and LOW2KEY) that are gathered for that column.
- The local type is set for the specific data source object when it is accessed with that nickname. The same data source object can have another nickname that uses the default data type mapping.

Indexes

The ALTER NICKNAME statement cannot be used to register a new data source index in the federated database. Use the CREATE INDEX statement with the SPECIFICATION ONLY clause to create an index specification.

LOCAL NAME and LOCAL TYPE parameters

- ALTER NICKNAME statement cannot be used to change the local names or data types for the columns in the nickname if:
  - The nickname is used in a view, SQL method, or SQL function
  - You define an informational constraint on the nickname
- The federated_column_options clause must be specified last if you also need to specify the LOCAL NAME parameter, the LOCAL TYPE parameter, or both in the ALTER NICKNAME statement..

Nicknames

The ALTER NICKNAME statement cannot be used to change the name of the BioRS databank that is referenced by or used in a BioRS nickname. If the name of a BioRS databank changes, you must drop the nickname and create the nickname again.

Units of work

The federated server cannot process an ALTER NICKNAME statement within a given unit of work under either of the following conditions:

- If the nickname referenced in the ALTER NICKNAME statement has a cursor open on it in the same units of work.
- If an insert, delete or update is issued in the same unit of work for the nickname that is referenced in the ALTER NICKNAME statement.

- For nonrelational data sources, if the ALTER NICKNAME statement references a nickname that is referenced by a SELECT statement in the same unit of work.

Related tasks:

- "Altering nickname options" on page 527
- "Altering a local type for a data source object" on page 530
- "Altering a nickname" on page 523
- "Altering nickname column names" on page 526
- "Altering nickname column options" on page 528

Altering nickname column names

When you create a nickname, the column names that are associated with the data source object are stored in the federated database. For some data sources, the wrapper specifies the column names. For other data sources, you must specify the column names when you create the nickname.

You can alter a nickname to change the column names.

Prerequisites:

The authorization ID issuing the statement must include at least one of the following privileges:

- SYSADM or DBADM authority
- ALTER privilege on the nickname specified in the statement
- CONTROL privilege on the nickname specified in the statement
- ALTERIN privilege on the schema, if the schema name of the nickname exists
- Definer of the nickname as recorded in the DEFINER column of the catalog view for the nickname

Restrictions:

See the topic on restrictions to altering nicknames.

Procedure:

You can change column names from the DB2 Control Center or the DB2 command line.

To do this task from the DB2 Control Center:

1. Select the **Nicknames** folder.
2. Right-click on the nickname that you want to change and click **Alter**. The Alter Nickname notebook opens.
3. On the Nicknames page, select the column that you want to change and click **Change**. The Change Column window opens.
4. Type the column name.
5. Click **OK** to change the column name and close the window.
6. Click **OK** to alter the nickname and close the notebook.

To do this task from the DB2 command line, issue the ALTER NICKNAME statement:

```
ALTER NICKNAME nickname
   ALTER COLUMN current_name
   LOCAL NAME new_name
```

Example: Changing the local name of a nickname column:

For example, the nickname Z_EMPLOYEES for a DB2 UDB for z/OS table includes a column with the name of EMPNO. To alter the nickname so that the local column name that users work with is *Employee_Number* instead of *EMPNO*, issue the following statement:

```
ALTER NICKNAME Z_EMPLOYEES ALTER COLUMN EMPNO
   LOCAL NAME "Employee_Number"
```

Related tasks:

- "Altering a nickname" on page 523

Related reference:

- "Restrictions on altering nicknames" on page 524
- "ALTER NICKNAME statement" in the *SQL Reference, Volume 2*

Altering nickname options

Nickname options are parameters that you specify on the nickname when you issue the CREATE NICKNAME and ALTER NICKNAME statements.

You can add, set, or drop nickname options by using the ALTER NICKNAME statement.

Prerequisites:

The authorization ID issuing the statement must include at least one of the following privileges:

- SYSADM or DBADM authority
- ALTER privilege on the nickname specified in the statement
- CONTROL privilege on the nickname specified in the statement
- ALTERIN privilege on the schema, if the schema name of the nickname exists
- Definer of the nickname as recorded in the DEFINER column of the catalog view for the nickname

Restrictions:

See the topic on restrictions to altering nicknames.

Procedure:

You can change column names from the DB2 Control Center or the DB2 command line.

To do this task from the DB2 Control Center:

1. Select the **Nicknames** folder.

2. Right-click on the nickname that you want to change and click **Alter**. The Alter Nickname notebook opens.
3. On the Settings page, select the check box next to any option that you want to add or remove. You cannot remove a required option.
4. To specify or change the value of an option, click the **Value** field for the option. Depending on the option, you can either select a value from the list, click to select multiple values, or you can type a new value.
5. Click **OK** to alter the nickname and close the notebook.

To do this task from the command line prompt, use the ALTER NICKNAME statement. For example:

```
ALTER NICKNAME nickname
    OPTIONS (SET option_name 'option_string_value')
```

For example, the nickname DRUGDATA1 is created for the table-structured file `drugdata1.txt`. The fully qualified path that was originally defined in the CREATE NICKNAME statement was `/user/pat/drugdata1.txt`.

To change the FILE_PATH nickname option, issue the following statement :

```
ALTER NICKNAME DRUGDATA1 OPTIONS (SET FILE_PATH '/usr/kelly/data/drugdata1.txt')
```

Related tasks:
- "Altering a nickname" on page 523

Related reference:
- "Restrictions on altering nicknames" on page 524
- "ALTER NICKNAME statement" in the *SQL Reference, Volume 2*
- Appendix F, "Nickname options for federated systems," on page 593

Altering nickname column options

You specify column information in the CREATE NICKNAME and ALTER NICKNAME statements by using parameters called *nickname column options.* You can specify any of these values in either uppercase or lowercase letters.

You can add, set, or drop nickname column options using the ALTER NICKNAME statement.

Prerequisites:

The authorization ID issuing the statement must include at least one of the following privileges:
- SYSADM or DBADM authority
- ALTER privilege on the nickname specified in the statement
- CONTROL privilege on the nickname specified in the statement
- ALTERIN privilege on the schema, if the schema name of the nickname exists
- Definer of the nickname as recorded in the DEFINER column of the catalog view for the nickname

Restrictions:

See the topic on restrictions to altering nicknames.

Procedure:

You can change column names from the DB2 Control Center or the DB2 command line.

To do this task from the DB2 Control Center:
1. Select the **Nicknames** folder.
2. Right-click on the nickname that you want to change and click **Alter**. The Alter Nickname notebook opens.
3. On the Nicknames page, select the column that you want to change and click **Change**. The Change Column window opens.
4. Select the column option that you want to add or remove.
5. For options that you are adding or changing, specify the value of an option.
6. Click **OK** to change the column option and close the window.
7. Click **OK** to alter the nickname and close the notebook.

To do this task from the command line prompt, use the ALTER NICKNAME statement.

Example 1: Specifying the NUMERIC_STRING column option with relational data sources:

The NUMERIC_STRING column option applies to character type columns (CHAR and VARCHAR). Suppose that a data source has a collating sequence that differs from the federated database collating sequence. The federated server typically would not sort any columns containing character data at the data source. It would return the data to the federated database and perform the sort locally. However, suppose that the column is a character data type and contains only numeric characters (’0’,’1’,...,’9’). You can indicate this by assigning a value of ’Y’ to the NUMERIC_STRING column option. This gives the DB2 UDB query optimizer the option of performing the sort at the data source. If the sort is performed remotely, you can avoid the overhead of sorting the data at the federated server.

The nickname ORA_INDSALES for an Oracle table called INDONESIA_SALES. The table contains the column POSTAL_CODE with the data type of VARCHAR. Originally the column contained only numeric characters, and the NUMERIC_STRING column option was set to ’Y’. However, the column now contains a mixture of numeric and non-numeric characters. To change the NUMERIC_STRING column option to ’N’, use this statement:

```
ALTER NICKNAME ORA_INDSALES ALTER COLUMN POSTAL_CODE
   OPTIONS (SET NUMERIC_STRING 'N')
```

Example 2: Specfying the VARCHAR_NO_TRAILING_BLANKS column option with relational data sources:

The VARCHAR_NO_TRAILING_BLANKS column option can be used to identity specific columns that contain no trailing blanks. The SQL Compiler will factor in this setting when it checks for all operations (such as comparison operations) performed on columns.

The nickname ORA_INDSALES is for an Oracle table called INDONESIA_SALES. The table contains the column NAME with the data type of VARCHAR. The NAME column does not have trailing blanks. To add the VARCHAR_NO_TRAILING_BLANKS option to the nickname, use this statement:

```
ALTER NICKNAME ORA_INDSALES ALTER COLUMN NAME
   OPTIONS (ADD VARCHAR_NO_TRAILING_BLANKS 'Y')
```

Example 3: Specifying the XPATH column option with nonrelational data sources:

The nickname EMPLOYEE is for an XML data source. An XPATH was specified for the *fname* column. To set the XPATH column option to a different path, use this statement:

```
ALTER NICKNAME EMPLOYEE ALTER COLUMN fname
   OPTIONS (SET XPATH './@first')
```

Related tasks:

- "Altering a nickname" on page 523

Related reference:

- "Restrictions on altering nicknames" on page 524
- "ALTER NICKNAME statement" in the *SQL Reference, Volume 2*
- Appendix G, "Nickname column options for federated systems," on page 603

Altering a local type for a data source object

| When you create a nickname, the data types that are associated with the data
| source object are stored in the federated database. For some data sources, the
| wrapper specifies the data types for you. For other data sources, you must specify
| the data types when you create the nickname.

You can specify a local type for a column of a specific data source object. You use the ALTER NICKNAME statement instead of the CREATE TYPE MAPPING statement.

| **Attention**: Changing the local data type can result in errors or loss of information
| if you change the local data type for a column to a type that differs greatly from its
| remote type.

Prerequisites:

The authorization ID issuing the statement must include at least one of the following privileges:

- SYSADM or DBADM authority
- ALTER privilege on the nickname specified in the statement
- CONTROL privilege on the nickname specified in the statement
- ALTERIN privilege on the schema, if the schema name of the nickname exists

The authorization ID associated with the statement must be the definer of the nickname as recorded in the DEFINER column of the catalog view for the nickname.

Restrictions:

See the topic on restrictions to altering nicknames.

Procedure:

You can change the data type from the DB2 Control Center or the DB2 command line.

To do this task from the DB2 Control Center:

1. Select the **Nicknames** folder.
2. Right-click on the nickname that you want to change and click **Alter**. The Alter Nickname notebook opens.
3. On the Nicknames page, select the column that you want to change and click **Change**. The Change Column window opens.
4. Select the data type.
5. Click **OK** to change the data type and close the window.
6. Click **OK** to alter the nickname and close the notebook.

To do this task from the command line prompt, use the ALTER NICKNAME statement. For example:

```
ALTER NICKNAME nickname ALTER COLUMN column_name
   LOCAL TYPE data_type
```

To treat the contents of a local column that has a character data type as bit (binary) data, use the FOR BIT DATA clause in the ALTER NICKNAME statement. When you use this clause to change the local data type of a column, code page conversions are not performed when data is exchanged with other systems. Comparisons are done in binary, irrespective of the remote database collating sequence.

Related tasks:

- "Altering a nickname" on page 523

Related reference:

- "Restrictions on altering nicknames" on page 524
- "ALTER NICKNAME statement" in the *SQL Reference, Volume 2*
- Appendix H, "Default forward data type mappings," on page 611
- "Altering a local type for a data source object – examples" on page 531
- "Data types supported for nonrelational data sources" on page 517

Altering a local type for a data source object – examples

The following examples show how to change the data types for a data source object.

Example: A numeric data type mapping:

In an Oracle table for employee information, the BONUS column is defined with a data type of NUMBER(32,3). The Oracle data type NUMBER(32,3) is mapped by default to the DB2 data type DOUBLE, a double-precision floating-point number data type. A query that includes the BONUS column might return values that look like this:

```
5.0000000000000E+002
1.0000000000000E+003
```

The scientific notation indicates the number of decimal places and the direction that the decimal point should be moved. In this example +002 signifies that the decimal point should be moved two places to the right, and +003 signifies that the decimal point should be moved three places to the right.

Queries that include the BONUS column can return values that look like dollar amounts. You change the local definition for the BONUS column in the table from the DOUBLE data type to DECIMAL data type. Use a precision and scale that reflect the format of actual bonuses. For example, if the dollar portion of the bonuses would not exceed six figures, map NUMBER(32,3) to DECIMAL(8,2). Under the constraint of this new local type, queries that include the BONUS column return values like this:

```
 500.00
1000.00
```

The nickname for the Oracle table is ORASALES. To map the BONUS column in the ORASALES table to the DB2 DECIMAL (8,2) data type, issue the following ALTER NICKNAME statement:

```
ALTER NICKNAME ORASALES ALTER COLUMN BONUS
   LOCAL TYPE DECIMAL(8,2)
```

ORASALES
: The nickname that you defined for the Oracle table.

ALTER COLUMN *BONUS*
: The name of the column that is defined locally in the federated database SYSCAT.COLUMNS catalog view.

LOCAL TYPE *DECIMAL(8,2)*
: Identifies the new local type for the column.

This mapping applies only to the BONUS column in the Oracle table that is identified by the nickname ORASALES. All other Oracle data source objects that include the BONUS column use the default data type mapping for the Oracle NUMBER data type.

Example: A date data type mapping:

The nickname for an Oracle table named SALES is ORASALES. The SALES table contains a column that is the Oracle DATE data type. The default type mapping for the Oracle DATE data type is to the DB2 TIMESTAMP data type. However, you want to display only the date value when you retrieve data from this column. You can alter the nickname for the SALES table to change the local type to the DB2 DATE data type.

```
ALTER NICKNAME ORASALES ALTER COLUMN ORDER_DATE
   LOCAL TYPE DATE
```

Example: A data type mapping for a nonrelational data source:

The nickname for a table-structured file named `drugdata1.txt` is DRUGDATA1. The `drugdata1.txt` file contains a column that lists pharmaceutical drug names. The column name is DRUG. The DRUG column was originally defined as a CHAR(20). The length of the column must be changed to CHAR(30). You can alter the nickname for the `drugdata1.txt` file to change the mapping to the correct length:

```
ALTER NICKNAME DRUGDATA1 ALTER COLUMN DRUG
   LOCAL TYPE CHAR(30)
```

Related tasks:

- “Creating data type mappings” in the *Federated Systems Guide*
- “Altering a local type for a data source object” on page 530

Related reference:
- “ALTER NICKNAME statement” in the *SQL Reference, Volume 2*
- “Restrictions on altering nicknames” on page 524

Chapter 29. DDL command reference

This chapter provides details of the syntax statements, arguments, and options for the wrapper DDL commands that are discussed in this book. The statements are ordered by wrapper.

BioRS DDL reference information

CREATE SERVER statement options - BioRS wrapper

Options for the CREATE SERVER statement for BioRS are:

TYPE Specifies the server type. The default value is `BioRS`. The default value is the only value that is supported for the BioRS wrapper. You do not need to specify this option.

VERSION
Specifies the server version. The default value is `1.0`. The default value is the only value that is supported for the BioRS wrapper You do not need to specify this option.

NODE
Specifies the host name of the system on which the BioRS query tool is available. The default value is *localhost*.

PORT Specifies the number of the port to be used to connect to the BioRS server. The default value is 5014.

TIMEOUT
Specifies the time, in minutes, that the BioRS wrapper should wait for a response from the BioRS server. The default value is 10. You must specify this option.

CASE_SENSITIVE
Specifies whether the BioRS server treats names in a case sensitive manner. Valid values are ’Y’ or ’N’. The default value is ’Y’.

In the BioRS product, a configuration parameter controls the case sensitivity of the data that is stored on the BioRS server machine. The CASE_SENSITIVE option is the DB2 Information Integrator counterpart to that BioRS system configuration parameter. You must synchronize the BioRS server case sensitivity configuration settings in your BioRS system and in DB2 Information Integrator. If you do not keep the case sensitivity configuration settings synchronized between BioRS and DB2 Information Integrator, errors will occur when you attempt to access BioRS data through DB2 Information Integrator.

Important: You cannot change or delete the CASE_SENSITIVE option after you create a new BioRS server in DB2 Information Integrator. If you need to change the CASE_SENSITIVE option, you must drop and then create the entire server again. If you drop the BioRS server, you must also create all of the corresponding BioRS nicknames again. DB2 Information Integrator automatically drops all nicknames that correspond to a dropped server.

Related tasks:

- “Registering the server definition for a BioRS data source” on page 72

- “Registering nicknames for BioRS data sources” on page 74

Related reference:
- “CREATE SERVER statement” in the *SQL Reference, Volume 2*
- “CREATE NICKNAME statement syntax - BioRS wrapper” on page 536

CREATE USER MAPPING statement options - BioRS wrapper

GUEST
: Specifies whether operations are to be performed under the BioRS guest authentication mechanism on the BioRS server. Valid values are ’Y’ or ’N’. The default value is ’Y’.

  If this option is set to ’Y’, then guest authentication is used to access the BioRS server for this DB2 Information Integrator user.

  If this option is set to ’N’, then a BioRS authorization ID and password must be provided to access the BioRS server for this DB2 Information Integrator user.

  If no user mapping is created, or if a user mapping is created with no options specified, then guest authentication is used to access the BioRS server for the DB2 Information Integrator user.

REMOTE_AUTHID
: Specifies a user ID that allows this DB2 user to access BioRS data sources. This remote ID must be in the format that is expected by the BioRS application. This option is required if the GUEST option is set to ’N’.

REMOTE_PASSWORD
: Specifies the password for this remote ID. This option is required if the GUEST option is set to ’N’.

Example:

The following CREATE USER MAPPING statement maps user `Charlie` to user `Charlene` on the `Biors_Server1` server.

```
CREATE USER MAPPING FOR Charlie SERVER Biors_Server1
    OPTIONS(GUEST 'N', REMOTE_AUTHID 'Charlene', REMOTE_PASSWORD 'Charlene_pw');
```

Related tasks:
- “Registering user mappings for BioRS data sources” on page 73

Related reference:
- “CREATE USER MAPPING statement” in the *SQL Reference, Volume 2*

CREATE NICKNAME statement syntax - BioRS wrapper

The syntax for the CREATE NICKNAME statement is:

```
                                       ┌─,──────────────────────────────────────┐
                                       ▼                                        │
►►──CREATE NICKNAME──nickname──(───column-name──┤ column-information ├──┴──)──────►
```

```
►─FOR SERVER──server-name──OPTIONS──(───────────────────────────────────────────────►


►─┬───────────────────────────────────────┬─)───────────────────────────────────►◄
  └─REMOTE_OBJECT──'BioRS_databank_name' ─┘
```

column-information:

```
├──┤ data-type ├──┤ nickname-column-options ├──────────────────────────────────────┤
```

data-type:

```
├───┬─┬─CLOB──────────────────────────┬──┬────────────────────────┬─┬────────────┤
    │ ├─┬─CHARACTER─┬──LARGE OBJECT───┘  └─(──integer──┬───────┬─)─┘ │
    │ │ └─CHAR──────┘                                  ├─K─────┤     │
    │ │                                                ├─M─────┤     │
    │ │                                                └─G─────┘     │
    ├─┬─CHARACTER─┬──┬────────────────┬──────────────────────────────┤
    │ └─CHAR──────┘  └─(──integer──)──┘                              │
    └─VARCHAR──(──integer──)─────────────────────────────────────────┘
```

nickname-column-options:

```
├──OPTIONS──(──┬─────────────────────────────────────────┬──────────────────────────►
               └─ELEMENT_NAME──'BioRS_element_name' ──,──┘

►─IS_INDEXED──┬─'Y' ─┬──,──REFERENCED_OBJECT──'BioRS_databank_name' ──)──────────┤
              └─'N' ─┘
```

Nickname column options

Nickname column option values must be enclosed in single quotation marks.

ELEMENT_NAME
: Specifies the BioRS element name. The case sensitivity of this name depends on the case sensitivity of the BioRS server and on the value of the CASE_SENSITIVE server option. You need to specify the BioRS element name only if it is different from the column name.

IS_INDEXED
: Indicates whether the corresponding column is indexed (whether the column can be referenced in a predicate). The valid values are 'Y' and 'N'. The value 'Y' can be specified only for columns whose corresponding element is indexed by the BioRS server.

  When a nickname is created, this option is automatically added with the value 'Y' to any columns that correspond to a BioRS indexed element.

REFERENCED_OBJECT
: This option is valid only for columns whose BioRS data type is Reference. This option specifies the name of the BioRS databank that is referenced by

the current column. The case sensitivity of this name depends on the case-sensitivity of the BioRS server and on the value of the CASE_SENSITIVE server option.

Nickname options

Nickname option values must be enclosed in single quotation marks.

REMOTE_OBJECT
Specifies the name of the BioRS databank that is associated with the nickname. This name determines the schema and the BioRS databank for the nickname. This name also specifies the relationship of the nickname to other nicknames. The case sensitivity of this name depends on the case sensitivity of the BioRS server and on the value of the CASE_SENSITIVE server option.

Important: You cannot change or delete this name with the ALTER NICKNAME statement. If the name of the BioRS databank that is used in this option changes, you must delete and then create the entire nickname again.

Related tasks:
- "Registering nicknames for BioRS data sources" on page 74

Related reference:
- "CREATE NICKNAME statement" in the *SQL Reference, Volume 2*
- "CREATE NICKNAME statement - Examples for BioRS wrapper" on page 75

BLAST DDL reference information

CREATE SERVER statement arguments - BLAST wrapper

CREATE SERVER arguments for the BLAST wrapper:

TYPE Determines the type of BLAST search performed using the given server. This argument is required. It must be set to one of the following values: blastn, blastp, blastx, tblastn, tblastx.

VERSION
Specifies the version of the server that you are using. It should be set to the version of blastall that you are running. This argument is required.

WRAPPER
Specifies the name of the wrapper that you registered using the CREATE WRAPPER statement. This argument is required.

Server options for the BLAST wrapper:

The options for BLAST that you can specify in the CREATE SERVER statement are:
- DAEMON_PORT
- NODE
- USE_CLOB_SEQUENCE

Related tasks:
- "Registering the server for a BLAST data source" on page 106

Related reference:

- "CREATE SERVER statement" in the *SQL Reference, Volume 2*
- Appendix D, "Server options for federated systems," on page 575

CREATE NICKNAME statement syntax - BLAST wrapper

The syntax for the CREATE NICKNAME statement is:

```
                                    ┌─,─────────────────────────────────────────────┐
                                    ▼                                               │
►►──CREATE NICKNAME──nickname──(────column-name──┤ column-information ├──┴──)──────►

►──FOR SERVER──server-name──OPTIONS──(──DATASOURCE──'data_source_name' ────────────►

►──┬─────────────────────────────────────┬──┬───────────────────────────────────┬──)──►◄
   └─,──PROCESSORS──'processor_number' ──┘  └─,──TIMEOUT──'timeout_duration' ──┘
```

column-information:

```
├──┤ data-type ├──┤ column-option ├──┤ nickname-column-options ├──────────────┤
```

data-type:

```
├──┬─┬─INTEGER─┬──────────────────────────────────────────────────┬──────────┤
   │ └─INT─────┘                                                  │
   ├─┬─FLOAT──┬─────────────────┬─┬────────────────────────────────┤
   │ │        └─(──integer──)───┘ │                                │
   │ │        ┌─PRECISION─┐       │                                │
   │ └─DOUBLE─┴───────────┴───────┘                                │
   └──┬─┬─CLOB *─────────────────────────┬──┬────────────────────┬─┬┘
      │ └─┬─CHARACTER─┬──LARGE OBJECT────┘  └─(──integer──┬───┬─)┘ │
      │   └─CHAR──────┘                                   ├─K─┤    │
      │                                                   ├─M─┤    │
      │                                                   └─G─┘    │
      ├─┬─CHARACTER─┬──┬─────────────────┬─────────────────────────┤
      │ └─CHAR──────┘  └─(──integer──)───┘                         │
      └─VARCHAR──(──integer──)─────────────────────────────────────┘
```

column-option:

```
├──┬──────────┬──────────────────────────────────────────────────────────────┤
   └─NOT NULL─┘
```

nickname-column-options:

```
├──OPTIONS──(──INDEX──'index_number' ──,──DELIMITER──'delimiter' ────────────►
►─┬──────────────────────────────┬─)──────────────────────────────────┤
  └─DEFAULT──'new_default_value'─┘
```

Restriction: The length of a CLOB is limited to 5 megabytes (5MB) for the BLAST wrapper.

There are 2 types of options that you can specify in the CREATE NICKNAME statement for the BLAST data sources:
- Nickname column options
- Nickname options

Related tasks:
- "Registering nicknames for BLAST data sources" on page 107

Related reference:
- "CREATE NICKNAME statement" in the *SQL Reference, Volume 2*
- Appendix G, "Nickname column options for federated systems," on page 603
- Appendix F, "Nickname options for federated systems," on page 593
- "CREATE NICKNAME statement - Examples for BLAST wrapper" on page 112

Documentum DDL reference information

CREATE SERVER statement arguments and options - Documentum wrapper

Arguments associated with the CREATE SERVER statement for Documentum are:

TYPE Specifies the type of the data source. For Documentum, the type is DCTM. This argument is required.

VERSION
Specifies the version of the data source. For EDMS98, the value is '3'. For 4i, the value is '4'. This argument is required.

WRAPPER
Specifies the name of the wrapper associated with this server. This argument is required.

Options associated with the CREATE SERVER statement for Documentum are:

CONTENT_DIR
Specifies the name of the locally-accessible root directory for storing content files retrieved by the GET_FILE, GET_FILE_DEL, GET_RENDITION, and GET_RENDITION_DEL pseudo columns. It must be writable by all users who can use these pseudo columns. Its default value is `/tmp`. This option is optional.

NODE
Specifies the actual name of the Documentum Docbase. This option is required.

OS_TYPE
Specifies the Docbase server's operating system. Valid values are AIX, SOLARIS, and WINDOWS. This option is required.

RDBMS_TYPE
: Specifies the RDBMS used by the Docbase. Valid values are DB2, INFORMIX, ORACLE, SQLSERVER or SYBASE. This option is required.

TRANSACTIONS
: Specifies the server transaction mode. The valid values are:
  - NONE — no transactions are enabled.
  - QUERY — transactions are enabled only for Dctm_Query methods.
  - ALL — transactions are enabled for the Dctm_Query method. ALL has the same function as QUERY in this release.

  The default is QUERY. This option is optional.

Related tasks:
- "Registering the server for Documentum data sources" on page 175

Related reference:
- "CREATE SERVER statement" in the *SQL Reference, Volume 2*

CREATE USER MAPPING statement options - Documentum wrapper

Option definitions:

REMOTE_AUTHID
: Authorization identifier for you at the remote server.

REMOTE_PASSWORD
: Password for you at the remote server.

REMOTE_DOMAIN
: Windows networking domain for you at the remote server. Valid only for Windows platforms.

Related tasks:
- "Registering user mappings for Documentum data sources" on page 176

Related reference:
- "CREATE USER MAPPING statement" in the *SQL Reference, Volume 2*

CREATE NICKNAME statement syntax - Documentum wrapper

The syntax for the CREATE NICKNAME statement for Documentum is:

```
                                    ┌─,──────────────────────────────────────┐
                                    ▼                                        │
►►──CREATE NICKNAME──nickname──(────column-name──┤ column-information ├──────┴──)──────►

►─FOR SERVER──server-name──OPTIONS──(──┬─────────────────────────────┬──────────────────►
                                       └─ALL_VERSIONS──┬─'Y'─┬──,────┘
                                                       └─'N'─┘
►─┬───────────────────────────┬──┬────────────────────────────┬──────────────►
  └─FOLDERS─'folder_string'─,─┘  └─IS_REG_TABLE─┬─'Y'─┬─,────┘
                                                └─'N'─┘

►─REMOTE_OBJECT─'remote_object_type'─)──────────────────────────────────────►◄
```

column-information:

```
├──┤ data-type ├──┤ column-option ├──┬──────────────────────────────┬──────────┤
                                     └──┤ nickname-column-options ├──┘
```

data-type:

```
├──┬─SMALLINT─────────────────────────┬────────────────────────────────────────┤
   ├─┬─INTEGER─┬──────────────────────┤
   │ └─INT─────┘                      │
   ├─DOUBLE─┬───────────┬─────────────┤
   │        └─PRECISION─┘             │
   ├─┬─CHARACTER─┬──┬───────────────┬─┤
   │ └─CHAR──────┘  └─(─integer─)───┘ │
   ├─VARCHAR─(─integer─)──────────────┤
   ├─DATE─────────────────────────────┤
   └─TIMESTAMP────────────────────────┘
```

column-option:

```
├──┬──────────┬────────────────────────────────────────────────────────────────┤
   └─NOT NULL─┘
```

nickname-column-options:

```
├──OPTIONS──(──┬──────────────────────────────────┬──────────────────────────►
               └─REMOTE_NAME─'attribute_name'─,───┘

►─┬────────────────────────┬──┬────────────────────────────┬────────────────►
  └─DELIMITER─'delimiter'─,┘  └─IS_REPEATING─┬─'Y'─┬──,────┘
                                             └─'N'─┘

►─┬─────────────────────────┬──────────────────────────────────────────────────┤
  └─ALL_VALUES─┬─'Y'─┬──────┘
               └─'N'─┘
```

Column options associated with the CREATE NICKNAME statement for Documentum are:

NOT NULL
: All single-valued columns except those defined as TIMESTAMP and DATE must be defined as NOT NULL. Repeating attributes must not be defined as NOT NULL in nicknames.

Nickname column options associated with the CREATE NICKNAME statement for Documentum are:

Nickname column option values must be enclosed in single quotation marks.

ALL_VALUES
: Specifies that all values of a repeating attribute will be returned, separated by the specified delimiter. If this option is missing or is ′N′, then only the last value of a repeating attribute will be returned. As noted under DELIMITER, ALL_VALUES can only be specified for VARCHAR columns for which the IS_REPEATING option is ′Y′ (and is invalid when IS_REG_TABLE = ′Y′).

DELIMITER
: Specifies the delimiter string to be used when concatenating multiple values of a repeating attribute. The delimiter can be one or more characters. The default delimiter is a comma. This option is only valid for attributes of objects with data type VARCHAR where the IS_REPEATING option is set to ′Y′. This option is optional.

IS_REPEATING
: Indicates if the column is multi-valued. Valid values are ′Y′ and ′N′. The default is ′N′. This option is optional.

  Only the last value is returned for

  - non-VARCHAR repeating attributes
  - VARCHAR columns when ALL_VALUES ′N′ is specified

  To overcome this limitation, you can create a dual definition for the repeating attribute column.

REMOTE_NAME
: Specifies the name of the corresponding Documentum attribute or column. This option maps remote attribute or column names to local DB2 column names. It defaults to the DB2 column name. This option is optional.

Nickname column options associated with the CREATE NICKNAME statement for Documentum are:

Nickname option values must be enclosed in single quotation marks.

ALL_VERSIONS
: Specifies whether all object versions will be searched. The valid values are ′y′, ′Y′, ′n′, and ′N′. The default value of ′N′ means that only the current object versions are included in query processing. This option is invalid when IS_REG_TABLE = ′Y′. This option is optional.

FOLDERS
: Specifies a string that contains one or more logically-combined and syntactically-correct Documentum FOLDER predicates. Specifying FOLDER predicates restricts the set of documents represented by this nickname to those in the designated folders.

  When you specify this option, enclose the entire value of the FOLDERS option in single quotes and use double quotes in place of the single quotes within the string.

  For example, if you want to insert:

```
FOLDER('/Tools',DESCEND) OR FOLDER('/Cars')
```

Specify the following FOLDERS option:

```
FOLDERS 'FOLDER("/Tools",DESCEND) OR FOLDER("/Cars")'
```

This option is invalid when IS_REG_TABLE = ’Y’. This option is optional.

IS_REG_TABLE
Specifies whether the object specified by the REMOTE_OBJECT option is a Documentum registered table. The valid values are ’y’, ’Y’, ’n’, and ’N’. The default value is ’N’. This option is optional.

You cannot change a nickname from a Documentum object to a registered table (or back) by changing this option with the ALTER NICKNAME statement. Instead, you must DROP and re-CREATE the nickname.

REMOTE_OBJECT
Specifies the name of the Documentum object type associated with the nickname. The name can be any Documentum object type or registered table. In the case of a registered table, it should be prefixed by the table owner’s name. If the registered table belongs to the Docbase owner, dm_dbo can be used for the owner name. This option is required.

Using ALTER NICKNAME to change the value of the REMOTE_OBJECT option will result in errors if the structure of the new object is not similar to that of the original object.

Related tasks:
- “Registering nicknames for Documentum data sources” on page 176

Related reference:
- “CREATE NICKNAME statement” in the *SQL Reference, Volume 2*
- “CREATE NICKNAME statement - Example for Documentum wrapper” on page 180

Excel DDL reference information

CREATE SERVER statement arguments - Excel wrapper

Arguments associated with the CREATE SERVER statement for Excel are:

WRAPPER
Specifies the name of the wrapper that you registered in the associated CREATE WRAPPER statement. This argument is required.

Related tasks:
- “Registering the server for an Excel data source” on page 220

Related reference:
- “CREATE SERVER statement” in the *SQL Reference, Volume 2*

CREATE NICKNAME statement syntax - Excel wrapper

```
►►──CREATE NICKNAME──nickname──(──────────────────────────────────────────►
   ┌─,──────────────────────────────────────────────────┐
   ▼                                                    │
►──column-name──┤ data-type ├──┤ column-option ├────────┴──)──────────────────►

►─FOR SERVER──server-name──OPTIONS──────────────────────────────────────────────►


►─(──FILE_PATH──'path' ──┬───────────────────────┬──)───────────────────────────►◄
                         └─,──RANGE──'range' ────┘
```

data-type:

```
├──┬──┬─INTEGER─┬─────────┬─────────────────────────────────────────┤
   │  └─INT─────┘         │
   ├─FLOAT──┬─────────────┤
   │        └─(──integer──)┘
   ├─VARCHAR──(──integer──)─┤
   └─DATE───────────────────┘
```

column-option:

```
├──┬──────────┬──────────────────────────────────────────────────────┤
   └─NOT NULL─┘
```

Where:

FOR SERVER
: Identifies the server that you registered in the associated CREATE SERVER statement. This server is used to access the Excel spreadsheet. Specify the server name.

The following list describes the CREATE NICKNAME options for Excel:

FILE_PATH
: Specifies the fully qualified directory path and file name of the Excel spreadsheet that you want to access.

  Data types must be consistent within each column and the column data types must be described correctly during the register nickname process.

  The Excel wrappers can only access the primary spreadsheet within an Excel workbook.

  Blank cells in the spreadsheet are interpreted as NULL.

  Up to 10 consecutive blank rows can exist in the spreadsheet and be included in the data set. More than 10 consecutive blank rows are interpreted as the end of the data set.

  Blank columns can exist in the spreadsheet. However, these columns must be registered and described as valid fields even if they will not be used.

  The database codepage must match the file's character set; otherwise, you could get unexpected results.

RANGE

Specifies a range of cells to be used in the data source. This option is not required.

Any syntax or semantic error in the range option value results in an SQL1882E message. Errors might include:

- The top left and bottom right indicators are not oriented correctly. An incorrect orientation is one in which the top-left cell indicator is either below or to the right of the bottom-right cell indicator.
- The number of columns designated by the range value does not correspond to the number of columns specified in the CREATE NICKNAME statement.
- A nonvalid character or other syntax error has been found.

Here is an example of the RANGE nickname option:

```
CREATE NICKNAME excel2
(c1  VARCHAR (10),
c2 VARCHAR (10),
c3  VARCHAR (10),
c4  VARCHAR (10)
)  FOR SERVER excel_server
OPTIONS (FILE_PATH 'C:\My Documents\test2.xls',
 RANGE 'B2:E5');
```

In this example, **B2** represents the top left of a cell range, and **E5** represents the bottom right of the cell range. The letter *B* in the B2 designation is the column designation. The number *2* in the B2 representation is the row number.

The bottom right designation can be omitted from the range. In this case, the bottom right valid row is used. If the top left value is omitted, then the value is taken as *A1*. If the range specifies more rows than are actually in the spreadsheet, then the actual number of rows is used.

Related tasks:

- "Registering nicknames for Excel data sources" on page 221

Related reference:

- "CREATE NICKNAME statement" in the *SQL Reference, Volume 2*

Extended Search DDL reference information

CREATE WRAPPER statement syntax - Extended Search wrapper

```
►►──CREATE WRAPPER──wrapper-name──LIBRARY──'library-name'──────────────────►◄
```

WRAPPER

Specifies a unique name for this Extended Search wrapper.

LIBRARY

Specifies one of the following platform-dependent library names:

- Windows: `db2uies.dll`
- AIX: `libdb2uies.a`

Related tasks:
- "Registering the Extended Search wrapper" on page 236

Related reference:
- "CREATE WRAPPER statement" in the *SQL Reference, Volume 2*

CREATE SERVER statement syntax - Extended Search wrapper

```
►►──CREATE SERVER──server-name──WRAPPER──wrapper-name──OPTIONS──(──────────────►

►─ES_HOST──'host-name' ──,─────────────────────────────────────────────────────►
                          └ES_PORT──'port-number' ──,┘

►──┬─────────────────────────────┬──┬─────────────────────────────────────────┬─►
   │              ┌─'OFF'─┐      │  │                   ┌─────────┐           │
   └ES_TRACING────┼───────┼──,───┘  │                   ▼  ┌─C─┐  │           │
                  └─'ON'──┘         └ES_TRACELEVEL──'────┼───┼──┴──' ──,─────┘
                                                         ├─N─┤
                                                         ├─W─┤
                                                         └─I─┘

►──┬──────────────────────────────────────────────────────────────┬──)─────────►◄
   │                  ┌─'$INSTHOME/sqllib/log/ESWrapper.log' ─┐   │
   │                  ├─'%DB2TEMPDIR%\ESWrapper.log' ─────────┤   │
   └ES_TRACEFILENAME──┼───────────────────────────────────────┼───┘
                      └─'path' ───────────────────────────────┘
```

SERVER
Specifies a unique name for this server definition. This parameter is required.

WRAPPER
Specifies the name of a previously registered Extended Search wrapper that you want to use with this server definition. This parameter is required.

ES_HOST
Specifies the fully qualified host name or IP address of the Extended Search server that you want to search. This option is required.

ES_PORT
Specifies the port number where this Extended Search server listens for requests. If you omit this option, the default value is 6001.

ES_TRACING
Specifies whether tracing should be enabled for error messages, warning messages, and informational messages that are produced by the remote Extended Search server. The default value, OFF, means that no trace messages will be logged.

ES_TRACELEVEL
If tracing is enabled, this option specifies the types of messages that will be written to the log file. The default value, C, logs only critical messages. You can enable and disable the following trace levels independently:

C – Critical error messages
N – Noncritical messages
W – Warning messages
I – Informational messages

For example:

```
ES_TRACELEVEL 'W'
ES_TRACELEVEL 'CN'
ES_TRACELEVEL 'CNWI'
```

ES_TRACEFILENAME
If tracing is enabled, this option specifies the name of a directory and file where messages will be written. If you omit this option:
- On UNIX, the default value is `$INSTHOME/sqllib/log/ESWrapper.log`
- On Windows, the default value is `%DB2TEMPDIR%\ESWrapper.log`

Related tasks:
- "Registering the server for Extended Search data sources" on page 237

Related reference:
- "CREATE SERVER statement" in the *SQL Reference, Volume 2*

CREATE USER MAPPING statement syntax - Extended Search wrapper

```
►►──CREATE USER MAPPING FOR──authorization-name──SERVER──server-name──OPTIONS──────►

►─(──REMOTE_AUTHID──'user-id' ──,──REMOTE_PASSWORD──'password' ──)────────────────►◄
```

FOR
Specifies the user ID of a DB2 user that you want to authorize to access Extended Search data sources. This parameter is required.

SERVER
Specifies the name of a previously registered server definition that was created for the Extended Search server that the user wants to search. This parameter is required.

REMOTE_AUTHID
Specifies a user ID that allows this DB2 user to access Extended Search data sources. This remote ID must be in the format that is expected by the data source that is being searched. This option is required.

REMOTE_PASSWORD
Specifies the password for this remote ID. This option is required.

Related tasks:
- "Registering user mappings for Extended Search data sources" on page 238

Related reference:
- "CREATE USER MAPPING statement" in the *SQL Reference, Volume 2*

CREATE NICKNAME statement syntax - Extended Search wrapper

```
                                  ┌─,──────────────────────────┐
                                  ▼                            │
►►──CREATE NICKNAME──nickname──(──┴─column-name──┤ data-type ├─┴──)──────────────►
```

```
►─FOR SERVER──server-name──OPTIONS──(──APPLICATIONID──'application-id' ──,──────────►

►──┬─CATEGORY──'category-name' ──,─┬──┬──────────────────────────────────┬────────────►
   └─DATASOURCE──'source-name' ──,─┘  │                  ┌─'NO' ──┐       │
                                      └─VERTICAL_TABLE───┼────────┼──,────┘
                                                         └─'YES' ─┘

►──┬──────────────────────────────┬──┬──────────────────────────────┬─────────────────►
   │          ┌─'30' ──────┐      │  │        ┌─'50' ──────┐        │
   └─TIMEOUT──┼────────────┼──,───┘  └─MAXHIT─┼────────────┼──,─────┘
              └─'timeout' ─┘                  └─'results' ─┘

►──┬───────────────────────────────────────┬──┬────────────────────────┬──────────────►
   │              ┌─'50' ────────────┐     │  │           ┌─'A' ─┐     │
   └─TOTALMAXHIT──┼──────────────────┼──,──┘  └─SORTORDER─┼──────┼──,──┘
                  └─'total-results' ─┘                    └─'D' ─┘

►──┬───────────────────────────────┬─)──────────────────────────────────────────────►◄
   │            ┌─'DOC_RANK' ───┐  │
   └─SORTFIELD──┼───────────────┼──┘
                └─'field-name' ─┘
```

data-type:

```
├──┬─SMALLINT─────────────────┬──────────────────────────────────────────────────────┤
   ├──┬─INTEGER─┬─────────────┤
   │  └─INT─────┘             │
   ├─DOUBLE──┬─────────────┬──┤
   │         └─PRECISION───┘  │
   ├─VARCHAR──(──integer──)───┤
   ├─DECIMAL──(──integer──)───┤
   └─DATE─────────────────────┘
```

NICKNAME
: Specifies a unique name for this Extended Search nickname table. This name must be distinct from all other nicknames in the schema for which it is being defined. This parameter is required.

column-name
: Specifies one or more user-defined column names. The column name must match the name of a native or mapped field that is defined in the Extended Search configuration database. This parameter is optional.

data-type
: Specifies the SQL data type of the named column. This data type must correspond to the data type that is defined for this field in the Extended Search configuration database. For example, to search a field in an Extended Search

data source that has a String data type, define a VARCHAR column for this field in the nickname table. If you specify a *column-name*, this parameter is required.

FOR SERVER
Specifies the name of a previously registered server definition that was created for the Extended Search server that you want to search. This parameter is required.

APPLICATIONID
Specifies the name of the Extended Search application that you want to search. This name must exist in the Extended Search configuration database. This parameter is required.

CATEGORY
Specifies one or more Extended Search categories that you want to search. If you omit this option, you must specify at least one data source name. To specify multiple categories, delimit the category names with a semicolon. For example:

```
CATEGORY 'LotusNotes;MSAccess;LDAP'
```

DATASOURCE
Specifies one or more Extended Search data sources that you want to search. If you omit this option, you must specify at least one category name. To specify multiple data sources, delimit the data source names with a semicolon. For example:

```
DATASOURCE 'AltaVista;Google!;CNN'
```

VERTICAL_TABLE
Specifies the presentation format for search results. If you specify YES, Extended Search returns all fields that are configured as returnable, not just the user-defined columns. The wrapper stores the results in the nickname table as a vertical list of column names. The default value is NO.

TIMEOUT
An INTEGER that specifies the number of seconds to wait for a response from a server before the request times out. This option is optional. The default value is 30.

MAXHIT
An INTEGER that specifies the maximum number of results that can be returned from each source that is being searched. This option is optional. The default value is 50.

TOTALMAXHIT
An INTEGER that specifies the maximum number of results that can be returned from all the sources that are being searched. The wrapper combines these results into a single result set. This option is optional. The default value is 50.

SORTORDER
Specifies a sort order for the return of search results, either ascending (A) or descending (D). The default value is A.

SORTFIELD
Specifies the name of a field on which search results should be sorted. The default value, DOC_RANK, is a field that Extended Search uses to weigh the relevancy of a result document. If you specify a different field name, be sure that name exists in the sources that you search.

Related concepts:

- “Extended Search nicknames” on page 232
- “Extended Search vertical tables” on page 233

Related tasks:
- “Registering nicknames for Extended Search data sources” on page 239

Related reference:
- “CREATE NICKNAME statement” in the *SQL Reference, Volume 2*
- “Extended Search wrapper - Example queries” on page 242

CREATE FUNCTION statement syntax - Extended Search wrapper

The syntax for the CREATE FUNCTION statement is:

```
►►──CREATE FUNCTION ESWRAPPER.ES_SEARCH──(──INTEGER──,──VARCHAR(1024)──)──────────►

►─RETURNS INTEGER AS TEMPLATE DETERMINISTIC NO EXTERNAL ACTION────────────────────►◄
```

INTEGER
Defines the query reference parameter. In a query, this parameter must specify the name of an INTEGER column that is defined in the nickname table for which this custom function is being called. The value must be a bind column of the nickname, not a constant (for example, DOC_RANK).

The reference parameter identifies the nickname to which the ES_SEARCH function should be applied. The parameter itself is not evaluated.

If a SELECT statement contains more than one table in the FROM clause, and the WHERE clause contains an ES_SEARCH statement, the reference parameter allows you to tell DB2 which table a particular search statement belongs to. For example:

```
SELECT *
FROM ES_Nickname_1 as N1, ES_Nickname_2 as N2
WHERE  ESWRAPPER.ES_SEARCH(N1.DOC_RANK, 'IBM')=1 AND
       ESWRAPPER.ES_SEARCH(N2.DOC_RANK, 'LOTUS')=1
```

VARCHAR(1024)
Defines the query expression. In a query, this parameter must specify a string that uses Extended Search generalized query language.

Related tasks:
- “Registering the custom functions for the Extended Search wrapper” on page 240

Related reference:
- “CREATE FUNCTION (Sourced or Template) statement” in the *SQL Reference, Volume 2*
- “Extended Search wrapper - Example queries” on page 242
- “Extended Search wrapper - Generalized query language” on page 244

Entrez DDL reference information

CREATE SERVER statement arguments - Entrez wrapper

Arguments for the CREATE SERVER statement for Entrez are:

TYPE Specifies the type of the data source. The acceptable values for server type are `PubMed` and `Nucleotide`. These are case-insensitive.

VERSION
Specifies the version of the NCBI XML schema that you are using. This argument is optional. If the version of the server is not specified, the default is `1.0`.

WRAPPER
Specifies the name of the wrapper that you registered by using the CREATE WRAPPER statement.

Related tasks:
- "Registering the server for an Entrez data source" on page 197

Related reference:
- "CREATE SERVER statement" in the *SQL Reference, Volume 2*
- Appendix D, "Server options for federated systems," on page 575

CREATE NICKNAME statement options - Entrez wrapper

The following list describes the CREATE NICKNAME options for Entrez:

REMOTE_OBJECT
Specifies the name of the Entrez object type associated with the nickname. This name determines the schema and NCBI database for the nickname and its relationship to other nicknames. This name is case-insensitive.

PARENT
Specified only for a child nickname whose parent has been renamed through the REMOTE_OBJECT option. The PARENT option associates a child with a parent when multiple nickname families are defined within a DB2 schema. This name is case-sensitive.

Related tasks:
- "Registering nicknames for Entrez data sources" on page 199

Related reference:
- "CREATE NICKNAME statement" in the *SQL Reference, Volume 2*

Table-structured files DDL reference information

CREATE NICKNAME statement syntax - Table-structured file wrapper

The syntax for the CREATE NICKNAME statement is:

```
                                         ┌─,────────────────────────────────────┐
                                         ▼                                      │
►►──CREATE NICKNAME──nickname──(──────column-name──┤ column–information ├──────┴──)──────►

►──FOR SERVER──server-name──OPTIONS──────────────────────────────────────────────────────►


►──┬─(───────────────────────────────────────────────┬─)─────────────────────────────────►◄
   ├─FILE_PATH──'path'───────────────────────────────┤
   ├─,──COLUMN_DELIMITER──'delimiter'────────────────┤
   ├─,──SORTED──┬─'Y'─┬──────────────────────────────┤
   │            └─'N'─┘                              │
   │  (1)                                            │
   ├────────,──KEY_COLUMN──'key-column-name'─────────┤
   │  (1)                                            │
   └────────,──VALIDATE_DATA_FILE──┬─'Y'─┬───────────┘
                                   └─'N'─┘
```

column–information:

```
├──┤ data-type ├──┤ column-option ├──┤ nickname–column–options ├──────────────┤
```

data-type:

```
├──┬─SMALLINT──────────────────────────────────────────────────────────┬──────┤
   ├─┬─INTEGER─┬───────────────────────────────────────────────────────┤
   │ └─INT─────┘                                                       │
   ├─FLOAT──┬──────────────────┬───────────────────────────────────────┤
   │        └─(──integer──)────┘                                       │
   ├─REAL──────────────────────────────────────────────────────────────┤
   ├─DOUBLE──┬─────────────┬───────────────────────────────────────────┤
   │         └─PRECISION───┘                                           │
   ├─┬─DECIMAL─┬──┬──────────────────────────────────────┬─────────────┤
   │ ├─DEC─────┤  └─(──integer──┬──────────────┬──)─────┘              │
   │ ├─NUMERIC─┤                └─,──integer───┘                       │
   │ └─NUM─────┘                                                       │
   └─┬─┬─CLOB *────────────────────────────┬──┬────────────────────────┬─┬┘
     │ └─┬─CHARACTER─┬──LARGE OBJECT───────┘  └─(──integer──┬─────┬──)─┘ │
     │   └─CHAR──────┘                                      ├─K───┤      │
     │                                                      ├─M───┤      │
     │                                                      └─G───┘      │
     ├─┬─CHARACTER─┬──┬──────────────────┬───────────────────────────────┤
     │ └─CHAR──────┘  └─(──integer──)────┘                               │
     └─VARCHAR──(──integer──)────────────────────────────────────────────┘
```

column-option:

```
├──┬────────────┬─────────────────────────────────────────────────────────────┤
   └─NOT NULL───┘
```

nickname–column–options:

```
├──OPTIONS──(──DOCUMENT──'FILE' ──)──────────────────────────────────────────┤
```

Notes:

1 Not allowed for unsorted files. Optional for sorted files.

Restriction: The length of a CLOB is limited to 5 megabytes (5MB) for the table-structured file wrapper.

nickname
A unique nickname for the table-structured file to be accessed. It must be distinct from all other nicknames, tables, and views in the schema in which it is being registered.

column-name
A unique name given to each field in the table-structured file. Follow each column name with its data type. Only columns of type CHAR, CLOB, DECIMAL, DOUBLE, FLOAT, INTEGER, REAL, SMALLINT, and VARCHAR are supported.

CHARACTER(*integer***) or CHAR(***integer***) or CHARACTER or CHAR**
For a fixed-length character string of length *integer*, which can range from 1 to 254. If the length specification is omitted, a length of 1 character is assumed.

CLOB(*integer***)**
For a character large object of length *integer*, with a maximum length of 5 megabytes. If the length specification is omitted, a length of 1 megabyte is assumed.

DECIMAL(*precision-integer, scale-integer***) or DEC(***precision-integer, scale-integer***)**
For a decimal number.

The first integer is the precision of the number; that is, the total number of digits. This value can range from 1 to 31.

The second integer is the scale of the number; that is, the number of digits to the right of the decimal point. This value can range from 0 to the precision of the number.

If precision and scale are not specified, the default values of 5,0 are used.

The words **NUMERIC** and **NUM** can be used as synonyms for **DECIMAL** and **DEC**.

DOUBLE or DOUBLE PRECISION
For double precision floating-point.

FLOAT(*integer***)**
For a single or double precision floating-point number, depending on the value of *integer*. The value of *integer* must be in the range 1 through 53. The values 1 through 24 indicate single precision and the values 25 through 53 indicate double precision.

INTEGER or INT
For a large integer.

REAL For single precision floating-point.

SMALLINT
For a small integer.

VARCHAR(*integer*)
For a varying-length character string of maximum length *integer*, which can range from 1 to 32672.

NOT NULL
Prevents the column from containing null values.

The wrapper does not enforce the NOT NULL constraint, but DB2 does. If you create a nickname and attach a NOT NULL constraint on a column and then select a row containing a null value for the column, DB2 will issue a SQL0407N error stating that you can't assign a NULL value to a NOT NULL column.

The exception to this rule is for sorted nicknames. The key column for sorted nicknames cannot be NULL. If a NULL key column is found for a sorted nickname, the SQL1822N error is issued, stating that the key column is missing.

FOR SERVER
Identifies the server you registered using the CREATE SERVER statement. This server will be used to access the table-structured file.

FILE_PATH
The fully qualified path to the table-structured file to be accessed, enclosed in single quotation marks. The data file must be a standard file or a symbolic link, rather then a pipe or another non-standard file type. Either the FILE_PATH or the DOCUMENT nickname column option should be specified. If the FILE_PATH nickname option is specified then no DOCUMENT nickname column option can be specified.

SORTED
Specifies whether the data source file is sorted or unsorted. This option accepts either 'Y', 'y', 'n', or 'N'. It has a default value of 'N'.

Sorted data sources must be sorted in ascending order according to the collation sequence for the current locale, as defined by the settings in the LC_COLLATE National Language Support category.

If you specify that the data source is sorted, it is recommended you set VALIDATE_DATA_FILE to 'Y'.

COLUMN_DELIMITER
The delimiter used to separate columns of the table-structured file, enclosed in single quotation marks. The delimiter can be a single character or multiple characters. If no column delimiter is defined, the column delimiter defaults to the comma. A single quote cannot be used as a delimiter. The column delimiter must be consistent throughout the file. A null value is represented by two delimiters next to each other or a delimiter followed by a line terminator, if the NULL field is the last one on the line. The column delimiter cannot exist as valid data for a column. For example, a column delimiter of a comma cannot be used if one of the columns contains data with embedded commas.

KEY_COLUMN
The name of the column in the file that forms the key on which the file is sorted, enclosed in single quotation marks. Use this option for sorted files only. A column that is designated with the DOCUMENT nickname column option must not be specified as the key column.

Only single-column keys are supported. Multi-column keys are not allowed. The value must be the name of a column defined in the CREATE

NICKNAME statement. The column must be sorted in ascending order. If the value is not specified for a sorted nickname, it defaults to the first column in the nicknamed file. It is recommended that the key column be designated not nullable by adding the NOT NULL option to its definition in the nickname statement.

This option is case-sensitive. However, DB2 folds column names to uppercase unless the column is defined with double quotes.

VALIDATE_DATA_FILE
For sorted files, this option specifies whether the wrapper verifies that the key column is sorted in ascending order and checks for NULL keys. The only valid values for this option are ′Y′ or ′N′, enclosed in single quotation marks. The check is done once at registration time. If this option is not specified, then no validation takes place. This option is not allowed if the DOCUMENT nickname column option is used for the file path.

DOCUMENT
Specifies the kind of table-structured file. Currently, this wrapper only supports FILE for this option. Only one column can be specified with the DOCUMENT option per nickname. The column associated with the DOCUMENT option has to be of data type VARCHAR or CHAR.

Using the DOCUMENT nickname column option, instead of the FILE_PATH nickname option, implies that the file corresponding to this nickname will be supplied during query execution. If the DOCUMENT option has the ″FILE″ value, it means that what will be supplied during query execution is the full path of the file whose schema matches the nickname definition for this nickname. The following CREATE NICKNAME example illustrates the use of the DOCUMENT nickname column option.

```
CREATE NICKNAME customers
(
doc VARCHAR(100) OPTIONS(DOCUMENT 'FILE'),
name VARCHAR(16),
address VARCHAR(30),
id VARCHAR(16)
)
FOR SERVER file_server
```

The following query, specifying the location of the table-structured file in the WHERE clause, can now be run against the customers nickname:

```
SELECT name, address, id FROM customers
WHERE doc='/home/db2user/Customers.txt'
```

Related tasks:

- "Registering the server for table-structured files" on page 359

Related reference:

- "CREATE NICKNAME statement" in the *SQL Reference, Volume 2*
- "CREATE NICKNAME statement - Examples for table-structured file wrapper" on page 360

XML DDL reference information

CREATE NICKNAME statement syntax - XML wrapper

The syntax for the CREATE NICKNAME statement is:

```
►►──CREATE NICKNAME──nickname──(──column-name───┤ Column structure ├───)──────────►

►─FOR SERVER──server-name──OPTIONS──(─────────────────────────────────────────────►
                                      └─┬─FILE_PATH──'path' ──,──────┬─┘
                                        └─DIRECTORY_PATH──'path' ──,─┘

►─XPATH──'xpath_expression' ──┤ Nickname parameters ├──)─────────────────────────►◄
```

Column structure:

```
├──┤ Data type options ├──┬──────────┬──┤ Nickname column options ├──────────┤
                          └─NOT NULL─┘
```

Data type options:

```
├───┬─SMALLINT──────────────────────────────────────────────────┬──────────┤
    ├─┬─INTEGER─┬───────────────────────────────────────────────┤
    │ └─INT─────┘                                               │
    ├─REAL──────────────────────────────────────────────────────┤
    ├─DOUBLE──┬───────────┬─────────────────────────────────────┤
    │         └─PRECISION─┘                                     │
    ├─┬─DECIMAL─┬──┬───────────────────────────────────┬────────┤
    │ ├─DEC─────┤  └─(──integer──┬─────────────┬──)────┘        │
    │ ├─NUMERIC─┤                └─,──integer──┘                │
    │ └─NUM─────┘                                               │
    ├─┬─┬─CLOB *──────────────────────────┬──┬────────────────────┬─┬─┤
    │ │ └─┬─CHARACTER─┬──LARGE OBJECT─────┘  └─(──integer──┬───┬──)─┘ │ │
    │ │   └─CHAR──────┘                                    ├─K─┤      │ │
    │ │                                                    ├─M─┤      │ │
    │ │                                                    └─G─┘      │ │
    │ ├─┬─CHARACTER─┬──┬────────────────┬─────────────────────────────┤ │
    │ │ └─CHAR──────┘  └─(──integer──)──┘                             │ │
    │ └─VARCHAR──(──integer──)────────────────────────────────────────┘ │
    └─DATE──────────────────────────────────────────────────────────────┘
```

Nickname column options:

```
├──OPTIONS──(──┬─DOCUMENT──┬─'FILE' ──────────┬──┬──)──────────────────────┤
               │           ├─'DIRECTORY' ─────┤  │
               │           ├─'URI' ───────────┤  │
               │           └─'COLUMN' ────────┘  │
               ├─XPATH──'xpath_expression' ──────┤
               ├─PRIMARY_KEY──'YES' ─────────────┤
               └─FOREIGN_KEY──'parent_nickname' ─┘
```

Nickname parameters:

```
├──┬───────────────────────────────┬──┬───────────────────────────────┬──────────────►
   │           ┌─'NO'──┐           │  │          ┌─'NO'──┐            │
   └─STREAMING─┴─'YES'─┴──,────────┘  └─VALIDATE─┴─'YES'─┴──,─────────┘

►──┬────────────────────────────────┬──┬────────────────────────────┬─────────────────►
   └─INSTANCE_PARSE_TIME─'value'─,──┘  └─XPATH_EVAL_TIME─'value'─,──┘

►──┬───────────────────────┬─────────────────────────────────────────────────────────┤
   └─NEXT_TIME─'value'─────┘
```

Restriction: The length of a CLOB is limited to 5 megabytes (5MB) for the XML wrapper.

Nickname parameters and options:

FILE_PATH
: Specifies the file path of the XML document. If you specify this nickname option, do not specify a DOCUMENT column. This FILE_PATH option is accepted only for the root nickname (the nickname that identifies the elements at the top level of the XML document).

DIRECTORY_PATH
: Specifies the path name of a directory that contains one or more XML files. Use this option to create a single nickname over multiple XML source files. The XML wrapper uses only those files with a `.xml` extension that are located in the directory that you specify. The XML wrapper ignores all other files in this directory. If you specify this nickname option, do not specify a DOCUMENT column. This DIRECTORY_PATH option is accepted only for the root nickname (the nickname that identifies the elements at the top level of the XML document).

XPATH
: Specifies an XPath expression that identifies the XML elements that represent individual tuples. The XPATH nickname option for a child nickname is evaluated in the context of the path that is specified by the XPATH nickname option of its parent. This XPath expression is used as a context for evaluating column values that are identified by the XPATH nickname column options.

  Do not specify a namespace prefix in an XPath expression. The XML wrapper does not support namespaces.

Nickname column options:

DOCUMENT
: Specifies that this column is a DOCUMENT column. The value of the DOCUMENT column indicates the type of XML source data that is supplied to the nickname when the query runs. This option is accepted only for columns of the root nickname (the nickname that identifies the elements at the top level of the XML document). Only one column can be specified with the DOCUMENT option per nickname. The column that is associated with the DOCUMENT option must be a VARCHAR data type.

  If you use a DOCUMENT column option, instead of the FILE_PATH or DIRECTORY_PATH nickname option, the document that corresponds to this nickname is supplied when the query runs.

The valid values for the DOCUMENT option are:

FILE Specifies that the value of the nickname column is bound to the path name of a file that contains an XML document. The data from this file is supplied when the query runs.

DIRECTORY
Specifies that the value of the nickname column is bound to the path name of a directory that contains multiple XML data files. The XML data from multiple files is supplied when the query runs. The data is located in XML files that reside under the specified directory path. The XML wrapper uses only those files with a `.xml` extension that are located in the directory that you specify. The XML wrapper ignores all other files in this directory.

URI Specifies that the value of the nickname column is bound to the path name of a remote XML file to which a URI refers. The URI address indicates the remote location of this XML file on the Web.

COLUMN
Specifies that the XML document is stored in a relational column.

XPATH
Specifies the XPath expression in the XML document that contains the data that corresponds to this column. The XML wrapper evaluates the XPath expression after the CREATE NICKNAME statement applies this XPath expression from this XPATH nickname option.

If you run a query on a column name that has an incorrectly configured XPATH tag reference such as incorrect case, your query returns null values in this column for all returned rows.

Do not specify a namespace prefix in an XPath expression. The XML wrapper does not support namespaces.

PRIMARY_KEY
Indicates that this nickname is a parent nickname. The column data type must be VARCHAR(16). A nickname can have at most one PRIMARY_KEY column option. 'YES' is the only valid value. The column that is designated with this option holds a key that is generated by the wrapper. The column's value cannot be retrieved in a SELECT query, and the XPATH option must not be specified for this column. The column can be used only to join parent nicknames and child nicknames.

FOREIGN_KEY
Indicates that this nickname is a child nickname and specifies the name of the corresponding parent nickname. A nickname can have at most one FOREIGN_KEY column option. The value for this option is case sensitive. The column that is designated with this option holds a key that is generated by the wrapper. The column's value cannot be retrieved in a SELECT query, and the XPATH option must not be specified for this column. The column can be used only to join parent nicknames and child nicknames.

A CREATE NICKNAME statement with a FOREIGN_KEY option will fail if the parent nickname has a different schema name.

Unless the nickname that is referred to in a FOREIGN_KEY clause was explicitly defined to be lowercase or mixed case by enclosing it in quotation marks under the corresponding CREATE NICKNAME statement, then when you refer to this nickname in the FOREIGN_KEY clause, you must specify the nickname in uppercase.

Nickname parameters:

STREAMING
Specifies whether the XML source document is separated into logical fragments that correspond to the node that matches the XPath expression of the nickname. The XML wrapper then parses and processes the XML source data fragment by fragment, reducing total memory use. You can specify streaming for any XML source document (`FILE`, `DIRECTORY`, `URI`, or `COLUMN`). This option is accepted only for columns of the root nickname (the nickname that identifies the elements at the top level of the XML document). The default streaming value is NO.

Do not set the STREAMING parameter to YES if you set the VALIDATE parameter to YES. If you set both parameters to YES, you will receive an error message.

VALIDATE
Specifies whether the XML source document is validated before the XML data is extracted. If this option is set to YES, the nickname option verifies that the structure of the source document conforms to an XML schema or to a document type definition (DTD). This option is accepted only for columns of the root nickname (the nickname that identifies the elements at the top level of the XML document). The default value is NO.

The XML source document is not validated if the XML wrapper cannot locate the XML schema file or DTD file (`.xsd` or `.dtd`). DB2 does not issue an error message if the validation does not occur. Therefore, ensure that the XML schema file or DTD file exists in the location that is specified in the XML source document.

Do not set the VALIDATE parameter to YES if you set the STREAMING parameter to YES. If you set both parameters to YES, you will receive an error message.

INSTANCE_PARSE_TIME
Specifies the time (in milliseconds) to parse the data in one row of the XML source document. You can modify the INSTANCE_PARSE_TIME, XPATH_EVAL_TIME, and NEXT_TIME options to optimize queries of large or complex XML source structures. This option is accepted only for columns of the root nickname (the nickname that identifies the elements at the top level of the XML document). The number that you specify can be an integer or a decimal value. The default value is 7 milliseconds.

XPATH_EVAL_TIME
Specifies the time (in milliseconds) to evaluate the XPath expression of the nickname and to locate the first element. You can modify the XPATH_EVAL_TIME, INSTANCE_PARSE_TIME, and NEXT_TIME options to optimize queries of large or complex XML source structures. This option is accepted for root nicknames and nonroot nicknames. The number that you specify can be an integer or a decimal value. The default value is 1 millisecond.

NEXT_TIME
Specifies the time (in milliseconds) that is required to locate subsequent source elements from the XPath expression. You can modify the NEXT_TIME, XPATH_EVAL_TIME, and INSTANCE_PARSE_TIME options

to optimize queries of large or complex XML source structures. This option is accepted for root nicknames and nonroot nicknames. The default value is 1 millisecond.

Usage notes:

If you use the DATE data type option, the dates in your XML source document must have the following format: CCYY-MM-DD. For example, if the date is 17 November 2002, the date must be specified as 2002-11-17 in the XML source document. If a date has any other format, you will receive an error message.

Do not set both the STREAMING parameter and the VALIDATE parameter to YES. The XML wrapper validates an entire XML source document and does not validate source document fragments. If you set both parameters to YES, you will receive an error message.

Related tasks:

- "Registering nicknames for XML data sources" on page 424

Related reference:

- "CREATE NICKNAME statement" in the *SQL Reference, Volume 2*
- "CREATE NICKNAME statement - Examples for XML wrapper" on page 425

Appendix A. Views in the global catalog table containing federated information

Most of the catalog views in a federated database are the same as the catalog views in any other DB2 for Linux, UNIX, and Windows database. There are several unique views which contain information pertinent to a federated system, such as the SYSCAT.WRAPPERS view.

As noted in the DB2 for Linux, UNIX, and Windows Version 6 and Version 7 SQL Reference manuals, the DB2 Version 8 SYSCAT views are now read-only. If you issue an UPDATE or INSERT operation on a view in the SYSCAT schema, it will fail. Using the SYSSTAT views is the recommended way to update the system catalog. Change applications that reference the SYSCAT view to reference the updatable SYSSTAT view instead.

The following table lists the SYSCAT views which contain federated information. These are read-only views.

Table 138. Catalog views typically used with a federated system

Catalog views	Description
SYSCAT.CHECKS	Contains check constraint information that you defined.
SYSCAT.COLCHECKS	Contains columns referenced by a check constraint.
SYSCAT.COLUMNS	Contains column information about the data source objects (tables and views) that you created nicknames for.
SYSCAT.COLOPTIONS	Contains information about column option values that you set for a nickname.
SYSCAT.CONSTDEP	Contains the dependency of an informational constraint that you defined.
SYSCAT.DATATYPES	Contains data type information about local built-in and user-defined DB2 data types.
SYSCAT.DBAUTH	Contains the database authorities held by individual users and groups.
SYSCAT.FUNCMAPOPTIONS	Contains information about option values that you have set for a function mapping.
SYSCAT.FUNCMAPPINGS	Contains the function mappings between the federated database and the data source objects.
SYSCAT.INDEXCOLUSE	Contains columns that participate in an index.
SYSCAT.INDEXES	Contains index specifications for data source objects.
SYSCAT.KEYCOLUSE	Contains columns that participate in a key defined by a unique key, primary key, or foreign key constraint.
SYSCAT.REFERENCES	Contains information about referential constraints defined by you.

Table 138. Catalog views typically used with a federated system (continued)

Catalog views	Description
SYSCAT.ROUTINES	Contains local DB2 user-defined functions, or function templates. Function templates are used to map to a data source function.
SYSCAT.REVTYPEMAPPINGS	This view is not used. All data type mappings are recorded in the SYSCAT.TYPEMAPPINGS view.
SYSCAT.SERVEROPTIONS	Contains information about server option values that you set with a server definition.
SYSCAT.SERVERS	Contains server definitions that you create for data source servers.
SYSCAT.TABCONST	Each row represents a table and nickname constraints of type CHECK, UNIQUE, PRIMARY KEY, or FOREIGN KEY.
SYSCAT.TABLES	Contains information about each local DB2 table, federated view, and nickname that you create.
SYSCAT.TYPEMAPPINGS	Contains forward data type mappings and reverse data type mappings. The mapping is to local DB2 data types from data source data types. These mappings are used when you create a nickname on a data source object.
SYSCAT.USEROPTIONS	Contains user authorization information that you set when you create user mappings between the federated database and the data source servers.
SYSCAT.VIEWS	Contains information about local federated views that you create.
SYSCAT.WRAPOPTIONS	Contains information about option values that you have set for a wrapper.
SYSCAT.WRAPPERS	Contains the name of the wrapper and library file for each data source that you create a wrapper for.

The following table lists the SYSSTAT views which contain federated information. These are read-write views that contain statistics you can update.

Table 139. Federated updatable global catalog views

Catalog views	Description
SYSSTAT.COLUMNS	Contains statistical information about each column in the data source objects (tables and views) that you have created nicknames for. Statistics are not recorded for inherited columns of typed tables.
SYSSTAT.INDEXES	Contains statistical information about each index specification for data source objects.
SYSSTAT.ROUTINES	Contains statistical information about each user-defined function. Does not include built-in functions. Statistics are not recorded for inherited columns of typed tables.

Table 139. Federated updatable global catalog views (continued)

Catalog views	Description
SYSSTAT.TABLES	Contains information about each base table. View, synonym, and alias information is not included in this view. For typed tables, only the root table of a table hierarchy is included in the view. Statistics are not recorded for inherited columns of typed tables.

Appendix B. Wrapper options for federated systems

Wrapper options are used to configure the wrapper or to define how the federated server uses the wrapper. Wrapper options can be set when you create or alter the wrapper.

All relational and nonrelational data sources use the DB2_FENCED wrapper option. The ODBC data source uses the MODULE wrapper option. The Entrez data source uses the EMAIL wrapper option.

Table 140. Wrapper options and their settings

Option	Valid settings	Default setting
DB2_FENCED	Specifies whether the wrapper runs in fenced or trusted mode. **Y** The wrapper runs in fenced mode. **N** The wrapper runs in trusted mode.	Relational wrappers: N. Nonrelational wrappers from IBM: N. Nonrelational wrappers from third parties: Y.
EMAIL	Specifies an e-mail address when you register the Entrez wrapper. This e-mail address is included with all queries and allows NCBI to contact you if there are problems, such as too many queries overloading the NCBI servers. This option is required.	
MODULE	Specifies the full path of the library that contains the ODBC Driver Manager implementation or the SQL/CLI implementation. Required for the ODBC wrapper on UNIX federated servers.	On Windows, the default value is odbc32.dll

Related concepts:

- "Parallelism with queries that reference nicknames" in the *Federated Systems Guide*

Related tasks:

- "Trusted and fenced mode process environments" in the *IBM DB2 Information Integrator Wrapper Developer's Guide*
- "Altering a wrapper" in the *Federated Systems Guide*
- "Registering wrappers for a data source" on page 61

Appendix C. Valid server types in SQL statements

Server types indicate what kind of data source that the server definition represents. Server types vary by vendor, purpose, and operating system. Supported values depend on the wrapper being used.

For most data sources, you must specify a valid server type in the CREATE SERVER statement.

BioRS wrapper

BioRS data sources.

Server Type	Data Source
Not required in the CREATE SERVER statement.	BioRS

BLAST wrapper

BLAST data sources supported by the BLAST daemon.

Server Type	Data Source
BLASTN	BLAST searches in which a nucleotide sequence is compared with the contents of a nucleotide sequence database to find sequences with regions homologous to regions of the original sequence.
BLASTP	BLAST searches in which an amino acid sequence is compared with the contents of an amino acid sequence database to find sequences with regions homologous to regions of the original sequence.
BLASTX	BLAST searches in which a nucleotide sequence is compared with the contents of an amino acid sequence database to find sequences with regions homologous to regions of the original sequence.
TBLASTN	BLAST searches in which an amino acid sequence is compared with the contents of a nucleotide sequence database to find sequences with regions homologous to regions of the original sequence.
TBLASTX	BLAST searches in which a nucleotide sequence is compared with the contents of a nucleotide sequence database to find sequences with regions homologous to regions of the original sequence.

CTLIB wrapper

Sybase data sources supported by the CTLIB client software.

Server Type	Data Source
SYBASE	Sybase

Documentum wrapper

Documentum data sources supported by the Documentum Client API/Library.

Server Type	Data Source
DCTM	Documentum

DRDA wrapper

DB2 Family data sources

Table 141. DB2 for Linux, UNIX, and Windows

Server Type	Data Source
DB2/UDB	IBM DB2 Universal Database
DB2/6000	IBM DB2 for AIX
DB2/AIX	IBM DB2 for AIX
DB2/HPUX	IBM DB2 for HP-UX
DB2/HP	IBM DB2 for HP-UX
DB2/NT	IBM DB2 for Windows NT
DB2/EEE	IBM DB2 Enterprise-Extended Edition
DB2/SUN	IBM DB2 for Solaris
DB2/PE	IBM DB2 for Personal Edition
DB2/2	IBM DB2 for OS/2
DB2/LINUX	IBM DB2 for Linux
DB2/PTX	IBM DB2 for NUMA-Q
DB2/SCO	IBM DB2 for SCO Unixware

Table 142. DB2 for iSeries (and AS/400)

Server Type	Data Source
DB2/400	IBM DB2 for iSeries and AS/400

Table 143. DB2 for z/OS and OS/390

Server Type	Data Source
DB2/ZOS	IBM DB2 for z/OS
DB2/390	IBM DB2 for OS/390
DB2/MVS	IBM DB2 for MVS

Table 144. DB2 Server for VM and VSE

Server Type	Data Source
DB2/VM	IBM DB2 for VM
DB2/VSE	IBM DB2 for VSE
SQL/DS	IBM SQL/DS

Entrez wrapper

Entrez data sources.

Server Type	Data Source
NUCLEOTIDE	Entrez
PUBMED	Entrez

Excel wrapper

Excel data sources supported by Microsoft Excel 97, 2000, and 2002.

Server Type	Data Source
Not required in the CREATE SERVER statement.	Microsoft Excel

Extended Search wrapper

Extended Search data sources supported by the Extended Search Client Library.

Server Type	Data Source
Not required in the CREATE SERVER statement.	IBM Lotus Extended Search

HMMER wrapper

HMMER data sources supported by the HMMER daemon.

Server Type	Data Source
PFAM	HMMER
SEARCH	HMMER

Informix wrapper

Informix data sources supported by Informix Client SDK software.

Server Type	Data Source
INFORMIX	Informix

MSSQLODBC3 wrapper

Microsoft SQL Server data sources supported by the DataDirect Connect ODBC 3.6 driver or the ODBC 3.0 (or later) driver

Server Type	Data Source
MSSQLSERVER	Microsoft SQL Server

NET8 wrapper

Oracle data sources supported by Oracle NET8 client software.

Server Type	Data Source
ORACLE	Oracle Version 8.0. or later

ODBC wrapper

ODBC data sources supported by the ODBC 3.x driver.

Server Type	Data Source
ODBC	ODBC

OLE DB wrapper

OLE DB providers compliant with Microsoft OLE DB 2.0 or later.

Server Type	Data Source
Not required in the CREATE SERVER statement.	Any OLE DB provider

Table-structured files wrapper

Table-structured file data sources.

Server Type	Data Source
Not required in the CREATE SERVER statement.	Table-structured files

Teradata wrapper

Teradata data sources supported by the Teradata V2R3, V2R4, and V2R5 client software.

Server Type	Data Source
TERADATA	Teradata

Web services wrapper

Web services data sources.

Server Type	Data Source
Not required in the CREATE SERVER statement.	Any Web services data source.

WebSphere Business Integration wrapper

Business application data sources supported by the WeSphere Business Integration wrapper.

Server Type	Data Source
WBI	WebSphere Business Integration 2.2 or 2.3

XML wrapper

XML data sources.

Server Type	Data Source
Not required in the CREATE SERVER statement.	XML

Appendix D. Server options for federated systems

Server options are used to describe a data source server. Server options specify data integrity, location, security, and performance information. Some server options are available for all data sources, and other server options are data source specific.

The common federated server options for relational data sources are:

- Compatibility options. COLLATING_SEQUENCE, IGNORE_UDT
- Data integrity options. IUD_APP_SVPT_ENFORCE
- Data and time options. DATEFORMAT, TIMEFORMAT, TIMESTAMPFORMAT
- Location options. CONNECTSTRING, DBNAME, IFILE
- Security options. FOLD_ID, FOLD_PW, INFORMIX_LOCK_MODE
- Performance options. COMM_RATE, CPU_RATIO, DB2_MAXIMAL_PUSHDOWN, IO_RATIO, LOGIN_TIMEOUT, PACKET_SIZE, PLAN_HINTS, PUSHDOWN, TIMEOUT, VARCHAR_NO_TRAILING_BLANKS

The following table lists the server definition server options applicable for each relational data source.

Table 145. Server options for relational data sources

Data Source	CODEPAGE	COLLATING_SEQUENCE	COMM_RATE	CONNECTSTRING	CPU_RATIO	DATEFORMAT	DB2_MAXIMAL_PUSHDOWN	DBNAME	FOLD_ID	FOLD_PW	IFILE	INFORMIX_LOCK_MODE	IO_RATIO	IUD_APP_SVPT_ENFORCE	LOGIN_TIMEOUT	NODE	PACKET_SIZE	PASSWORD	PLAN_HINTS	PUSHDOWN	TIMEOUT	TIMEFORMAT	TIMESTAMPFORMAT	VARCHAR_NO_TRAILING_BLANKS
DB2 UDB for iSeries		X	X		X		X	X	X	X			X	X				X		X				X
DB2 UDB for z/OS and OS/390		X	X		X		X	X	X	X			X	X				X		X				X
DB2 for VM and VSE		X	X		X		X	X	X	X			X	X				X		X				X
DB2 UDB for Linux, UNIX, and Windows		X	X		X		X	X	X	X			X	X				X		X				X
Informix		X	X		X		X	X	X	X		X	X	X		X		X		X				
Microsoft SQL Server	X	X	X		X		X	X	X	X			X	X		X		X		X				
ODBC	X	X	X		X	X	X	X	X	X			X	X		X		X		X		X	X	X

Table 145. Server options for relational data sources (continued)

Data Source	CODEPAGE	COLLATING_SEQUENCE	COMM_RATE	CONNECTSTRING	CPU_RATIO	DATEFORMAT	DB2_MAXIMAL_PUSHDOWN	DBNAME	FOLD_ID	FOLD_PW	IFILE	INFORMIX_LOCK_MODE	IO_RATIO	IUD_APP_SVPT_ENFORCE	LOGIN_TIMEOUT	NODE	PACKET_SIZE	PASSWORD	PLAN_HINTS	PUSHDOWN	TIMEOUT	TIMEFORMAT	TIMESTAMPFORMAT	VARCHAR_NO_TRAILING_BLANKS
OLE DB		X		X																				
Oracle		X	X		X		X		X	X			X			X		X	X	X				X
Sybase		X	X		X		X	X	X	X	X		X		X	X	X	X	X	X	X			
Teradata		X	X		X		X						X	X		X				X				

The following table lists the server definition server options applicable for each nonrelational data source, except WebSphere Business Integration. The server definition server options for WebSphere Business Integration are listed in Table 147 on page 577.

Table 146. Server options for nonrelational data sources.

Data Source	CASE_SENSITIVE	CONTENT_DIR	DAEMON_PORT	ES_HOST	ES_PORT	ES_TRACING	ES_TRACELEVEL	ES_TRACEFILENAME	HMMPFAM_OPTIONS	HMMSEARCH_OPTIONS	MAX_ROWS	NODE	OS_TYPE	PORT	PROCESSORS	PROXU_AUTHID	PROXY_PASSWORD	PROXY_SERVER_NAME	PROXY_SERVER_PORT	PROXY_TYPE	RDBMS_TYPE	SOCKET_TIMEOUT	TIMEOUT	TRANSACTIONS	USE_CLOB_SEQUENCE
BioRS	X											X		X									X		
BLAST			X									X													X
Documentum		X										X	X								X			X	
Entrez											X					X	X	X	X	X		X			
Excel																									
Extended Search				X	X	X	X	X																	
HMMER			X						X	X		X			X										X
Table-structured files																									
Web services																									
XML																X	X	X	X	X		X			

The following table lists the server definition server options applicable for WebSphere Business Integration data sources.

Table 147. Server options for WebSphere Business Integration data sources.

Data Source	APP_TYPE	FAULT_QUEUE	MQ_CONN_NAME	MQ_MANAGER	MQ_RESPONSE_TIMEOUT	MQ_SVRCONN_CHANNELNAME	REQUEST_QUEUE	RESPONSE_QUEUE
WebSphere Business Integration	X	X	X	X	X	X	X	X

The following table describes each server option and lists the valid and default settings.

Table 148. Server options and their settings

Option	Description and valid settings	Default setting
APP_TYPE	The type of remote application. Valid values are ′PSOFT′, ′SAP′, and ′SIEBEL′. This option is required.	None.

Table 148. Server options and their settings (continued)

Option	Description and valid settings	Default setting
CASE_SENSITIVE	Specifies whether the BioRS server treats names in a case sensitive manner. Valid values are Y or N. 'Y' The BioRS server treats names in a case sensitive manner. 'N' The BioRS server does not treat names in a case sensitive manner In the BioRS product, a configuration parameter controls the case sensitivity of the data that is stored on the BioRS server. The CASE_SENSITIVE option is the DB2 Information Integrator counterpart to that BioRS system configuration parameter. You must synchronize the BioRS server case sensitivity configuration settings in your BioRS system and in DB2 Information Integrator. If you do not keep the case sensitivity configuration settings synchronized between BioRS and DB2 Information Integrator, errors will occur when you attempt to access BioRS data through DB2 Information Integrator. You cannot change or delete the CASE_SENSITIVE option after you create a new BioRS server in DB2 Information Integrator. If you need to change the CASE_SENSITIVE option, you must drop and then create the entire server again. If you drop the BioRS server, you must also create all of the corresponding BioRS nicknames again. DB2 Information Integrator automatically drops all nicknames that correspond to a dropped server.	Y
CODEPAGE	Specifies the DB2 code page identifier corresponding to the coded character set of the data source client configuration. You must specify the client's code page if the client's code page and the federated database code page do not match. For data sources that support Unicode, the CODEPAGE option can be set to the DB2 code page identifier corresponding to the supported Unicode encoding of the data source client.	On UNIX or Windows systems with a non-Unicode federated database: The federated database code page. On UNIX systems with a Unicode federated database: 1208 On Windows systems with a Unicode federated database: 1202

Table 148. Server options and their settings (continued)

Option	Description and valid settings	Default setting
COLLATING_ SEQUENCE	Specifies whether the data source uses the same default collating sequence as the federated database, based on the NLS code set and the country/region information. 'Y' The data source has the same collating sequence as the DB2 federated database. 'N' The data source has a different collating sequence than the DB2 federated database collating sequence. 'I' The data source has a different collating sequence than the DB2 federated database collating sequence, and the data source collating sequence is insensitive to case (for example, 'STEWART' and 'StewART' are considered equal).	'N'
COMM_RATE	Specifies the communication rate between the federated server and the data source server. Expressed in megabytes per second. Valid values are greater than 0 and less than 1x10^{23}. Values can be expressed in any valid REAL notation.	'2'
CONTENT_DIR	Specifies the name of the locally-accessible root directory for storing content files retrieved by the GET_FILE, GET_FILE_DEL, GET_RENDITION, and GET_RENDITION_DEL pseudo columns. It must be writable by all users who can use these pseudo columns.	On UNIX systems: '/tmp' On Windows systems: 'C:\temp'
CONNECTSTRING	Specifies initialization properties needed to connect to an OLE DB provider.	None.
CPU_RATIO	Indicates how much faster or slower a data source CPU runs than the federated server CPU. Valid values are greater than 0 and less than 1x10^{23}. Values can be expressed in any valid REAL notation. A setting of 1 indicates that the DB2 federated CPU speed and the data source CPU speed have the same CPU speed, a 1:1 ratio. A setting of .5 indicates that the DB2 federated CPU speed is 50% slower than the data source CPUO speed. A setting of 2 indicates that the DB2 federated CPU speed is twice as fast as the data source CPU speed.	'1.0'
DATEFORMAT	The date format used by the data source. Enter the format using 'DD', 'MM', and 'YY' or 'YYYY' to represent the numeric form of the date. You should also specify the delimiter such as a space or comma. For example, to represent the date format for '2003-01-01', use 'YYYY-MM-DD'. This field is nullable.	None.

Table 148. Server options and their settings (continued)

Option	Description and valid settings	Default setting
DAEMON_PORT	Specifies the port number on which the daemon will listen for BLAST or HMMER job requests. The port number must be the same number specified in the DAEMON_PORT option of the daemon configuration file.	BLAST: ’4007’; HMMER: ’4098’
DB2_MAXIMAL_ PUSHDOWN	Specifies the primary criteria that the query optimizer uses when choosing an access plan. The query optimizer can choose access plans based on cost or based on the user requirement that as much query processing as possible be performed by the remote data sources. ’Y’ The query optimizer chooses an access plan that pushes down more query operations to the data source than other plans. When several access plans provide the same amount of pushdown, the query optimizer then chooses the plan with the lowest cost. If a materialized query table (MQT) on the federated server can process part or all of the query, then an access plan that includes the materialized query table is might be used. The federated database does not push down queries that result in a Cartesian product. ’N’ The query optimizer chooses an access plan based on cost.	’N’
DBNAME	Name of the data source database that you want the federated server to access. For DB2 database, this value corresponds to a specific database for the initial remote DB2 database connection. This specific database is the database alias for the remote DB2 database that is cataloged at the federated server using the CATALOG DATABASE command or the DB2 Configuration Assistant. Does not apply to Oracle data sources because Oracle instances contain only one database.	None.
ES_HOST	Specifies the fully qualified host name or IP address of the Extended Search server that you want to search. This option is required.	None.
ES_PORT	Specifies the port number where this Extended Search server listens for requests. This option is optional.	’6001’
ES_TRACING	Specifies whether tracing should be enabled for error messages, warning messages, and informational messages that are produced by the remote Extended Search server. Valid values are: ’OFF’ No trace messages will be logged. ’ON’ Trace messages will be logged. This option is optional.	’OFF’

Table 148. Server options and their settings (continued)

Option	Description and valid settings	Default setting

ES_TRACELEVEL	If tracing is enabled, this option specifies the types of messages that will be written to the log file. You can enable and disable the following trace levels independently:
	'C' Critical error messages.
	'N' Noncritical messages.
	'W' Warning messages.
	'I' Informational messages.
	For example:
	`ES_TRACELEVEL 'W'`
	`ES_TRACELEVEL 'CN'`
	This option is optional.
		'C'

ES_TRACEFILENAME	If tracing is enabled, this option specifies the name of a directory and file where messages will be written. This option is optional.	For UNIX operating systems: $INSTHOME/sqllib/log/ ESWrapper.log. For Windows operating systems: %DB2TEMPDIR%\ ESWrapper.log.
FAULT_QUEUE	The name of the fault queue that delivers error messages from the adapter to the wrapper. The name must conform to the specifications for queue names for WebSphere MQ. This is a required option.	None.
FOLD_ID (See notes 1 and 4 at the end of this table.)	Applies to user IDs that the federated server sends to the data source server for authentication. Valid values are: 'U' The federated server folds the user ID to uppercase before sending it to the data source. This is a logical choice for DB2 family and Oracle data sources (See note 2 at end of this table.) 'N' The federated server does nothing to the user ID before sending it to the data source. (See note 2 at end of this table.) 'L' The federated server folds the user ID to lowercase before sending it to the data source. If none of these settings are used, the federated server tries to send the user ID to the data source in uppercase. If the user ID fails, the server tries sending it in lowercase.	None.

Table 148. Server options and their settings (continued)

Option	Description and valid settings	Default setting
FOLD_PW (See notes 1, 3 and 4 at the end of this table.)	Applies to passwords that the federated server sends to data sources for authentication. Valid values are: ’U’ The federated server folds the password to uppercase before sending it to the data source. This is a logical choice for DB2 family and Oracle data sources. ’N’ The federated server does nothing to the password before sending it to the data source. ’L’ The federated server folds the password to lowercase before sending it to the data source. If none of these settings are used, the federated server tries to send the password to the data source in uppercase. If the password fails, the server tries sending it in lowercase.	None.
HMMPFAM_OPTIONS	Specifies hmmpfam options such as --null2, --pvm, and --xnu that have no corresponding column name in a reference table that maps options to column names. For example: `HMMPFAM_OPTIONS '--xnu --pvm'` In this example, the daemon runs the HMMPFAM program with options from the WHERE clause of the query, plus the additional options --xnu --pvm.	
HMMSEARCH_ OPTIONS	Allows the user to provide additional command line options to the hmmsearch command. Only valid with type SEARCH. See the HMMER User’s Guide for more information.	None.
IFILE	Specifies the path and name of the Sybase Open Client interfaces file. On Windows NT federated servers, the default is %DB2PATH%\interfaces. On UNIX federated servers, the default path and name value is `$DB2INSTANCE/sqllib/interfaces`.	None.

Table 148. Server options and their settings (continued)

Option	Description and valid settings	Default setting
INFORMIX_LOCK_ MODE	Specifies the lock mode to be set for an Informix data source. The Informix wrapper issues the 'SET LOCK MODE' command immediately after establishing the connection to an Informix data source. Valid values are: **'W'** Sets the Informix lock mode to WAIT. If the wrapper tries to access a locked table or row, Informix waits until the lock is released. **'N'** Sets the Informix lock mode to NOWAIT. If the wrapper tries to access a locked table or row, Informix returns an error. **'n'** Sets the Informix lock mode to WAIT *n* seconds. If the wrapper tries to access a locked table or row and the lock is not released within the specified number of seconds, Informix returns an error.	'W'
IO_RATIO	Denotes how much faster or slower a data source I/O system runs than the federated server I/O system. Valid values are greater than 0 and less than 1x10^{23} . Values can be expressed in any valid REAL notation. A setting of 1 indicates that the DB2 federated I/O speed and the data source I/O speed have the same I/O speed, a 1:1 ratio. A setting of .5 indicates that the DB2 federated I/O speed is 50% slower than the data source I/O speed. A setting of 2 indicates that the DB2 federated I/O speed is twice as fast as the data source I/O speed.	'1.0'
IUD_APP_SVPT_ ENFORCE	Specifies whether the DB2 federated system should enforce detecting or building of application savepoint statements. When set using the SET SERVER OPTION statement, this server option will have no effect with static SQL statements. 'Y' The federated server rolls back insert, update, or delete transactions if an error occurs in an insert, update, or delete operation and the data source does not enforce application savepoint statements. SQL error code SQL1476N is returned. 'N' The federated server will not roll back transactions when an error is encountered. Your application must handle the error recovery.	'Y'
LOGIN_TIMEOUT	Specifies the number of seconds for the DB2 federated server to wait for a response from Sybase Open Client to the login request. The default values are the same as for TIMEOUT.	'0'

Table 148. Server options and their settings (continued)

Option	Description and valid settings	Default setting
MAX_ROWS	Specifies the number of rows that the federated server returns for a query that uses the Entrez wrapper. You can specify only positive numbers and zero. When you set the option to be zero, you enable queries to retrieve an unlimited number of rows from the NCBI Web site. However, setting the MAX_ROWS server option to zero or to a very high number can impact your query performance. The MAX_ROWS server option is not required.	Microsoft Windows operating systems: 2000 rows. UNIX-based operating systems: 5000 rows.
MQ_CONN_NAME	The hostname or network address of the computer where the Websphere MQ server is running. An example of a connection name is: 9.30.76.151(1420) where 1420 is the port number. If the port number is excluded a default value of 1414 will be used. This option is optional. If it is omitted, the MQSERVER environment variable (if sepcified in db2dj.ini file) is used to select the channel definition. If MQSERVER is not set, the client channel table is used.	The wrapper uses the MQSERVER environment variable, if specified in the db2dj.ini file, to select the channel definition. If the MQSERVER environment variable is not set, the wrapper uses the client channel table.
MQ_MANAGER	The name of the WebSphere MQ manager. Any valid WebSphere MQ manager name. This option is required.	None.
MQ_RESPONSE_ TIMEOUT	The amount of time that the wrapper should wait for a response message from the response queue. The value is in milliseconds. You can specify a special value of -1 to indicate that there is no timeout period. This option is optional.	10000
MQ_SVRCONN_ CHANNELNAME	The name of the server-connection channel on the Webspehere MQ Manager that the wrapper should try to connect to. This parameter can be specified only if the MQ_CONN_NAME server option is specified. The default server-connection channel, SYSTEM.DEF.SVRCONN, is used if this option is omitted.	SYSTEM.DEF.SVRCONN
NODE	Relational data sources: Name by which a data source is defined as an instance to its RDBMS. Documentum: Specifies the actual name of the Documentum Docbase. This option is required. BLAST: Specifies the host name of the system on which the BLAST daemon process is running. This option is required. HMMER: Specifies the host name of the server on which the HMMER daemon process runs. This option is required. BioRS: Specifies the host name of the system on which the BioRS query tool is available. This option is optional.	BioRS: *localhost*

Table 148. Server options and their settings (continued)

Option	Description and valid settings	Default setting
OS_TYPE	Specifies the Docbase server's operating system. Valid values are AIX, SOLARIS, and WINDOWS. This option is required.	None.
PACKET_SIZE	Specifies the packet size of the Sybase interfaces file in bytes. If the data source does not support the specified packet size, the connection will fail. Increasing the packet size when each record is very large (for example, when inserting rows into large tables) significantly increases performance. The byte size is a numeric value.	
PASSWORD	Specifies whether passwords are sent to a data source. 'Y' Passwords are sent to the data source and validated. 'N' Passwords are not sent to the data source and not validated.	'Y'
PLAN_HINTS	Specifies whether *plan hints* are to be enabled. Plan hints are statement fragments that provide extra information for data source optimizers. This information can, for certain query types, improve query performance. The plan hints can help the data source optimizer decide whether to use an index, which index to use, or which table join sequence to use. 'Y' Plan hints are to be enabled at the data source if the data source supports plan hints. 'N' Plan hints are not to be enabled at the data source. This option is only available for Oracle and Sybase data sources.	'N'
PORT	Specifies the number of the port the wrapper uses to connect to the BioRS server. This option is optional.	'5014'
PROCESSORS	Specifies the number of processors that the HMMER program uses. This option is equivalent to the --cpu option of the `hmmpfam` command.	None.
PROXY_AUTHID	Specifies the user name to use when the value of PROXY_TYPE is 'SOCKS5'. This field is optional if the value of PROXY_TYPE is 'SOCKS5'. Contact your network administrator for the user name to use. This option is invalid if the PROXY_TYPE is not 'SOCKS5'.	None.
PROXY_PASSWORD	Specifies the password to use when the value of PROXY_TYPE is 'SOCKS5'. This field is optional if the value of PROXY_TYPE is 'SOCKS5'. Contact your network administrator for the password to use. This option is invalid if the PROXY_TYPE is not 'SOCKS5'.	None.

Table 148. Server options and their settings (continued)

Option	Description and valid settings	Default setting
PROXY_SERVER_ NAME	Specifies the proxy server name or the IP address. This field is required if the value of PROXY_TYPE is ’HTTP’, ’SOCKS4’, or ’SOCKS5’. Contact your network administrator for the proxy server name or the IP address.	None.
PROXY_SERVER_ PORT	Specifies the proxy server port number. This field is required if the value of PROXY_TYPE is ’HTTP’, ’SOCKS4’, or ’SOCKS5’. Contact your network administrator for the proxy server port number that should be used.	None.
PROXY_TYPE	Specifies the proxy type that is used to access the Internet when behind a firewall. The valid values are ’NONE’, ’HTTP’, ’SOCKS4’, or ’SOCKS5’. The default value is ’NONE’. Contact your network administrator for the type of proxy that is used.	’NONE’
PUSHDOWN	’Y’ DB2 UDB will consider letting the data source evaluate operations. ’N’ DB2 UDB will send the data source SQL statements that include only SELECT with column names. Predicates (such as WHERE=) column and scalar functions (such as MAX and MIN), sorts (such as ORDER BY or GROUP BY), and joins will not be included in any SQL sent to the data source.	’Y’
RDBMS_TYPE	Specifies the RDBMS used by the Docbase. Valid values are DB2, INFORMIX, ORACLE, SQLSERVER or SYBASE. This option is required.	None.
RESPONSE_QUEUE	The name of the response queue that delivers query results from the adapter to the wrapper. The name must conform to the specifications for queue names for WebSphere MQ. This option is required.	None.
REQUEST_QUEUE	The name of the request queue that delivers query requests from the wrapper to the adapter. The name must conform to the specifications for queue names for WebSphere MQ. This option is required.	None.
SOCKET_TIMEOUT	Specifies the maximum time in minutes that the DB2 federated server will wait for results from the proxy server. A valid value is any number that is greater than or equal to zero. The default is zero ’0’. A value of zero denotes an unlimited amount of time to wait.	0
TIMEFORMAT	The time format used by the data source. Enter the format using ’hh12’, ’hh24’, ’mm’, ’ss’, ’AM’, or ’A.M’. For example, to represent the time format of ’16:00:00’, use ’hh24:mm:ss’. To represent the time format of ’8:00:00 AM’, use ’hh12:mm:ss AM’. This field is nullable.	None.

Table 148. Server options and their settings (continued)

Option	Description and valid settings	Default setting
TIMESTAMPFORMAT	The timestamp format used by the data source. The format follows that for date and time, plus 'n' for tenth of a second, 'nn' for hundredth of a second, 'nnn' for milliseconds, and so on, up to 'nnnnnn' for microseconds. For example, to represent the timestamp format of '2003-01-01-24:00:00.000000', use 'YYYY-MM-DD-hh24:mm:ss.nnnnnn'. This field is nullable.	None.
TIMEOUT	Sybase: Specifies the number of seconds the DB2 federated server will wait for a response from Sybase Open Client for any SQL statement. The value of *seconds* is a positive whole number in DB2 Universal Database's integer range. The timeout value that you specify depends on which wrapper you are using. The default behavior of the TIMEOUT option for the Sybase wrappers is 0, which causes DB2 UDB to wait indefinitely for a response. BioRS: Specifies the time, in minutes, that the BioRS wrapper should wait for a response from the BioRS server. The default value is 10. This option is optional.	'0'; BioRS: '10'
TRANSACTIONS	Specifies the server transaction mode. The valid values are: 'NONE' No transactions are enabled. 'QUERY' Transactions are enabled only for Dctm_Query methods. **'ALL'** Transactions are enabled for the Dctm_Query method. ALL has the same function as QUERY in this release.	'QUERY'
USE_CLOB_ SEQUENCE	This option specifies the data type the federated server uses for the BlastSeq or HmmQSeq column. The values can be 'Y' or 'N'. You can use the CREATE NICKNAME or ALTER NICKNAME statement. to override the default data type for the BlastSeq or HmmQSeq column.	'Y'

Table 148. Server options and their settings (continued)

Option	Description and valid settings	Default setting
VARCHAR_NO_ TRAILING_BLANKS	This option applies to data sources which have variable character data types that do not pad the length with trailing blanks during comparison. Some data sources, such as Oracle, do not have blank-padded character comparison semantics that return the same results as the DB2 for Linux, UNIX, and Windows comparison semantics. Set this option when you want it to apply to all the VARCHAR and VARCHAR2 columns in the data source objects that will be accessed from the designated server. This includes views. Y — Trailing blanks are absent from these VARCHAR columns, or the data source has blank-padded character comparison semantics that are similar to the semantics on the federated server. The federated server pushes down character comparison operations to the data source for processing. N — Trailing blanks are present in these VARCHAR columns and the data source has blank-padded character comparison semantics that are different than the federated server. The federated server processes character comparison operations if it is not possible to compensate for equivalent semantics. For example, rewriting the predicate.	N for affected data sources.

Notes on this table:

1. This field is applied regardless of the value specified for authentication.
2. Because DB2 UDB stores user IDs in uppercase, the values ‘N’ and ‘U’ are logically equivalent to each other.
3. The setting for FOLD_PW has no effect when the setting for password is ‘N’. Because no password is sent, case cannot be a factor.
4. Avoid null settings for either of these options. A null setting can seem attractive because DB2 UDB will make multiple attempts to resolve user IDs and passwords; however, performance might suffer (it is possible that DB2 UDB will send a user ID and password four times before successfully passing data source authentication).

Related concepts:

- “Server characteristics affecting pushdown opportunities” in the *Federated Systems Guide*
- “Server characteristics affecting global optimization” in the *Federated Systems Guide*

Related tasks:

- “Registering server definitions for a data source” on page 61

Related reference:

- “DROP statement” in the *SQL Reference, Volume 2*
- “ALTER SERVER statement” in the *SQL Reference, Volume 2*
- “CREATE SERVER statement” in the *SQL Reference, Volume 2*

Appendix E. User mapping options for federated systems

These options are valid for all relational data sources. For nonrelational data sources, the REMOTE_AUTHID and REMOTE_PASSWORD options are valid for the following data sources: BioRS, Documentum, Extended Search, and Web services. The GUEST option is valid for the BioRS data source.

These options are used with the CREATE USER MAPPING and ALTER USER MAPPING statements.

Table 149. User mapping options and their settings

Option	Valid settings	Default setting
ACCOUNTING	DRDA: Used to specify a DRDA accounting string. Valid settings include any string of length 255 or less. This option is required only if accounting information needs to be passed. See the DB2 Connect Users Guide for more information.	None
GUEST	Specifies if the wrapper is to use the guest access mode to the BioRS server. **Y** The wrapper uses the guest access mode to the BioRS server. **N** The wrapper does not use the guest access mode to the BioRS server. When set to a value of Y, this option is mutually exclusive with the REMOTE_AUTHID option and the REMOTE_PASSWORD option.	N
REMOTE_AUTHID	Indicates the authorization ID used at the data source. Valid settings include any string of length 255 or less.	The authorization ID you use to connect to the DB2 Universal Database.
REMOTE_DOMAIN	Documentum: Indicates the Windows NT domain used to authenticate users connecting to a Documentum data source. Valid settings include any valid Windows NT domain name.	The default authentication domain for the Documentum database.
REMOTE_PASSWORD	Indicates the authorization password used at the data source. Valid settings include any string of length 32 or less. You do not need to set this option if the following conditions are met: • The database manager configuration parameter AUTHENTICATON is set to SERVER. • When you connected to the DB2 database, you specified an auth ID and password. If your server requires a password and you do not set this option, you must ensure both of the previous conditions are met or the connection will fail.	The password you use to connect to the DB2 Universal Database if both conditions listed in the valid settings column are met.

Related concepts:

- "DB2 Connect and DRDA" in the *DB2 Connect User's Guide*
- "DRDA and data access" in the *DB2 Connect User's Guide*

Related tasks:

- “Registering user mappings for a data source” on page 63

Appendix F. Nickname options for federated systems

Table 150 and Table 151 list the nickname options for each data source. Table 152 on page 594 describes each nickname option and lists the valid and default settings.

Table 150. Available nickname options – A through P

Data source	ALL_VERSIONS	APPLICATIONID	BUSOBJ_NAME	CATEGORY	COLUMN_DELIMITER	DATASOURCE	DIRECTORY_PATH	FILE_PATH	FOLDERS	HMMTYPE	INSTANCE_PARSE_TIME	IS_REG_TABLE	KEY_COLUMN	MAXHIT	NAMESPACES	NEXT_TIME	PARENT	PROCESSORS
BioRS																		
BLAST						X												X
Documentum	X								X			X						
Entrez																	X	
Excel								X										
Extended Search		X		X		X								X				
HMMER						X				X								
Table-structured files					X			X					X					
Web services															X			
WebSphere Business Integration			X												X			
XML							X	X			X					X		

Table 151 lists the nickname options, R through X, for each data source.

Table 151. Available nickname options – R through X

Data Source	RANGE	REMOTE_OBJECT	SOAPACTION	SORTED	SORTFIELD	SORTORDER	STREAMING	TEMPLATE	TOTALMAXHIT	TIMEOUT	URL	VALIDATE	VALIDATE_DATA_FILE	VERTICAL_TABLE	XPATH	XPATH_EVAL_TIME
BioRS		X														
BLAST										X						
Documentum		X														
Entrez		X														
Excel	X															
Extended Search					X	X			X	X				X		
HMMER										X						

Table 151. Available nickname options – R through X (continued)

Data Source	RANGE	REMOTE_OBJECT	SOAPACTION	SORTED	SORTFIELD	SORTORDER	STREAMING	TEMPLATE	TOTALMAXHIT	TIMEOUT	URL	VALIDATE	VALIDATE_DATA_FILE	VERTICAL_TABLE	XPATH	XPATH_EVAL_TIME
Table-structured files				X									X			
Web services			X				X	X			X				X	
WebSphere Business Integration								X							X	
XML							X					X			X	X

Table 152 describes each nickname option and lists the valid and default settings.

Table 152. Nickname options and their settings

Option	Description and valid settings	Default setting
ALL_VERSIONS	Specifies whether all object versions will be searched. The valid values are y, Y, n, and N. The default value of N means that only the current object versions are included in query processing. This option is invalid when IS_REG_TABLE = 'Y'.	N
APPLICATIONID	Specifies the name of the Extended Search application that you want to search. This name must exist in the Extended Search configuration database. This option is required.	
BUSOBJ_NAME	The name of the XML schema definition file (.xsd) that represents the business object. For example sap_bapi_customer_get_detail2 . This option must be specified in a parent nickname.	
CATEGORY	Specifies one or more Extended Search categories that you want to search. If you omit this option, you must specify at least one data source name. To specify multiple categories, delimit the category names with a semicolon.	
COLUMN_DELIMITER	The delimiter that is used to separate columns of a table-structured file, enclosed in single quotation marks. The column delimiter can be more than one character in length. If no column delimiter is defined, the default delimiter is a comma. A single quotation mark cannot be used as a delimiter. The column delimiter must be consistent throughout the file. A null value is represented by two delimiters next to each other or a delimiter followed by a line terminator, if the NULL field is the last one on the line. The column delimiter cannot exist as valid data for a column.	The default delimiter is a comma.

Table 152. Nickname options and their settings (continued)

Option	Description and valid settings	Default setting
DATASOURCE	For Extended Search: Specifies one or more Extended Search data sources that you want to search. If you omit this option, you must specify at least one category name. To specify multiple data sources, delimit the data source names with a semicolon. For BLAST: The name of the data source on which the BLAST search will run. The same string that is used here must be present in the configuration file of the BLAST daemon. This option is required. For HMMER (type PFAM): The name of the HMM Profile database that is to be searched by HMMPFAM. The same string that is used here must be present in the configuration file of the HMMER daeamon. This option is required. For HMMER (type SEARCH): The name of the sequence file that is to be searched by HMMSEARCH. The same string that is used here must be present in the configuration file of the HMMER daeamon. This option is required.	
DIRECTORY_PATH	Specifies the path name of a directory that contains one or more XML files. Use this option to create a single nickname over multiple XML source files. The XML wrapper uses only the files with an .xml extension that are located in the directory that you specify. The XML wrapper ignores all other files in this directory. If you specify this nickname option, do not specify a DOCUMENT column. This option is accepted only for the root nickname (the nickname that identifies the elements at the top level of the XML document).	
FILE_PATH	For Microsoft Excel: Specifies the fully qualified directory path and file name of the Excel spreadsheet that you want to access. This option is required. For table-structured files: The fully qualified path to the table-structured file to be accessed, enclosed in single quotation marks. The data file must be a standard file or a symbolic link, rather then a pipe or another non-standard file type. Either the FILE_PATH or the DOCUMENT nickname column option must be specified. If the FILE_PATH nickname option is specified, then no DOCUMENT nickname column option can be specified. For XML: Specifies the file path of the XML document. If you specify this nickname option, do not specify a DOCUMENT column. This option is accepted only for the root nickname (the nickname that identifies the elements at the top level of the XML document).	

Table 152. Nickname options and their settings (continued)

Option	Description and valid settings	Default setting
FOLDERS	Specifies a string that contains one or more logically combined and syntactically correct Documentum FOLDER predicates. Specifying FOLDER predicates restricts the set of documents that are represented by this nickname to the documents in the designated folders. When you specify this option, enclose the entire value of the FOLDERS option in single quotation marks and use double quotation marks in place of the single quotation marks within the string. This option is not valid when IS_REG_TABLE = 'Y'.	
HMMTYPE	Optional: The alphabet that is used in both models and gene sequences. The value can be either NUCLEIC or PROTEIN and is not case sensitive.	PROTEIN
INSTANCE_PARSE_TIME	Specifies the time (in milliseconds) to parse the data in one row of the XML source document. You can modify the INSTANCE_PARSE_TIME, XPATH_EVAL_TIME, and NEXT_TIME options to optimize queries of large or complex XML source structures. This option is accepted only for columns of the root nickname (the nickname that identifies the elements at the top level of the XML document). The number that you specify can be an integer or a decimal value.	7
IS_REG_TABLE	Specifies whether the object that is specified by the REMOTE_OBJECT option is a Documentum registered table. The valid values are 'y', 'Y', 'n', and 'N'. You cannot change a nickname from a Documentum object to a registered table (or back) by changing this option with the ALTER NICKNAME statement. Instead, you must drop and recreate the nickname.	N
KEY_COLUMN	The name of the column in the file that forms the key on which the file is sorted, enclosed in single quotation marks. Use this option for sorted files only. A column that is designated with the DOCUMENT nickname column option must not be specified as the key column. Only single-column keys are supported. Multi-column keys are not allowed. The value must be the name of a column that is defined in the CREATE NICKNAME statement. The column must be sorted in ascending order. The key column must be designated not nullable by adding the NOT NULL option to its definition in the nickname statement. This option is case-sensitive. However, DB2 UDB changes column names to uppercase unless the column is defined with double quotation marks.	If the value is not specified for a sorted nickname, the value is the name of the first column in the nicknamed file.

Table 152. Nickname options and their settings (continued)

Option	Description and valid settings	Default setting
MAXHIT	An INTEGER that specifies the maximum number of results that can be returned from each source that is being searched.	50

NAMESPACES	The namespaces that are associated with the namespace prefixes that is used in the XPATH and TEMPLATE options for each column. The syntax is:
	`NAMESPACES 'prefix1=`
	`"actual_namespace1",`
	`prefix2="actual_namespace2" '`
	
	Separate each namespace with a comma. For example:
	`NAMESPACES '`
	`c="http://www.myweb.com/cust",`
	`i="http://www.myweb.com/cust/id",`
	`n="http://www.myweb.com/cust/name"'`	

NEXT_TIME	Specifies the time (in milliseconds) that is required to locate subsequent source elements from the XPath expression. You can modify the NEXT_TIME, XPATH_EVAL_TIME, and INSTANCE_PARSE_TIME options to optimize queries of large or complex XML source structures. This option is accepted for root nicknames and non-root nicknames.	1
PARENT	Specified only for a child nickname whose parent was renamed through the REMOTE_OBJECT option. The PARENT option associates a child with a parent when multiple nickname families are defined within a DB2 schema. This name is case-sensitive.	
PROCESSORS	Specifies the number of processors to be used when a BLAST query is evaluated. This option corresponds to the blastall -a option.	1
RANGE	Specifies a range of cells to be used in the data source.	

Table 152. Nickname options and their settings (continued)

Option	Description and valid settings	Default setting
REMOTE_OBJECT	For BioRS: Specifies the name of the BioRS databank that is associated with the nickname. This name determines the schema and the BioRS databank for the nickname. This name also specifies the relationship of the nickname to other nicknames. The case sensitivity of this name depends on the case sensitivity of the BioRS server and on the value of the CASE_SENSITIVE server option. You cannot use the ALTER NICKNAME statement to change or delete this name. If the name of the BioRS databank that is used in this option changes, you must delete and then create the entire nickname again. For Documentum: Specifies the name of the Documentum object type that is associated with the nickname. The name can be any Documentum object type or registered table. The name of a registered table must be prefixed by the table owner's name. If the registered table belongs to the Docbase owner, the value dm_dbo can be used for the owner name. This option is required. Using the ALTER NICKNAME statement to change the value of the REMOTE_OBJECT option results in errors if the structure of the new object is not similar to that of the original object. For Entrez: Specifies the name of the Entrez object type that is associated with the nickname. This name determines the schema and NCBI database for the nickname and its relationship to other nicknames. This name is case insensitive.	
SOAPACTION	The URI SOAPACTION attribute from the Web Service Description Language (WSDL) format. This option is required for the root nickname. This option is not allowed with nonroot nicknames.	
SORTED	Specifies whether the data source file is sorted or unsorted. This option accepts either Y, y, n, or N. Sorted data sources must be sorted in ascending order according to the collation sequence for the current locale, as defined by the settings in the LC_COLLATE National Language Support category. If you specify that the data source is sorted, set the VALIDATE_DATA_FILE option to Y.	N
SORTFIELD	Specifies the name of a field on which search results should be sorted. The default value, DOC_RANK, is a field that Extended Search uses to determine the relevancy of a result document. If you specify a different field name, that name must exist in the sources that you search.	DOC_RANK

Table 152. Nickname options and their settings (continued)

Option	Description and valid settings	Default setting
SORTORDER	Specifies a sort order for the return of search results, either ascending (A) or descending (D).	A
STREAMING	Specifies whether the XML source document should be separated into logical fragments for processing. The fragments correspond to the node that matches the XPath expression of the nickname. The wrapper then parses and processes the XML source data fragment by fragment. This type of parsing minimizes memory usage. This option is specified on only the root nickname. You can specify streaming for any XML source document (FILE, DIRECTORY, URI, or COLUMN). This option is accepted only for columns of the root nickname (the nickname that identifies the elements at the top level of the XML document). Valid values are: Y The XML documents are parsed. N The XML documents are not parsed. Do not set the STREAMING parameter to YES if you set the VALIDATE parameter to YES. If you set both parameters to YES, you will receive an error message.	N
TEMPLATE	For WebSphere Business Integration: The nickname template fragment to use to construct an XML input document. The fragment must conform to the specified template syntax. For Web Services: The nickname template fragment to use to construct a SOAP request. The fragment must conform to the specified template syntax.	
TOTALMAXHIT	An INTEGER that specifies the maximum number of results that can be returned from all the sources that are being searched. The wrapper combines these results into a single result set.	50
TIMEOUT	For Extended Search: An INTEGER that specifies the number of seconds to wait for a response from a server before the request times out. For BLAST and HMMER: The maximum time, in minutes, that the wrapper waits for results from the daemon.	For Extended Search: 30. For BLAST and HMMER: 60.
URL	The URL for the Web service endpoint. This option is required for the root nickname. This option is not allowed with nonroot nicknames. Supported protocols are HTTP and HTTPS.	

Table 152. Nickname options and their settings (continued)

Option	Description and valid settings	Default setting
VALIDATE	Specifies whether the XML source document is validated before the XML data is extracted. If this option is set to YES, the nickname option verifies that the structure of the source document conforms to an XML schema or to a document type definition (DTD). This option is accepted only for columns of the root nickname (the nickname that identifies the elements at the top level of the XML document). The XML source document is not validated if the XML wrapper cannot locate the XML schema file or DTD file (.xsd or .dtd). DB2 UDB does not issue an error message if the validation does not occur. Therefore, ensure that the XML schema file or DTD file exists in the location that is specified in the XML source document. Do not set the VALIDATE parameter to YES if you set the STREAMING parameter to YES. If you set both parameters to YES, you will receive an error message.	NO
VALIDATE_DATA_FILE	For sorted files, this option specifies whether the wrapper verifies that the key column is sorted in ascending order and checks for null keys. The only valid values for this option are Y or N. The check is done once at registration time. This option is not allowed if the DOCUMENT nickname column option is used for the file path.	N
VERTICAL_TABLE	Specifies the presentation format for search results. If you specify YES, Extended Search returns all fields that are configured as returnable, in addition to the user-defined columns. The wrapper stores the results in the nickname table as a vertical list of column names.	NO
XPATH	Specifies the XPATH expression that identifies the elements that represent the individual tuples. The XPATH nickname option for a child nickname is evaluated in the context of the path that is specified by the XPATH nickname option of its parent. This XPATH expression is used as a context for evaluating column values that are identified by the XPATH nickname column options. For XML: Do not specify a namespace prefix in an XPATH expression. The XML wrapper does not support namespaces.	

Table 152. Nickname options and their settings (continued)

Option	Description and valid settings	Default setting
XPATH_EVAL_TIME	Specifies the time (in milliseconds) to evaluate the XPath expression of the nickname and to locate the first element. You can modify the XPATH_EVAL_TIME, INSTANCE_PARSE_TIME, and NEXT_TIME options to optimize queries of large or complex XML source structures. This option is accepted for root nicknames and nonroot nicknames. The number that you specify can be an integer or a decimal value.	1

Appendix G. Nickname column options for federated systems

You can specify column information in the CREATE NICKNAME or ALTER NICKNAME statements using parameters called nickname column options.

The following table lists the nickname column options for each data source.

Table 153. Available nickname column options

Data source	ALL_VALUES	DEFAULT	DELIMITER	DOCUMENT	ESCAPE_INPUT	FOREIGN_KEY	INDEX	IS_REPEATING	NUMERIC_STRING	PRIMARY_KEY	REMOTE_NAME	SOAPACTIONCOLUMN	TEMPLATE	URLCOLUMN	VARCHAR_NO_TRAILING_BLANKS	XPATH
BLAST		X	X				X									
DB2 Universal Database for iSeries									X							
DB2 Universal Database for z/OS and OS/390									X							
DB2 Universal Database for VM and VSE									X							
DB2 Universal Database for Linux, UNIX, and Windows									X							
Documentum	X		X					X			X					
Informix									X							
Microsoft SQL Server									X							
ODBC									X							
OLE DB																
Oracle									X						X	
Sybase									X							
Table-structured files				X												
Teradata									X							
WebSphere Business Integration					X	X				X			X			X

Table 153. Available nickname column options (continued)

Data source	ALL_VALUES	DEFAULT	DELIMITER	DOCUMENT	ESCAPE_INPUT	FOREIGN_KEY	INDEX	IS_REPEATING	NUMERIC_STRING	PRIMARY_KEY	REMOTE_NAME	SOAPACTIONCOLUMN	TEMPLATE	URLCOLUMN	VARCHAR_NO_TRAILING_BLANKS	XPATH
Web services					X	X				X		X	X	X		X
XML				X		X				X						X

Table 154. Column options and their settings

Option	Description and valid settings	Default setting
ALL_VALUES	Specifies that all values of a repeating attribute will be returned, separated by the specified delimiter. If this option is missing or is N, then only the last value of a repeating attribute is returned. The ALL_VALUES option can be specified only for VARCHAR columns for which the IS_REPEATING option is 'Y' (and is invalid when IS_REG_TABLE = 'Y').	
DEFAULT	Specifies a new default value for the following input fixed columns: • E_value • QueryStrands • GapAlign • NMisMatchPenalty • NMatchReward • Matrix • FilterSequence • NumberOfAlignments • GapCost • ExtendedGapCost • WordSize • ThresholdEx This new value overrides the preset default values. The new default value must be of the same type as the indicated value for a given column.	

Table 154. Column options and their settings (continued)

Option	Description and valid settings	Default setting
DELIMITER	For Documentum: Specifies the delimiter string to be used when multiple values of a repeating attribute are concatenated. The delimiter can be one or more characters. This option is valid only for attributes of objects with data type VARCHAR where the IS_REPEATING option is set to Y. For BLAST: The delimiter characters to be used to determine the end point of the definition line information for the column on which this option appears. If more than one character appears in this option's value, then the first occurrence of any one of the characters signals the end of this field's information. The default is end of line. This option is required, unless you want the last specified column to contain the remainder of the definition line.	For Documentum: The default delimiter is a comma. For BLAST: The default delimiter is end of line.

Table 154. Column options and their settings (continued)

Option	Description and valid settings	Default setting
DOCUMENT	For table-structured files: Specifies the kind of table-structured file. This wrapper supports only the value FILE for this option. Only one column can be specified with the DOCUMENT option per nickname. The column that is associated with the DOCUMENT option must be a data type of VARCHAR or CHAR. Using the DOCUMENT nickname column option instead of the FILE_PATH nickname option implies that the file that corresponds to this nickname will be supplied when the query runs. If the DOCUMENT option has the FILE value, the value that is supplied when the query runs is the full path of the file whose schema matches the nickname definition for this nickname. For XML: Specifies that this column is a DOCUMENT column. The value of the DOCUMENT column indicates the type of XML source data that is supplied to the nickname when the query runs. This option is accepted only for columns of the root nickname (the nickname that identifies the elements at the top level of the XML document). Only one column can be specified with the DOCUMENT option per nickname. The column that is associated with the DOCUMENT option must be a VARCHAR data type. If you use a DOCUMENT column option instead of the FILE_PATH or DIRECTORY_PATH nickname option the document that corresponds to this nickname is supplied when the query runs. The valid values for the DOCUMENT option are: **FILE** Specifies that the value of the nickname column is bound to the path name of a file. The data from this file is supplied when the query runs. **DIRECTORY** Specifies that the value of the nickname column is bound to the path name of a directory that contains multiple XML data files. The XML data from multiple files is supplied when the query runs. The data is in XML files in the specified directory path. The XML wrapper uses only the files with an .xml extension that are located in the directory that you specify. The XML wrapper ignores all other files in this directory. **URI** Specifies that the value of the nickname column is bound to the path name of a remote XML file to which a URI refers. The URI address indicates the remote location of this XML file on the Web. **COLUMN** Specifies that the XML document is stored in a relational column.	

Table 154. Column options and their settings (continued)

Option	Description and valid settings	Default setting
ELEMENT_NAME	Specifies the BioRS element name. The case sensitivity of this name depends on the case sensitivity of the BioRS server and on the value of the CASE_SENSITIVE server option. You must specify the BioRS element name only if it is different from the column name.	
ESCAPE_INPUT	Specifies whether XML special characters are replaced in XML input values or not. Use this option to include XML fragments as input, such as XML fragments with repeating elements. The TEMPLATE column option must be defined on columns that use the ESCAPE_INPUT column option. The column data type must be VARCHAR or CHAR. Valid values are: Y If the XML input contains special characters these are replaced with their counterpart characters that XML uses to represent the input characters. N Input characters are preserved exactly as they appear.	Y
FOREIGN_KEY	Indicates that this nickname is a child nickname and specifies the name of the corresponding parent nickname. A nickname can have at most one FOREIGN_KEY column option. The value for this option is case sensitive. The table that is designated with this option holds a key that is generated by the wrapper. The XPATH option must not be specified for this column. The column can be used only to join parent nicknames and child nicknames. A CREATE NICKNAME statement with a FOREIGN_KEY option will fail if the parent nickname has a different schema name. Unless the nickname that is referred to in a FOREIGN_KEY clause was explicitly defined as lowercase or mixed case by enclosing it in quotation marks in the corresponding CREATE NICKNAME statement, then when you refer to this nickname in the FOREIGN_KEY clause, you must specify the nickname in uppercase. When this option is set on a column, no other option can be set on the column.	
INDEX	The ordinal number of the column on which this option appears in the group of definition line columns. This option is required.	
IS_INDEXED	Indicates whether the corresponding column is indexed (whether the column can be referenced in a predicate). The valid values are Y and N. The value Y can be specified only for columns whose corresponding element is indexed by the BioRS server.	When a nickname is created, this option is automatically added with the value Y to any columns that correspond to a BioRS indexed element.

Table 154. Column options and their settings (continued)

Option	Description and valid settings	Default setting
IS_REPEATING	Indicates whether the column is multi-valued. Valid values are Y and N. Only the last value is returned for: • Non-VARCHAR repeating attributes • VARCHAR columns when ALL_VALUES 'N' is specified To overcome this limitation, you can create a dual definition for the repeating attribute column.	N
NUMERIC_STRING	Specifies whether a column contains strings of numeric characters. Y This column contains strings of numeric characters '0', '1', '2', ... '9'. It does not contain blanks. If this column contains only numeric strings followed by trailing blanks, do not specify Y. When you set NUMERIC_STRING to Y for a column, you are informing the optimizer that this column contains no blanks that could interfere with sorting of the column's data. Use this option when the collating sequence of a data source is different from the collating sequence that the federated server uses. Columns that use this option are not excluded from remote evaluation because of a different collating sequence. N This column is either not a numeric string column or is a numeric string column that contains blanks.	N
PRIMARY_KEY	Indicates that this nickname is a parent nickname. The column data type must be VARCHAR(16). A nickname can have at most one PRIMARY_KEY column option. YES is the only valid value. The column that is designated with this option holds a key that is generated by the wrapper. The XPATH option must not be specified for this column. The column can be used only to join parent nicknames and child nicknames. When this option is set on a column, no other option can be set on the column.	
REFERENCED_OBJECT	This option is valid only for columns whose BioRS data type is Reference. This option specifies the name of the BioRS databank that is referenced by the current column. The case sensitivity of this name depends on the case sensitivity of the BioRS server and on the value of the CASE_SENSITIVE server option.	
REMOTE_NAME	Specifies the name of the corresponding Documentum attribute or column. This option maps remote attribute or column names to local DB2 UDB column names.	The DB2 UDB column name.
SOAPACTIONCOLUMN	A column to dynamically specify the URI SOAPACTION attribute from the Web Service Description Language (WSDL) format. This option is specified on only the root nickname. When this option is set on a column, no other option can be set on the column.	

Table 154. Column options and their settings (continued)

Option	Description and valid settings	Default setting
TEMPLATE	The column template fragment to use to construct the XML input document. The fragment must conform to the specified template syntax.	
URLCOLUMN	A column to dynamically specify the URL for the Web service endpoint when you run a query. This option is specified on only the root nickname. When this option is set on a column, no other option can be set on the column.	
VARCHAR_NO_ TRAILING_BLANKS	This option applies to data sources that have variable character data types that do not pad the length with trailing blanks during comparison. Some data sources, such as Oracle, do not have blank-padded character comparison semantics that return the same results as the DB2 UDB for Linux, UNIX, and Windows comparison semantics. Set this option when you want it to apply only to a specific VARCHAR or VARCHAR2 column in a data source object. Y Trailing blanks are absent from these VARCHAR columns, or the data source has blank-padded character comparison semantics that are similar to the semantics on the federated server. The federated server sends character comparison operations to the data source for processing. N Trailing blanks are present in these VARCHAR columns, and the data source has blank-padded character comparison semantics that are different than the federated server. The federated server processes character comparison operations if it is not possible to compensate for equivalent semantics. For example, rewriting the predicate.	N for affected data sources
XPATH	Specifies the XPath expression in the XML document that contains the data that corresponds to this column. The wrapper evaluates the XPath expression after the CREATE NICKNAME statement applies this XPath expression from this XPATH nickname option.	

Related concepts:

- "Pushdown analysis" in the *Federated Systems Guide*

Related tasks:

- "Global optimization" in the *Federated Systems Guide*

Appendix H. Default forward data type mappings

The two kinds of mappings between data source data types and federated database data types are forward type mappings and reverse type mappings. In a *forward type mapping*, the mapping is from a remote type to a comparable local type.

You can override a default type mapping, or create a new type mapping with the CREATE TYPE MAPPING statement.

These mappings are valid with all the supported versions, unless otherwise noted.

For all default forward data types mapping from a data source to DB2 for Linux, UNIX, and Windows, the DB2 federated schema is SYSIBM.

The following tables show the default forward mappings between DB2 for Linux, UNIX, and Windows data types and data source data types.

DB2 for z/OS and OS/390 data sources

Table 155. DB2 for z/OS and OS/390 forward default data type mappings (Not all columns shown)

REMOTE_TYPENAME	REMOTE_LOWER_LEN	REMOTE_UPPER_LEN	REMOTE_LOWER_SCALE	REMOTE_UPPER_SCALE	REMOTE_BIT_DATA	REMOTE_DATA_OPERATORS	FEDERATED_TYPENAME	FEDERATED_LENGTH	FEDERATED_SCALE	FEDERATED_BIT_DATA
BLOB	-	-	-	-	-	-	BLOB	-	-	-
CHAR	1	254	-	-	-	-	CHAR	-	0	N
CHAR	255	32672	-	-	-	-	VARCHAR	-	0	N
CHAR	1	254	-	-	Y	-	CHAR	-	0	Y
CHAR	255	32672	-	-	Y	-	VARCHAR	-	0	Y
CLOB	-	-	-	-	-	-	CLOB	-	-	-
DATE	-	-	-	-	-	-	DATE	-	0	-
DBCLOB	-	-	-	-	-	-	DBCLOB	-	-	-
DECIMAL	-	-	-	-	-	-	DECIMAL	-	-	-
| FLOAT	4	-	-	-	-	-	REAL	-	-	-
| FLOAT	8	-	-	-	-	-	DOUBLE	-	-	-
GRAPHIC	1	127	-	-	-	-	GRAPHIC	-	0	N
INTEGER	-	-	-	-	-	-	INTEGER	-	0	-
| ROWID	-	-	-	-	Y	-	VARCHAR	40	-	Y
SMALLINT	-	-	-	-	-	-	SMALLINT	-	0	-

Table 155. DB2 for z/OS and OS/390 forward default data type mappings (Not all columns shown) (continued)

REMOTE_TYPENAME	REMOTE_LOWER_LEN	REMOTE_UPPER_LEN	REMOTE_LOWER_SCALE	REMOTE_UPPER_SCALE	REMOTE_BIT_DATA	REMOTE_DATA_OPERATORS	FEDERATED_TYPENAME	FEDERATED_LENGTH	FEDERATED_SCALE	FEDERATED_BIT_DATA
TIME	-	-	-	-	-	-	TIME	-	0	-
TIMESTAMP	-	-	-	-	-	-	TIMESTAMP	-	0	-
TIMESTMP	-	-	-	-	-	-	TIMESTAMP	-	0	-
VARCHAR	1	32672	-	-	-	-	VARCHAR	-	0	N
VARCHAR	1	32672	-	-	Y	-	VARCHAR	-	0	Y
VARG	1	16336	-	-	-	-	VARGRAPHIC	-	0	N
VARGRAPHIC	1	16336	-	-	-	-	VARGRAPHIC	-	0	N

DB2 for iSeries data sources

Table 156. DB2 for iSeries forward default data type mappings (Not all columns shown)

REMOTE_TYPENAME	REMOTE_LOWER_LEN	REMOTE_UPPER_LEN	REMOTE_LOWER_SCALE	REMOTE_UPPER_SCALE	REMOTE_BIT_DATA	REMOTE_DATA_OPERATORS	FEDERATED_TYPENAME	FEDERATED_LENGTH	FEDERATED_SCALE	FEDERATED_BIT_DATA
BLOB	-	-	-	-	-	-	BLOB	-	-	-
CHAR	1	254	-	-	-	-	CHAR	-	0	N
CHAR	255	32672	-	-	-	-	VARCHAR	-	0	N
CHAR	1	254	-	-	Y	-	CHAR	-	0	Y
CHAR	255	32672	-	-	Y	-	VARCHAR	-	0	Y
CLOB	-	-	-	-	-	-	CLOB	-	-	-
DATE	-	-	-	-	-	-	DATE	-	0	-
DBCLOB	-	-	-	-	-	-	DBCLOB	-	-	-
DECIMAL	-	-	-	-	-	-	DECIMAL	-	-	-
FLOAT	4	-	-	-	-	-	REAL	-	-	-
FLOAT	8	-	-	-	-	-	DOUBLE	-	-	-
GRAPHIC	1	127	-	-	-	-	GRAPHIC	-	0	N

Table 156. DB2 for iSeries forward default data type mappings (Not all columns shown) (continued)

REMOTE_TYPENAME	REMOTE_LOWER_LEN	REMOTE_UPPER_LEN	REMOTE_LOWER_SCALE	REMOTE_UPPER_SCALE	REMOTE_BIT_DATA	REMOTE_DATA_OPERATORS	FEDERATED_TYPENAME	FEDERATED_LENGTH	FEDERATED_SCALE	FEDERATED_BIT_DATA
GRAPHIC	128	16336	-	-	-	-	VARGRAPHIC	-	0	N
INTEGER	-	-	-	-	-	-	INTEGER	-	0	-
NUMERIC	-	-	-	-	-	-	DECIMAL	-	-	-
SMALLINT	-	-	-	-	-	-	SMALLINT	-	0	-
TIME	-	-	-	-	-	-	TIME	-	0	-
TIMESTAMP	-	-	-	-	-	-	TIMESTAMP	-	0	-
TIMESTMP	-	-	-	-	-	-	TIMESTAMP	-	0	-
VARCHAR	1	32672	-	-	-	-	VARCHAR	-	0	N
VARCHAR	1	32672	-	-	Y	-	VARCHAR	-	0	Y
VARG	1	16336	-	-	-	-	VARGRAPHIC	-	0	N
VARGRAPHIC	1	16336	-	-	-	-	VARGRAPHIC	-	0	N

DB2 Server for VM and VSE data sources

Table 157. DB2 Server for VM and VSE forward default data type mappings (Not all columns shown)

REMOTE_TYPENAME	REMOTE_LOWER_LEN	REMOTE_UPPER_LEN	REMOTE_LOWER_SCALE	REMOTE_UPPER_SCALE	REMOTE_BIT_DATA	REMOTE_DATA_OPERATORS	FEDERATED_TYPENAME	FEDERATED_LENGTH	FEDERATED_SCALE	FEDERATED_BIT_DATA
BLOB	-	-	-	-	-	-	BLOB	-	-	-
CHAR	1	254	-	-	-	-	CHAR	-	0	N
CHAR	1	254	-	-	Y	-	CHAR	-	0	Y
CLOB	-	-	-	-	-	-	CLOB	-	-	-
DATE	-	-	-	-	-	-	DATE	-	0	-
DBAHW	-	-	-	-	-	-	SMALLINT	-	0	-
DBAINT	-	-	-	-	-	-	INTEGER	-	0	-
DBCLOB	-	-	-	-	-	-	DBCLOB	-	-	-

Table 157. DB2 Server for VM and VSE forward default data type mappings (Not all columns shown) (continued)

REMOTE_TYPENAME	REMOTE_LOWER_LEN	REMOTE_UPPER_LEN	REMOTE_LOWER_SCALE	REMOTE_UPPER_SCALE	REMOTE_BIT_DATA	REMOTE_DATA_OPERATORS	FEDERATED_TYPENAME	FEDERATED_LENGTH	FEDERATED_SCALE	FEDERATED_BIT_DATA
DECIMAL	-	-	-	-	-	-	DECIMAL	-	-	-
FLOAT	4	-	-	-	-	-	REAL	-	-	-
FLOAT	8	-	-	-	-	-	DOUBLE	-	-	-
GRAPHIC	1	127	-	-	-	-	GRAPHIC	-	0	N
INTEGER	-	-	-	-	-	-	INTEGER	-	-	-
SMALLINT	-	-	-	-	-	-	SMALLINT	-	-	-
TIME	-	-	-	-	-	-	TIME	-	0	-
TIMESTAMP	-	-	-	-	-	-	TIMESTAMP	-	0	-
TIMESTMP	-	-	-	-	-	-	TIMESTAMP	-	0	-
VARCHAR	1	32672	-	-	-	-	VARCHAR	-	0	N
VARCHAR	1	32672	-	-	Y	-	VARCHAR	-	0	Y
VARGRAPHIC	1	16336	-	-	-	-	VARGRAPHIC	-	0	N
VARGRAPH	1	16336	-	-	-	-	VARGRAPHIC	-	0	N

DB2 for Linux, UNIX, and Windows data sources

Table 158. DB2 for Linux, UNIX, and Windows forward default data type mappings (Not all columns shown)

REMOTE_TYPENAME	REMOTE_LOWER_LEN	REMOTE_UPPER_LEN	REMOTE_LOWER_SCALE	REMOTE_UPPER_SCALE	REMOTE_BIT_DATA	REMOTE_DATA_OPERATORS	FEDERATED_TYPENAME	FEDERATED_LENGTH	FEDERATED_SCALE	FEDERATED_BIT_DATA
BIGINT	-	-	-	-	-	-	BIGINT	-	0	-
BLOB	-	-	-	-	-	-	BLOB	-	-	-
CHAR	-	-	-	-	-	-	CHAR	-	0	N
CHAR	-	-	-	-	Y	-	CHAR	-	0	Y
CLOB	-	-	-	-	-	-	CLOB	-	-	-
DATE	-	-	-	-	-	-	DATE	-	0	-

Table 158. DB2 for Linux, UNIX, and Windows forward default data type mappings (Not all columns shown) (continued)

REMOTE_TYPENAME	REMOTE_LOWER_LEN	REMOTE_UPPER_LEN	REMOTE_LOWER_SCALE	REMOTE_UPPER_SCALE	REMOTE_BIT_DATA	REMOTE_DATA_OPERATORS	FEDERATED_TYPENAME	FEDERATED_LENGTH	FEDERATED_SCALE	FEDERATED_BIT_DATA
| DBCLOB	-	-	-	-	-	-	DBCLOB	-	-	-
DECIMAL	-	-	-	-	-	-	DECIMAL	-	-	-
DOUBLE	-	-	-	-	-	-	DOUBLE	-	-	-
FLOAT	-	-	-	-	-	-	DOUBLE	-	-	-
GRAPHIC	-	-	-	-	-	-	GRAPHIC	-	0	N
INTEGER	-	-	-	-	-	-	INTEGER	-	0	-
| LONGVAR	-	-	-	-	N	-	CLOB	-	-	-
| LONGVAR	-	-	-	-	Y	-	BLOB	-	-	-
| LONGVARG	-	-	-	-	-	-	DBCLOB	-	-	-
REAL	-	-	-	-	-	-	REAL	-	-	-
SMALLINT	-	-	-	-	-	-	SMALLINT	-	0	-
TIME	-	-	-	-	-	-	TIME	-	0	-
TIMESTAMP	-	-	-	-	-	-	TIMESTAMP	-	0	-
TIMESTMP	-	-	-	-	-	-	TIMESTAMP	-	0	-
VARCHAR	-	-	-	-	-	-	VARCHAR	-	0	N
VARCHAR	-	-	-	-	Y	-	VARCHAR	-	0	Y
VARGRAPH	-	-	-	-	-	-	VARGRAPHIC	-	0	N
VARGRAPHIC	-	-	-	-	-	-	VARGRAPHIC	-	0	N

Informix data sources

Table 159. Informix forward default data type mappings (Not all columns shown)

REMOTE_TYPENAME	REMOTE_LOWER_LEN	REMOTE_UPPER_LEN	REMOTE_LOWER_SCALE	REMOTE_UPPER_SCALE	REMOTE_BIT_DATA	REMOTE_DATA_OPERATORS	FEDERATED_TYPENAME	FEDERATED_LENGTH	FEDERATED_SCALE	FEDERATED_BIT_DATA
BLOB	-	-	-	-	-	-	BLOB	2147483647	-	-
| BOOLEAN	-	-	-	-	-	-	CHARACTER	1	-	-
BYTE	-	-	-	-	-	-	BLOB	2147483647	-	-
CHAR	1	254	-	-	-	-	CHARACTER	-	-	-
CHAR	255	32672	-	-	-	-	VARCHAR	-	-	-
CLOB	-	-	-	-	-	-	CLOB	2147483647	-	-
DATE	-	-	-	-	-	-	DATE	4	-	-
DATETIME	0	4	0	4	-	-	DATE	4	-	-
DATETIME	6	10	6	10	-	-	TIME	3	-	-
DATETIME	0	4	6	15	-	-	TIMESTAMP	10	-	-
DATETIME	6	10	11	15	-	-	TIMESTAMP	10	-	-
DECIMAL	1	31	0	31	-	-	DECIMAL	-	-	-
| DECIMAL	32	130	-	-	-	-	DOUBLE	8	-	-
FLOAT	-	-	-	-	-	-	DOUBLE	8	-	-
INTEGER	-	-	-	-	-	-	INTEGER	4	-	-
| INTERVAL	-	-	-	-	-	-	VARCHAR	25	-	-
INT8	-	-	-	-	-	-	BIGINT	19	0	-
LVARCHAR	1	32672	-	-	-	-	VARCHAR	-	-	-
MONEY	1	31	0	31	-	-	DECIMAL	-	-	-
MONEY	32	32	-	-	-	-	DOUBLE	8	-	-
NCHAR	1	254	-	-	-	-	CHARACTER	-	-	-
NCHAR	255	32672	-	-	-	-	VARCHAR	-	-	-
NVARCHAR	1	32672	-	-	-	-	VARCHAR	-	-	-
REAL	-	-	-	-	-	-	REAL	4	-	-
SERIAL	-	-	-	-	-	-	INTEGER	4	-	-
SERIAL8	-	-	-	-	-	-	BIGINT	-	-	-
SMALLFLOAT	-	-	-	-	-	-	REAL	4	-	-
SMALLINT	-	-	-	-	-	-	SMALLINT	2	-	-
TEXT	-	-	-	-	-	-	CLOB	2147483647	-	-
VARCHAR	1	32672	-	-	-	-	VARCHAR	-	-	-

Table 159. Informix forward default data type mappings (Not all columns shown) (continued)

REMOTE_TYPENAME	REMOTE_LOWER_LEN	REMOTE_UPPER_LEN	REMOTE_LOWER_SCALE	REMOTE_UPPER_SCALE	REMOTE_BIT_DATA	REMOTE_DATA_OPERATORS	FEDERATED_TYPENAME	FEDERATED_LENGTH	FEDERATED_SCALE	FEDERATED_BIT_DATA

Notes:

| • For the Informix DATETIME data type, the DB2 UNIX and Windows federated server uses the Informix high-level
| qualifer as the REMOTE_LENGTH and the Informix low-level qualifier as the REMOTE_SCALE.

| The Informix qualifiers are the ″TU_″ constants defined in the Informix Client SDK `datatime.h` file. The contstants
| are:

0 = YEAR	8 = MINUTE	13 = FRACTION(3)
2 = MONTH	10 = SECOND	14 = FRACTION(4)
4 = DAY	11 = FRACTION(1)	15 = FRACTION(5)
6 = HOUR	12 = FRACTION(2)	

Microsoft SQL Server data sources

Table 160. Microsoft SQL Server forward default data type mappings

REMOTE_TYPENAME	REMOTE_LOWER_LEN	REMOTE_UPPER_LEN	REMOTE_LOWER_SCALE	REMOTE_UPPER_SCALE	REMOTE_BIT_DATA	REMOTE_DATA_OPERATORS	FEDERATED_TYPENAME	FEDERATED_LENGTH	FEDERATED_SCALE	FEDERATED_BIT_DATA
| bigint [4]	-	-	-	-	-	-	BIGINT	-	-	-
binary	1	254	-	-	-	-	CHARACTER	-	-	Y
binary	255	8000	-	-	-	-	VARCHAR	-	-	Y
bit	-	-	-	-	-	-	SMALLINT	2	-	-
char	1	254	-	-	-	-	CHAR	-	-	N
char	255	8000	-	-	-	-	VARCHAR	-	-	N
datetime	-	-	-	-	-	-	TIMESTAMP	10	-	-
datetimen	-	-	-	-	-	-	TIMESTAMP	10	-	-
decimal	1	31	0	31	-	-	DECIMAL	-	-	-
decimal	32	38	0	38	-	-	DOUBLE	-	-	-
decimaln	1	31	0	31	-	-	DECIMAL	-	-	-

Table 160. Microsoft SQL Server forward default data type mappings (continued)

REMOTE_TYPENAME	REMOTE_LOWER_LEN	REMOTE_UPPER_LEN	REMOTE_LOWER_SCALE	REMOTE_UPPER_SCALE	REMOTE_BIT_DATA	REMOTE_DATA_OPERATORS	FEDERATED_TYPENAME	FEDERATED_LENGTH	FEDERATED_SCALE	FEDERATED_BIT_DATA
decimaln	32	38	0	38	-	-	DOUBLE	-	-	-
DUMMY65 [1]	1	38	-84	127	-	-	DOUBLE	-	-	-
DUMMY2000 [3]	1	38	-84	127	-	-	DOUBLE	-	-	-
float	-	8	-	-	-	-	DOUBLE	8	-	-
floatn	-	8	-	-	-	-	DOUBLE	8	-	-
float	-	4	-	-	-	-	REAL	4	-	-
floatn	-	4	-	-	-	-	REAL	4	-	-
image	-	-	-	-	-	-	BLOB	2147483647	-	Y
int	-	-	-	-	-	-	INTEGER	4	-	-
intn	-	-	-	-	-	-	INTEGER	4	-	-
money	-	-	-	-	-	-	DECIMAL	19	4	-
moneyn	-	-	-	-	-	-	DECIMAL	19	4	-
| nchar	1	127	-	-	-	-	CHAR	-	-	N
| nchar	128	4000	-	-	-	-	VARCHAR	-	-	N
numeric	1	31	0	31	-	-	DECIMAL	-	-	-
numeric	32	38	0	38	-	-	DOUBLE	8	-	-
numericn	32	38	0	38	-	-	DOUBLE	-	-	-
numericn	1	31	0	31	-	-	DECIMAL	-	-	-
| ntext [2]	-	-	-	-	-	-	CLOB	2147483647	-	Y
| nvarchar	1	4000	-	-	-	-	VARCHAR	-	-	N
real	-	-	-	-	-	-	REAL	4	-	-
smallint	-	-	-	-	-	-	SMALLINT	2	-	-
smalldatetime	-	-	-	-	-	-	TIMESTAMP	10	-	-
smallmoney	-	-	-	-	-	-	DECIMAL	10	4	-
smallmoneyn	-	-	-	-	-	-	DECIMAL	10	4	-
SQL_BIGINT	-	-	-	-	-	-	DECIMAL	-	-	-
| SQL_BIGINT [4]	-	-	-	-	-	-	BIGINT	-	-	-
SQL_BINARY	1	254	-	-	-	-	CHARACTER	-	-	Y
SQL_BINARY	255	8000	-	-	-	-	VARCHAR	-	-	Y
SQL_BIT	-	-	-	-	-	-	SMALLINT	2	-	-
SQL_CHAR	1	254	-	-	-	-	CHAR	-	-	N
SQL_CHAR	255	8000	-	-	-	-	VARCHAR	-	-	N

Table 160. Microsoft SQL Server forward default data type mappings (continued)

REMOTE_TYPENAME	REMOTE_LOWER_LEN	REMOTE_UPPER_LEN	REMOTE_LOWER_SCALE	REMOTE_UPPER_SCALE	REMOTE_BIT_DATA	REMOTE_DATA_OPERATORS	FEDERATED_TYPENAME	FEDERATED_LENGTH	FEDERATED_SCALE	FEDERATED_BIT_DATA
SQL_DATE	-	-	-	-	-	-	DATE	4	-	-
SQL_DECIMAL	1	31	0	31	-	-	DECIMAL	-	-	-
SQL_DECIMAL	32	38	0	38	-	-	DOUBLE	8	-	-
SQL_DECIMAL	32	32	0	31	-	-	DOUBLE	8	-	-
SQL_DOUBLE	-	-	-	-	-	-	DOUBLE	8	-	-
SQL_FLOAT	-	-	-	-	-	-	DOUBLE	8	-	-
SQL_GUID [2]	1	4000	-	-	Y	-	VARCHAR	16	-	Y
SQL_INTEGER	-	-	-	-	-	-	INTEGER	4	-	-
SQL_LONGVARCHAR	-	-	-	-	-	-	CLOB	2147483647	-	N
SQL_LONGVARBINARY	-	-	-	-	-	-	BLOB	-	-	Y
SQL_NUMERIC	1	31	0	31	-	-	DECIMAL	-	-	-
SQL_REAL	-	-	-	-	-	-	DOUBLE	8	-	-
SQL_SMALLINT	-	-	-	-	-	-	SMALLINT	2	-	-
SQL_TIME	-	-	-	-	-	-	TIME	3	-	-
SQL_TIMESTAMP	-	-	-	-	-	-	TIMESTAMP	10	-	-
SQL_TINYINT	-	-	-	-	-	-	SMALLINT	2	-	-
SQL_VARBINARY	1	8000	-	-	-	-	VARCHAR	-	-	Y
SQL_VARCHAR	1	8000	-	-	-	-	VARCHAR	-	-	N
text	-	-	-	-	-	-	CLOB	-	-	N
timestamp	-	-	-	-	-	-	VARCHAR	8		Y
tinyint	-	-	-	-	-	-	SMALLINT	2	-	-
uniqueidentifier [2]	1	4000	-	-	Y	-	VARCHAR	16	-	Y
varbinary	1	8000	-	-	-	-	VARCHAR	-	-	Y
varchar	1	8000	-	-	-	-	VARCHAR	-	-	N

Notes:

1. This type mapping is valid only with Microsoft SQL Server Version 6.5.
2. This type mapping is valid only with Microsoft SQL Server Version 7 and Version 2000.
3. This type mapping is valid only with Windows 2000 operating systems.
4. This type mapping is valid only with Microsoft SQL Server Version 2000.

ODBC data sources

Table 161. ODBC forward default data type mappings (Not all columns shown)

REMOTE_TYPENAME	REMOTE_LOWER_LEN	REMOTE_UPPER_LEN	REMOTE_LOWER_SCALE	REMOTE_UPPER_SCALE	REMOTE_BIT_DATA	REMOTE_DATA_OPERATORS	FEDERATED_TYPENAME	FEDERATED_LENGTH	FEDERATED_SCALE	FEDERATED_BIT_DATA
| SQL_BIGINT	-	-	-	-	-	-	BIGINT	8	-	-
SQL_BINARY	1	254	-	-	-	-	CHARACTER	-	-	Y
SQL_BINARY	255	32672	-	-	-	-	VARCHAR	-	-	Y
SQL_BIT	-	-	-	-	-	-	SMALLINT	2	-	-
SQL_CHAR	1	254	-	-	-	-	CHAR	-	-	N
SQL_CHAR	255	32672	-	-	-	-	VARCHAR	-	-	N
SQL_DECIMAL	1	31	0	31	-	-	DECIMAL	-	-	-
SQL_DECIMAL	32	38	0	38	-	-	DOUBLE	8	-	-
SQL_DOUBLE	-	-	-	-	-	-	DOUBLE	8	-	-
SQL_FLOAT	-	-	-	-	-	-	DOUBLE	8	-	-
SQL_INTEGER	-	-	-	-	-	-	INTEGER	4	-	-
SQL_LONGVARCHAR	-	-	-	-	-	-	CLOB	2147483647	-	N
SQL_LONGVARBINARY	-	-	-	-	-	-	BLOB	-	-	Y
SQL_NUMERIC	1	31	0	31	-	-	DECIMAL	-	-	-
SQL_NUMERIC	32	32	0	31	-	-	DOUBLE	8	-	-
| SQL_REAL	-	-	-	-	-	-	REAL	4	-	-
SQL_SMALLINT	-	-	-	-	-	-	SMALLINT	2	-	-
| SQL_TYPE_DATE	-	-	-	-	-	-	DATE	4	-	-
| SQL_TYPE_TIME	-	-	-	-	-	-	TIME	3	-	-
| SQL_TYPE_TIMESTAMP	-	-	-	-	-	-	TIMESTAMP	10	-	-
SQL_TINYINT	-	-	-	-	-	-	SMALLINT	2	-	-
SQL_VARBINARY	1	32672	-	-	-	-	VARCHAR	-	-	Y
SQL_VARCHAR	1	32672	-	-	-	-	VARCHAR	-	-	N
| SQL_WCHAR	1	127	-	-	-	-	CHAR	-	-	N
| SQL_WCHAR	128	16336	-	-	-	-	VARCHAR	-	-	N
| SQL_WVARCHAR	1	16336	-	-	-	-	VARCHAR	-	-	N
| SQL_WLONGVARCHAR	-	1073741823	-	-	-	-	CLOB	2147483647	-	N

Oracle NET8 data sources

Table 162. Oracle NET8 forward default data type mappings

REMOTE_TYPENAME	REMOTE_LOWER_LEN	REMOTE_UPPER_LEN	REMOTE_LOWER_SCALE	REMOTE_UPPER_SCALE	REMOTE_BIT_DATA	REMOTE_DATA_OPERATORS	FEDERATED_TYPENAME	FEDERATED_LENGTH	FEDERATED_SCALE	FEDERATED_BIT_DATA
BLOB	0	0	0	0	-	\0	BLOB	2147483647	0	Y
CHAR	1	254	0	0	-	\0	CHAR	0	0	N
CHAR	255	2000	0	0	-	\0	VARCHAR	0	0	N
CLOB	0	0	0	0	-	\0	CLOB	2147483647	0	N
DATE	0	0	0	0	-	\0	TIMESTAMP	0	0	N
FLOAT	1	126	0	0	-	\0	DOUBLE	0	0	N
LONG	0	0	0	0	-	\0	CLOB	2147483647	0	N
LONG RAW	0	0	0	0	-	\0	BLOB	2147483647	0	Y
MLSLABEL	0	0	0	0	-	\0	VARCHAR	255	0	N
NUMBER	1	38	-84	127	-	\0	DOUBLE	0	0	N
NUMBER	1	31	0	31	-	>=	DECIMAL	0	0	N
NUMBER	1	4	0	0	-	\0	SMALLINT	0	0	N
NUMBER	5	9	0	0	-	\0	INTEGER	0	0	N
NUMBER	-	10	0	0	-	\0	DECIMAL	0	0	N
RAW	1	2000	0	0	-	\0	VARCHAR	0	0	Y
ROWID	0	0	0	NULL	-	\0	CHAR	18	0	N
TIMESTAMP [1]	-	-	-	-	-	-	TIMESTAMP	10	-	-
VARCHAR2	1	4000	0	0	-	\0	VARCHAR	0	0	N

Notes:

1. This type mapping is valid only for Oracle 9i (or later) client and server configurations.

Sybase data sources

Table 163. Sybase CTLIB forward default data type mappings

REMOTE_TYPENAME	REMOTE_LOWER_LEN	REMOTE_UPPER_LEN	REMOTE_LOWER_SCALE	REMOTE_UPPER_SCALE	REMOTE_BIT_DATA	REMOTE_DATA_OPERATORS	FEDERATED_TYPENAME	FEDERATED_LENGTH	FEDERATED_SCALE	FEDERATED_BIT_DATA
binary	1	254	-	-	-	-	CHAR	-	-	Y
binary	255	16384	-	-	-	-	VARCHAR	-	-	Y
bit	-	-	-	-	-	-	SMALLINT	-	-	-
char	1	254	-	-	-	-	CHAR	-	-	N
char	255	16384	-	-	-	-	VARCHAR	-	-	N
char null (see varchar)										
datetime	-	-	-	-	-	-	TIMESTAMP	-	-	-
datetimn	-	-	-	-	-	-	TIMESTAMP	-	-	-
decimal	1	31	0	31	-	-	DECIMAL	-	-	-
decimal	32	38	0	38	-	-	DOUBLE	-	-	-
decimaln	1	31	0	31	-	-	DECIMAL	-	-	-
decimaln	32	38	0	38	-	-	DOUBLE	-	-	-
float	-	4	-	-	-	-	REAL	-	-	-
float	-	8	-	-	-	-	DOUBLE	-	-	-
floatn	-	4	-	-	-	-	REAL	-	-	-
floatn	-	8	-	-	-	-	DOUBLE	-	-	-
image	-	-	-	-	-	-	BLOB	-	-	-
int	-	-	-	-	-	-	INTEGER	-	-	-
intn	-	-	-	-	-	-	INTEGER	-	-	-
money	-	-	-	-	-	-	DECIMAL	19	4	-
moneyn	-	-	-	-	-	-	DECIMAL	19	4	-
nchar	1	254	-	-	-	-	CHAR	-	-	N
nchar	255	16384	-	-	-	-	VARCHAR	-	-	N
nchar null (see nvarchar)										
numeric	1	31	0	31	-	-	DECIMAL	-	-	-
numeric	32	38	0	38	-	-	DOUBLE	-	-	-
numericn	1	31	0	31	-	-	DECIMAL	-	-	-
numericn	32	38	0	38	-	-	DOUBLE	-	-	-
nvarchar	1	16384	-	-	-	-	VARCHAR	-	-	N
real	-	-	-	-	-	-	REAL	-	-	-

Table 163. Sybase CTLIB forward default data type mappings (continued)

REMOTE_TYPENAME	REMOTE_LOWER_LEN	REMOTE_UPPER_LEN	REMOTE_LOWER_SCALE	REMOTE_UPPER_SCALE	REMOTE_BIT_DATA	REMOTE_DATA_OPERATORS	FEDERATED_TYPENAME	FEDERATED_LENGTH	FEDERATED_SCALE	FEDERATED_BIT_DATA
smalldatetime	-	-	-	-	-	-	TIMESTAMP	-	-	-
smallint	-	-	-	-	-	-	SMALLINT	-	-	-
smallmoney	-	-	-	-	-	-	DECIMAL	10	4	-
sysname	1	254	-	-	-	-	CHAR	-	-	N
text	-	-	-	-	-	-	CLOB	-	-	-
timestamp	-	-	-	-	-	-	VARCHAR	8	-	Y
tinyint	-	-	-	-	-	-	SMALLINT	-	-	-
unichar[1]	1	254	-	-	-	-	CHAR	-	-	N
unichar[1]	255	16384	-	-	-	-	VARCHAR	-	-	N
unichar null (see univarchar)										
univarchar[1]	1	16384	-	-	-	-	VARCHAR	-	-	N
varbinary	1	16384	-	-	-	-	VARCHAR	-	-	Y
varchar	1	16384	-	-	-	-	VARCHAR	-	-	N

Notes:

1. Valid for non-Unicode federated databases.

Teradata data sources

Table 164. Teradata forward default data type mappings (Not all columns shown)

REMOTE_TYPENAME	REMOTE_LOWER_LEN	REMOTE_UPPER_LEN	REMOTE_LOWER_SCALE	REMOTE_UPPER_SCALE	REMOTE_BIT_DATA	REMOTE_DATA_OPERATORS	FEDERATED_TYPENAME	FEDERATED_LENGTH	FEDERATED_SCALE	FEDERATED_BIT_DATA
BYTE	1	254	-	-	-	-	CHAR	-	-	Y
BYTE	255	32672	-	-	-	-	VARCHAR	-	-	Y
BYTE	32673	64000	-	-	-	-	BLOB	-	-	-

Table 164. Teradata forward default data type mappings (Not all columns shown) (continued)

REMOTE_TYPENAME	REMOTE_LOWER_LEN	REMOTE_UPPER_LEN	REMOTE_LOWER_SCALE	REMOTE_UPPER_SCALE	REMOTE_BIT_DATA	REMOTE_DATA_OPERATORS	FEDERATED_TYPENAME	FEDERATED_LENGTH	FEDERATED_SCALE	FEDERATED_BIT_DATA
BYTEINT	-	-	-	-	-	-	SMALLINT	-	-	-
CHAR	1	254	-	-	-	-	CHARACTER	-	-	-
CHAR	255	32672	-	-	-	-	VARCHAR	-	-	-
CHAR	32673	64000	-	-	-	-	CLOB	-	-	-
DATE	-	-	-	-	-	-	DATE	-	-	-
DECIMAL	1	18	0	18	-	-	DECIMAL	-	-	-
DOUBLE PRECISION	-	-	-	-	-	-	DOUBLE	-	-	-
FLOAT	-	-	-	-	-	-	DOUBLE	-	-	-
GRAPHIC	1	127	-	-	-	-	GRAPHIC	-	-	-
GRAPHIC	128	16336	-	-	-	-	VARGRAPHIC	-	-	-
GRAPHIC	16337	32000	-	-	-	-	DBCLOB	-	-	-
INTEGER	-	-	-	-	-	-	INTEGER	-	-	-
INTERVAL	-	-	-	-	-	-	CHAR	-	-	-
NUMERIC	1	18	0	18	-	-	DECIMAL	-	-	-
REAL	-	-	-	-	-	-	DOUBLE	-	-	-
SMALLINT	-	-	-	-	-	-	SMALLINT	-	-	-
TIMESTAMP	-	-	-	-	-	-	TIMESTAMP	-	-	-
VARBYTE	1	32762	-	-	-	-	VARCHAR	-	-	Y
VARBYTE	32763	64000	-	-	-	-	BLOB	-	-	-
VARCHAR	1	32672	-	-	-	-	VARCHAR	-	-	-
VARCHAR	32673	64000	-	-	-	-	CLOB	-	-	-
VARGRAPHIC	1	16336	-	-	-	-	VARGRAPHIC	-	-	-
VARGRAPHIC	16337	32000	-	-	-	-	DBCLOB	-	-	-

Related concepts:

- "Forward and reverse data type mappings" in the *Federated Systems Guide*

Related reference:

- "Altering long data types to varchar data types" in the *Federated Systems Guide*
- "Unicode default forward data type mappings - NET8 wrapper" in the *Federated Systems Guide*
- "Unicode default forward data type mappings - Sybase wrapper" in the *Federated Systems Guide*

- “Unicode default forward data type mappings - ODBC wrapper” in the *Federated Systems Guide*
- “Unicode default forward data type mappings - Microsoft SQL Server wrapper” in the *Federated Systems Guide*

Appendix I. Default reverse data type mappings

The two kinds of mappings between data source data types and federated database data types are forward type mappings and reverse type mappings. In a *forward type mapping*, the mapping is from a remote type to a comparable local type. The other type of mapping is a *reverse type mapping*, which is used with transparent DDL to create or modify remote tables.

For most data sources, the default type mappings are in the wrappers. The default type mappings for DB2 family data sources are in the DRDA wrapper. The default type mappings for Informix are in the INFORMIX wrapper, and so forth.

When you define a remote table or view to the DB2 federated database, the definition includes a reverse type mapping. The mapping is from a *local* DB2 for Linux, UNIX, and Windows data type for each column, and the corresponding *remote* data type. For example, there is a default reverse type mapping in which the local type REAL points to the Informix type SMALLFLOAT.

DB2 for Linux, UNIX, and Windows federated servers do not support mappings for LONG VARCHAR, LONG VARGRAPHIC, DATALINK, and user-defined types.

When you use the CREATE TABLE statement to create a remote table, you specify the local data types you want to include in the remote table. These default reverse type mappings will assign corresponding remote types to these columns. For example, suppose that you use the CREATE TABLE statement to define an Informix table with a column C2. You specify BIGINT as the data type for C2 in the statement. The default reverse type mapping of BIGINT depends on which version of Informix you are creating the table on. The mapping for C2 in the Informix table will be to DECIMAL in Informix Version 8 and to INT8 in Informix Version 9.

You can override a default reverse type mapping, or create a new reverse type mapping with the CREATE TYPE MAPPING statement.

The following tables show the default reverse mappings between DB2 for Linux, UNIX, and Windows local data types and remote data source data types.

These mappings are valid with all the supported versions, unless otherwise noted.

DB2 for z/OS and OS/390 data sources

Table 165. DB2 for z/OS and OS/390 reverse default data type mappings (Not all columns shown)

FEDERATED_TYPENAME	FEDERATED_LOWER_LEN	FEDERATED_UPPER_LEN	FEDERATED_LOWER_SCALE	FEDERATED_UPPER_SCALE	FEDERATED_BIT_DATA	FEDERATED_DATA_OPERATORS	REMOTE_TYPENAME	REMOTE_LENGTH	REMOTE_SCALE	REMOTE_BIT_DATA
BLOB	-	-	-	-	-	-	BLOB	-	-	-
CHARACTER	-	-	-	-	-	-	CHAR	-	-	N
CHARACTER	-	-	-	-	Y	-	CHAR	-	-	Y
CLOB	-	-	-	-	-	-	CLOB	-	-	-
DATE	-	4	-	-	-	-	DATE	-	-	-
DBCLOB	-	-	-	-	-	-	DBCLOB	-	-	-
DECIMAL	-	-	-	-	-	-	DECIMAL	-	-	-
DOUBLE	-	8	-	-	-	-	DOUBLE	-	-	–
FLOAT	-	8	-	-	-	-	DOUBLE	-	-	-
GRAPHIC	-	-	-	-	-	-	GRAPHIC	-	-	N
INTEGER	-	4	-	-	-	-	INTEGER	-	-	-
REAL	-	4	-	-	-	-	REAL	-	-	-
SMALLINT	-	2	-	-	-	-	SMALLINT	-	-	-
TIME	-	3	-	-	-	-	TIME	-	-	-
TIMESTAMP	-	10	-	-	-	-	TIMESTAMP	-	-	-
VARCHAR	-	-	-	-	-	-	VARCHAR	-	-	N
VARCHAR	-	-	-	-	Y	-	VARCHAR	-	-	Y
VARGRAPHIC	-	-	-	-	-	-	VARGRAPHIC	-	-	N

DB2 for iSeries data sources

Table 166. DB2 for iSeries reverse default data type mappings (Not all columns shown)

FEDERATED_TYPENAME	FEDERATED_LOWER_LEN	FEDERATED_UPPER_LEN	FEDERATED_LOWER_SCALE	FEDERATED_UPPER_SCALE	FEDERATED_BIT_DATA	FEDERATED_DATA_OPERATORS	REMOTE_TYPENAME	REMOTE_LENGTH	REMOTE_SCALE	REMOTE_BIT_DATA
BLOB	-	-	-	-	-	-	BLOB	-	-	-
CHARACTER	-	-	-	-	-	-	CHARACTER	-	-	N
CHARACTER	-	-	-	-	Y	-	CHARACTER	-	-	Y
CLOB	-	-	-	-	-	-	CLOB	-	-	-
DATE	-	4	-	-	-	-	DATE	-	-	-
DBCLOB	-	-	-	-	-	-	DBCLOB	-	-	-
DECIMAL	-	-	-	-	-	-	NUMERIC	-	-	-
DECIMAL	-	-	-	-	-	-	DECIMAL	-	-	-
DOUBLE	-	8	-	-	-	-	FLOAT	-	-	-
GRAPHIC	-	-	-	-	-	-	GRAPHIC	-	-	N
INTEGER	-	4	-	-	-	-	INTEGER	-	-	-
REAL	-	4	-	-	-	-	FLOAT	-	-	-
SMALLINT	-	2	-	-	-	-	SMALLINT	-	-	-
TIME	-	3	-	-	-	-	TIME	-	-	-
TIMESTAMP	-	10	-	-	-	-	TIMESTAMP	-	-	-
VARCHAR	-	-	-	-	-	-	VARCHAR	-	-	N
VARCHAR	-	-	-	-	Y	-	VARCHAR	-	-	Y
VARGRAPHIC	-	-	-	-	-	-	VARG	-	-	N

DB2 for VM and VSE data sources

Table 167. DB2 for VM and VSE reverse default data type mappings (Not all columns shown)

FEDERATED_TYPENAME	FEDERATED_LOWER_LEN	FEDERATED_UPPER_LEN	FEDERATED_LOWER_SCALE	FEDERATED_UPPER_SCALE	FEDERATED_BIT_DATA	FEDERATED_DATA_OPERATORS	REMOTE_TYPENAME	REMOTE_LENGTH	REMOTE_SCALE	REMOTE_BIT_DATA
BLOB	-	-	-	-	-	-	BLOB	-	-	-
CHARACTER	-	-	-	-	-	-	CHAR	-	-	-
CHARACTER	-	-	-	-	Y	-	CHAR	-	-	Y
CLOB	-	-	-	-	-	-	CLOB	-	-	-
DATE	-	4	-	-	-	-	DATE	-	-	-
DBCLOB	-	-	-	-	-	-	DBCLOB	-	-	-
DECIMAL	-	-	-	-	-	-	DECIMAL	-	-	-
DOUBLE	-	8	-	-	-	-	FLOAT	-	-	-
GRAPHIC	-	-	-	-	-	-	GRAPHIC	-	-	N
INTEGER	-	4	-	-	-	-	INTEGER	-	-	-
REAL	-	4	-	-	-	-	REAL	-	-	-
SMALLINT	-	2	-	-	-	-	SMALLINT	-	-	-
TIME	-	3	-	-	-	-	TIME	-	-	-
TIMESTAMP	-	10	-	-	-	-	TIMESTAMP	-	-	-
VARCHAR	-	-	-	-	-	-	VARCHAR	-	-	-
VARCHAR	-	-	-	-	Y	-	VARCHAR	-	-	Y
VARGRAPH	-	-	-	-	-	-	VARGRAPH	-	-	N

DB2 for Linux, UNIX, and Windows data sources

Table 168. DB2 for Linux, UNIX, and Windows reverse default data type mappings (Not all columns shown)

FEDERATED_TYPENAME	FEDERATED_LOWER_LEN	FEDERATED_UPPER_LEN	FEDERATED_LOWER_SCALE	FEDERATED_UPPER_SCALE	FEDERATED_BIT_DATA	FEDERATED_DATA_OPERATORS	REMOTE_TYPENAME	REMOTE_LENGTH	REMOTE_SCALE	FEDERATED_BIT_DATA
BIGINT	-	8	-	-	-	-	BIGINT	-	-	-
BLOB	-	-	-	-	-	-	BLOB	-	-	-
CHARACTER	-	-	-	-	-	-	CHAR	-	-	N
CHARACTER	-	-	-	-	Y	-	CHAR	-	-	Y
CLOB	-	-	-	-	-	-	CLOB	-	-	-
DATE	-	4	-	-	-	-	DATE	-	-	-
DBCLOB	-	-	-	-	-	-	DBCLOB	-	-	-
DECIMAL	-	-	-	-	-	-	DECIMAL	-	-	-
DOUBLE	-	8	-	-	-	-	DOUBLE	-	-	-
FLOAT	-	8	-	-	-	-	DOUBLE	-	-	-
GRAPHIC	-	-	-	-	-	-	GRAPHIC	-	-	N
INTEGER	-	4	-	-	-	-	INTEGER	-	-	-
REAL	-	-	-	-	-	-	REAL	-	-	-
SMALLINT	-	2	-	-	-	-	SMALLINT	-	-	-
TIME	-	3	-	-	-	-	TIME	-	-	-
TIMESTAMP	-	10	-	-	-	-	TIMESTAMP	-	-	-
VARCHAR	-	-	-	-	-	-	VARCHAR	-	-	N
VARCHAR	-	-	-	-	Y	-	VARCHAR	-	-	Y
VARGRAPH	-	-	-	-	-	-	VARGRAPHIC	-	-	N
VARGRAPHIC	-	-	-	-	-	-	VARGRAPHIC	-	-	-

Informix data sources

Table 169. Informix reverse default data type mappings

FEDERATED_TYPENAME	FEDERATED_LOWER_LEN	FEDERATED_UPPER_LEN	FEDERATED_LOWER_SCALE	FEDERATED_UPPER_SCALE	FEDERATED_BIT_DATA	FEDERATED_DATA_OPERATORS	REMOTE_TYPENAME	REMOTE_LENGTH	REMOTE_SCALE	REMOTE_BIT_DATA
BIGINT [1]	-	-	-	-	-	-	DECIMAL	19	-	-
BIGINT [2]	-	-	-	-	-	-	INT8	-	-	-
BLOB	1	2147483647	-	-	-	-	BYTE	-	-	-
CHARACTER	-	-	-	-	N	-	CHAR	-	-	-
CHARACTER	-	-	-	-	Y	-	BYTE	-	-	-
CLOB	1	2147483647	-	-	-	-	TEXT	-	-	-
DATE	-	4	-	-	-	-	DATE	-	-	-
DECIMAL	-	-	-	-	-	-	DECIMAL	-	-	-
DOUBLE	-	8	-	-	-	-	FLOAT	-	-	-
INTEGER	-	4	-	-	-	-	INTEGER	-	-	-
REAL	-	4	-	-	-	-	SMALLFLOAT	-	-	-
SMALLINT	-	2	-	-	-	-	SMALLINT	-	-	-
TIME	-	3	-	-	-	-	DATETIME	6	10	-
TIMESTAMP	-	10	-	-	-	-	DATETIME	0	15	-
VARCHAR	1	254	-	-	N	-	VARCHAR	-	-	-
VARCHAR	255	32672	-	-	N	-	TEXT	-	-	-
VARCHAR	-	-	-	-	Y	-	BYTE	-	-	-
VARCHAR [2]	255	2048	-	-	N	-	LVARCHAR	-	-	-
VARCHAR [2]	2049	32672	-	-	N	-	TEXT	-	-	-

Notes:

1. This type mapping is valid only with Informix server Version 8 (or lower).
2. This type mapping is valid only with Informix server Version 9.

For the Informix DATETIME data type, the DB2 UNIX and Windows federated server uses the Informix high-level qualifer as the REMOTE_LENGTH and the Informix low-level qualifier as the REMOTE_SCALE.

The Informix qualifiers are the "TU_" constants defined in the Informix Client SDK `datatime.h` file. The contstants are:

0 = YEAR	8 = MINUTE	13 = FRACTION(3)
2 = MONTH	10 = SECOND	14 = FRACTION(4)
4 = DAY	11 = FRACTION(1)	15 = FRACTION(5)
6 = HOUR	12 = FRACTION(2)	

Microsoft SQL Server data sources

Table 170. Microsoft SQL Server reverse default data type mappings (Not all columns shown)

FEDERATED_TYPENAME	FEDERATED_LOWER_LEN	FEDERATED_UPPER_LEN	FEDERATED_LOWER_SCALE	FEDERATED_UPPER_SCALE	FEDERATED_BIT_DATA	FEDERATED_DATA_OPERATORS	REMOTE_TYPENAME	REMOTE_LENGTH	REMOTE_SCALE	REMOTE_BIT_DATA
BIGINT [1]	-	-	-	-	-	-	bigint	-	-	-
BLOB	-	-	-	-	-	-	image	-	-	-
CHARACTER	-	-	-	-	Y	-	binary	-	-	-
CHARACTER	-	-	-	-	N	-	char	-	-	-
CLOB	-	-	-	-	-	-	text	-	-	-
DATE	-	4	-	-	-	-	datetime	-	-	-
DECIMAL	-	-	-	-	-	-	decimal	-	-	-
DOUBLE	-	8	-	-	-	-	float	-	-	-
INTEGER	-	-	-	-	-	-	int	-	-	-
SMALLINT	-	-	-	-	-	-	smallint	-	-	-
REAL	-	4	-	-	-	-	real	-	-	-
TIME	-	3	-	-	-	-	datetime	-	-	-
TIMESTAMP	-	10	-	-	-	-	datetime	-	-	-
VARCHAR	1	8000	-	-	N	-	varchar	-	-	-
VARCHAR	8001	32672	-	-	N	-	text	-	-	-
VARCHAR	1	8000	-	-	Y	-	varbinary	-	-	-
VARCHAR	8001	32672	-	-	Y	-	image	-	-	-

Notes:

1. This type mapping is valid only with Microsoft SQL Server Version 2000.

Oracle NET8 data sources

Table 171. Oracle NET8 reverse default data type mappings

FEDERATED_TYPENAME	FEDERATED_LOWER_LEN	FEDERATED_UPPER_LEN	FEDERATED_LOWER_SCALE	FEDERATED_UPPER_SCALE	FEDERATED_BIT_DATA	FEDERATED_DATA_OPERATORS	REMOTE_TYPENAME	REMOTE_LENGTH	REMOTE_SCALE	REMOTE_BIT_DATA
BLOB	0	2147483647	0	0	Y	\0	BLOB	0	0	Y
CHARACTER	1	254	0	0	N	\0	CHAR	0	0	N
CHARACTER	1	254	0	0	Y	\0	RAW	0	0	Y
CLOB	0	2147483647	0	0	N	\0	CLOB	0	0	N
DATE	0	4	0	0	N	\0	DATE	0	0	N
DECIMAL	0	0	0	0	N	\0	NUMBER	0	0	N
DOUBLE	0	8	0	0	N	\0	FLOAT	126	0	N
FLOAT	0	8	0	0	N	\0	FLOAT	126	0	N
INTEGER	0	4	0	0	N	\0	NUMBER	9	0	N
REAL	0	4	0	0	N	\0	FLOAT	63	0	N
SMALLINT	0	2	0	0	N	\0	NUMBER	4	0	N
TIME	0	3	0	0	N	\0	DATE	0	0	N
TIMESTAMP	0	10	0	0	N	\0	DATE	0	0	N
VARCHAR	1	4000	0	0	N	\0	VARCHAR2	0	0	N
VARCHAR	1	2000	0	0	Y	\0	RAW	0	0	Y

Note: The DB2 Universal Database for Linux, UNIX, and Windows BIGINT data type is not available for transparent DDL. You cannot specify the BIGINT data type in a CREATE TABLE statement when you create a remote Oracle table.

Sybase data sources

Table 172. Sybase CTLIB default reverse data type mappings

FEDERATED_TYPENAME	FEDERATED_LOWER_LEN	FEDERATED_UPPER_LEN	FEDERATED_LOWER_SCALE	FEDERATED_UPPER_SCALE	FEDERATED_BIT_DATA	FEDERATED_DATA_OPERATORS	REMOTE_TYPENAME	REMOTE_LENGTH	REMOTE_SCALE	REMOTE_BIT_DATA
BIGINT	-	-	-	-	-	-	decimal	19	0	-
BLOB	-	-	-	-	-	-	image	-	-	-
CHARACTER	-	-	-	-	N	-	char	-	-	-
CHARACTER	-	-	-	-	Y	-	binary	-	-	-
CLOB	-	-	-	-	-	-	text	-	-	-
DATE	-	-	-	-	-	-	datetime	-	-	-
DECIMAL	-	-	-	-	-	-	decimal	-	-	-
DOUBLE	-	-	-	-	-	-	float	-	-	-
GRAPHIC	-	-	-	-	-	-	unichar	-	-	-
VARGRAPHIC	-	-	-	-	-	-	univarchar	-	-	-
INTEGER	-	-	-	-	-	-	integer	-	-	-
REAL	-	-	-	-	-	-	real	-	-	-
SMALLINT	-	-	-	-	-	-	smallint	-	-	-
TIME	-	-	-	-	-	-	datetime	-	-	-
TIMESTAMP	-	-	-	-	-	-	datetime	-	-	-
VARCHAR[1]	1	255	-	-	N	-	varchar	-	-	-
VARCHAR[1]	256	32672	-	-	N	-	text	-	-	-
VARCHAR [2]	1	16384	-	-	N	-	varchar	-	-	-
VARCHAR [2]	16385	32672	-	-	N	-	text	-	-	-
VARCHAR[1]	1	255	-	-	Y	-	varbinary	-	-	-
VARCHAR[1]	256	32672	-	-	Y	-	image	-	-	-
VARCHAR [2]	1	16384	-	-	Y	-	varbinary	-	-	-
VARCHAR [2]	16385	32672	-	-	Y	-	image	-	-	-

Notes:
1. This type mapping is valid only for CTLIB with Sybase server version 12.0 (or earlier).
2. This type mapping is valid only for CTLIB with Sybase server version 12.5 (or later).

Teradata data sources

Table 173. Teradata reverse default data type mappings (Not all columns shown)

FEDERATED_TYPENAME	FEDERATED_LOWER_LEN	FEDERATED_UPPER_LEN	FEDERATED_LOWER_SCALE	FEDERATED_UPPER_SCALE	FEDERATED_BIT_DATA	FEDERATED_DATA_OPERATORS	REMOTE_TYPENAME	REMOTE_LENGTH	REMOTE_SCALE	FEDERATED_BIT_DATA
BLOB [1]	1	64000	-	-	-	-	VARBYTE	-	-	-
CHARACTER	-	-	-	-	-	-	CHARACTER	-	-	-
CHARACTER	-	-	-	-	Y	-	BYTE	-	-	-
CLOB [2]	1	64000	-	-		-	VARCHAR	-	-	-
DATE	-	-	-	-	-	-	DATE	-	-	-
DBCLOB [3]	1	32000	-	-	-	-	VARGRAPHIC	-	-	-
DECIMAL	1	18	0	18	-	-	DECIMAL	-	-	-
DECIMAL	19	31	0	31	-	-	FLOAT	-	-	-
DOUBLE	-	-	-	-	-	-	FLOAT	-	-	-
GRAPHIC	-	-	-	-	-	-	GRAPHIC	-	-	-
INTEGER	-	-	-	-	-	-	INTEGER	-	-	-
REAL	-	-	-	-	-	-	FLOAT	-	-	-
SMALLINT	-	-	-	-	-	-	SMALLINT	-	-	-
TIME	-	-	-	-	-	-	TIME	-	-	-
TIMESTAMP	-	-	-	-	-	-	TIMESTAMP	-	-	-
VARCHAR	-	-	-	-	-	-	VARCHAR	-	-	-
VARCHAR	-	-	-	-	Y	-	VARBYTE	-	-	-
VARGRAPHIC	-	-	-	-	-	-	VARGRAPHIC	-	-	-

Notes:

1. The Teradata VARBYTE data type can contain only the specified length (1 to 64000) of a DB2 BLOB data type.
2. The Teradata VARCHAR data type can contain only the specified length (1 to 64000) of a DB2 CLOB data type.
3. The Teradata VARGRAPHIC data type can contain only the specified length (1 to 32000) of a DB2 DBCLOB data type.

Related concepts:

- "Forward and reverse data type mappings" in the *Federated Systems Guide*

Appendix J. Function mapping options for federated systems

DB2 Information Integrator supplies default mappings between existing built-in data source functions and built-in DB2 functions. For most data sources, the default function mappings are in the wrappers. To use a data source function that the federated server does not recognize, you must create a function mapping between a data source function and a counterpart function at the federated database.

The primary purpose of function mapping options, is to provide information about the potential cost of executing a data source function at the data source. Pushdown analysis determines if a function at the data source is able to execute a function in a query. The query optimizer decides if pushing down the function processing to the data source is the least cost alternative.

The statistical information provided in the function mapping definition helps the query optimizer compare the estimated cost of executing the data source function with the estimated cost of executing the DB2 function.

Table 174. Function mapping options and their settings

Option	Valid settings	Default setting
DISABLE	Disable a default function mapping. Valid values are 'Y' and 'N'.	'N'
INITIAL_INSTS	Estimated number of instructions processed the first and last time that the data source function is invoked.	'0'
INITIAL_IOS	Estimated number of I/Os performed the first and last time that the data source function is invoked.	'0'
IOS_PER_ARGBYTE	Estimated number of I/Os expended for each byte of the argument set that's passed to the data source function.	'0'
IOS_PER_INVOC	Estimated number of I/Os per invocation of a data source function.	'0'
INSTS_PER_ARGBYTE	Estimated number of instructions processed for each byte of the argument set that's passed to the data source function.	'0'
INSTS_PER_INVOC	Estimated number of instructions processed per invocation of the data source function.	'450'
PERCENT_ARGBYTES	Estimated average percent of input argument bytes that the data source function will actually read.	'100'
REMOTE_NAME	Name of the data source function.	local name

DB2 Information Integrator documentation

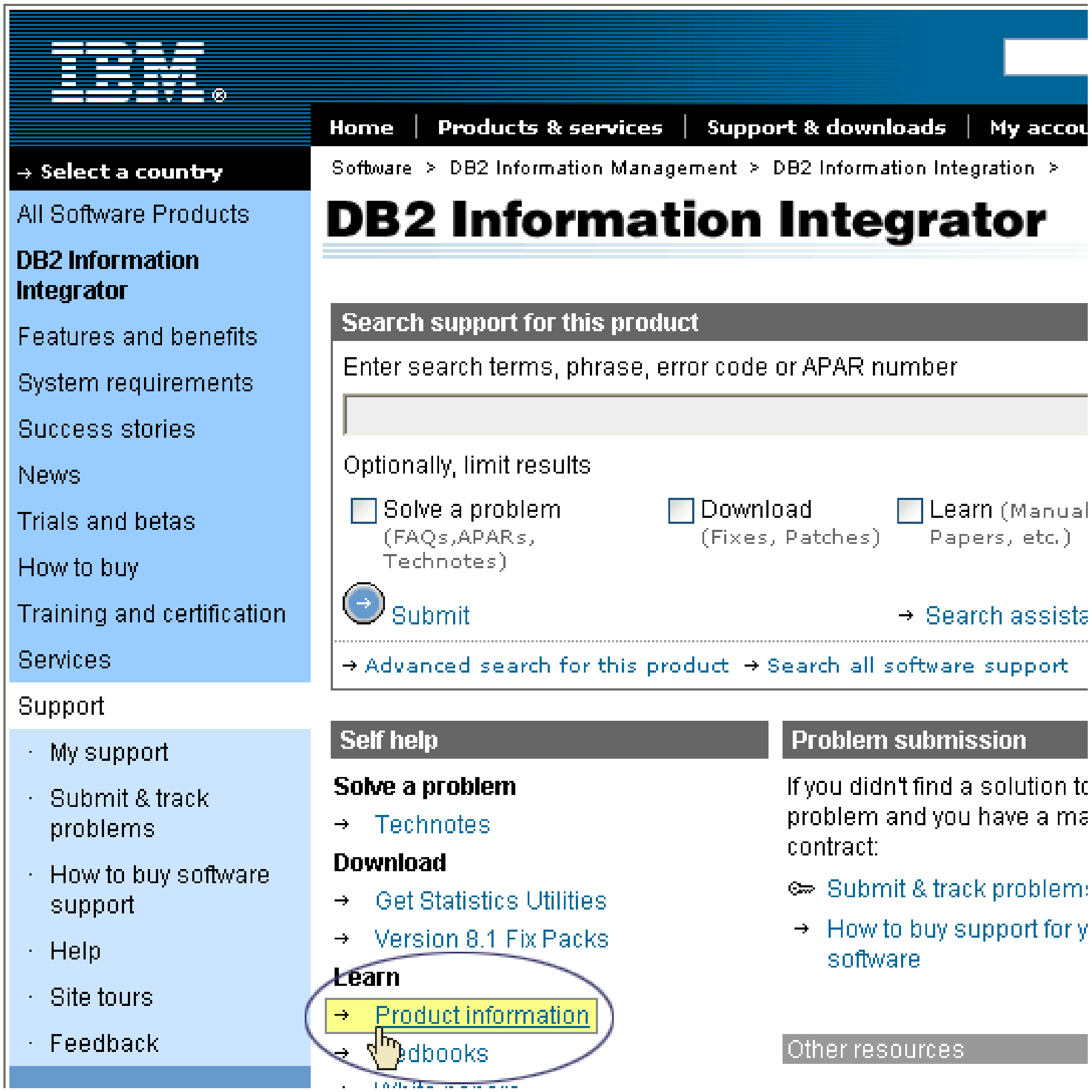
This topic provides information about the documentation that is available for DB2 Information Integrator. The tables in this topic provide the official document title, form number, and location of each PDF book. To order a printed book, you must know either the official book title or the document form number. Titles, file names, and the locations of the DB2 Information Integrator release notes and installation requirements are also provided in this topic.

This topic contains the following sections:
- Accessing DB2 Information Integrator documentation
- Documentation for replication function on z/OS
- Documentation for event publishing function for DB2 Universal Database on z/OS
- Documentation for event publishing function for IMS and VSAM on z/OS
- Documentation for event publishing and replication function on Linux, UNIX, and Windows
- Documentation for federated function on z/OS
- Documentation for federated function on Linux, UNIX, and Windows
- Documentation for enterprise search on Linux, UNIX, and Windows
- Release notes and installation requirements

Accessing DB2 Information Integrator documentation

All DB2 Information Integrator books and release notes are available in PDF files from the DB2 Information Integrator Support Web site at www.ibm.com/software/data/integration/db2ii/support.html.

To access the latest DB2 Information Integrator product documentation, from the DB2 Information Integrator Support Web site, click on the Product Information link, as shown in Figure 48 on page 640.


Home
Products & services
Support & downloads
My accou
→ Select a country
Software > DB2 Information Management > DB2 Information Integration >
DB2 Information Integrator
All Software Products
DB2 Information Integrator
Features and benefits
System requirements
Success stories
News
Trials and betas
How to buy
Training and certification
Services
Support
· My support
· Submit & track problems
· How to buy software support
· Help
· Site tours
· Feedback
Search support for this product
Optionally, limit results
Solve a problem (FAQs,APARs, Technotes)
Download (Fixes, Patches)
Learn (Manual Papers, etc.)
Submit
→ Search assista
→ Advanced search for this product
→ Search all software support
Self help
Solve a problem
→ Technotes
Download
→ Get Statistics Utilities
→ Version 8.1 Fix Packs
Learn
→ Product information
→ Redbooks
Problem submission
Submit & track problem
→ How to buy support for y software
Other resources

Enter search terms, phrase, error code or APAR number

If you didn't find a solution to problem and you have a ma contract:

Figure 48. Accessing the Product Information link from DB2 Information Integrator Support Web site

You can access the latest DB2 Information Integrator documentation, in all supported languages, from the Product Information link:

- DB2 Information Integrator product documentation in PDF files
- Fix pack product documentation, including release notes
- Instructions for downloading and installing the DB2 Information Center for Linux, UNIX, and Windows
- Links to the DB2 Information Center online

Scroll though the list to find the product documentation for the version of DB2 Information Integrator that you are using.

The DB2 Information Integrator Support Web site also provides support documentation, IBM Redbooks, white papers, product downloads, links to user groups, and news about DB2 Information Integrator.

You can also view and print the DB2 Information Integrator PDF books from the *DB2 PDF Documentation* CD.

To view or print the PDF documentation:

1. From the root directory of the *DB2 PDF Documentation* CD, open the `index.htm` file.
2. Click the language that you want to use.
3. Click the link for the document that you want to view.

Documentation about replication function on z/OS

Table 175. DB2 Information Integrator documentation about replication function on z/OS

Name	Form number	Location
ASNCLP Program Reference for Replication and Event Publishing	N/A	DB2 Information Integrator Support Web site
Introduction to Replication and Event Publishing	GC18-7567	DB2 Information Integrator Support Web site
Migrating to SQL Replication	N/A	DB2 Information Integrator Support Web site
Replication and Event Publishing Guide and Reference	SC18-7568	• *DB2 PDF Documentation* CD • DB2 Information Integrator Support Web site
Replication Installation and Customization Guide for z/OS	SC18-9127	DB2 Information Integrator Support Web site
SQL Replication Guide and Reference	SC27-1121	• *DB2 PDF Documentation* CD • DB2 Information Integrator Support Web site
Tuning for Replication and Event Publishing Performance	N/A	DB2 Information Integrator Support Web site
Tuning for SQL Replication Performance	N/A	DB2 Information Integrator Support Web site
Release Notes for IBM DB2 Information Integrator Standard Edition, Advanced Edition, and Replication for z/OS	N/A	• In the DB2 Information Center, **Product Overviews** > **Information Integration** > **DB2 Information Integrator overview** > **Problems, workarounds, and documentation updates** • DB2 Information Integrator Installation launchpad • DB2 Information Integrator Support Web site • The *DB2 Information Integrator* product CD

Documentation about event publishing function for DB2 Universal Database on z/OS

Table 176. DB2 Information Integrator documentation about event publishing function for DB2 Universal Database on z/OS

Name	Form number	Location
ASNCLP Program Reference for Replication and Event Publishing	N/A	DB2 Information Integrator Support Web site
Introduction to Replication and Event Publishing	GC18-7567	• *DB2 PDF Documentation* CD • DB2 Information Integrator Support Web site
Replication and Event Publishing Guide and Reference	SC18-7568	• *DB2 PDF Documentation* CD • DB2 Information Integrator Support Web site
Tuning for Replication and Event Publishing Performance	N/A	DB2 Information Integrator Support Web site
Release Notes for IBM DB2 Information Integrator Standard Edition, Advanced Edition, and Replication for z/OS	N/A	• In the DB2 Information Center, **Product Overviews** > **Information Integration** > **DB2 Information Integrator overview** > **Problems, workarounds, and documentation updates** • DB2 Information Integrator Installation launchpad • DB2 Information Integrator Support Web site • The *DB2 Information Integrator* product CD

Documentation about event publishing function for IMS and VSAM on z/OS

Table 177. DB2 Information Integrator documentation about event publishing function for IMS and VSAM on z/OS

Name	Form number	Location
Client Guide for Classic Federation and Event Publisher for z/OS	SC18-9160	DB2 Information Integrator Support Web site
Data Mapper Guide for Classic Federation and Event Publisher for z/OS	SC18-9163	DB2 Information Integrator Support Web site
Getting Started with Event Publisher for z/OS	GC18-9186	DB2 Information Integrator Support Web site
Installation Guide for Classic Federation and Event Publisher for z/OS	GC18-9301	DB2 Information Integrator Support Web site
Operations Guide for Event Publisher for z/OS	SC18-9157	DB2 Information Integrator Support Web site

Table 177. DB2 Information Integrator documentation about event publishing function for IMS and VSAM on z/OS (continued)

Name	Form number	Location
Planning Guide for Event Publisher for z/OS	SC18-9158	DB2 Information Integrator Support Web site
Reference for Classic Federation and Event Publisher for z/OS	SC18-9156	DB2 Information Integrator Support Web site
System Messages for Classic Federation and Event Publisher for z/OS	SC18-9162	DB2 Information Integrator Support Web site
Release Notes for IBM DB2 Information Integrator Event Publisher for IMS for z/OS	N/A	DB2 Information Integrator Support Web site
Release Notes for IBM DB2 Information Integrator Event Publisher for VSAM for z/OS	N/A	DB2 Information Integrator Support Web site

Documentation about event publishing and replication function on Linux, UNIX, and Windows

Table 178. DB2 Information Integrator documentation about event publishing and replication function on Linux, UNIX, and Windows

Name	Form number	Location
ASNCLP Program Reference for Replication and Event Publishing	N/A	DB2 Information Integrator Support Web site
Installation Guide for Linux, UNIX, and Windows	GC18-7036	• *DB2 PDF Documentation* CD • DB2 Information Integrator Support Web site
Introduction to Replication and Event Publishing	GC18-7567	• *DB2 PDF Documentation* CD • DB2 Information Integrator Support Web site
Migrating to SQL Replication	N/A	DB2 Information Integrator Support Web site
Replication and Event Publishing Guide and Reference	SC18-7568	• *DB2 PDF Documentation* CD • DB2 Information Integrator Support Web site
SQL Replication Guide and Reference	SC27-1121	DB2 Information Integrator Support Web site
Tuning for Replication and Event Publishing Performance	N/A	DB2 Information Integrator Support Web site
Tuning for SQL Replication Performance	N/A	DB2 Information Integrator Support Web site

Table 178. DB2 Information Integrator documentation about event publishing and replication function on Linux, UNIX, and Windows (continued)

Name	Form number	Location
Release Notes for IBM DB2 Information Integrator Standard Edition, Advanced Edition, and Replication for z/OS	N/A	• In the DB2 Information Center, **Product Overviews** > **Information Integration** > **DB2 Information Integrator overview** > **Problems, workarounds, and documentation updates** • DB2 Information Integrator Installation launchpad • DB2 Information Integrator Support Web site • The *DB2 Information Integrator* product CD

Documentation about federated function on z/OS

Table 179. DB2 Information Integrator documentation about federated function on z/OS

Name	Form number	Location
Client Guide for Classic Federation and Event Publisher for z/OS	SC18-9160	DB2 Information Integrator Support Web site
Data Mapper Guide for Classic Federation and Event Publisher for z/OS	SC18-9163	DB2 Information Integrator Support Web site
Getting Started with Classic Federation for z/OS	GC18-9155	DB2 Information Integrator Support Web site
Installation Guide for Classic Federation and Event Publisher for z/OS	GC18-9301	DB2 Information Integrator Support Web site
Reference for Classic Federation and Event Publisher for z/OS	SC18-9156	DB2 Information Integrator Support Web site
System Messages for Classic Federation and Event Publisher for z/OS	SC18-9162	DB2 Information Integrator Support Web site
Transaction Services Guide for Classic Federation for z/OS	SC18-9161	DB2 Information Integrator Support Web site
Release Notes for IBM DB2 Information Integrator Classic Federation for z/OS	N/A	DB2 Information Integrator Support Web site

Documentation about federated function on Linux, UNIX, and Windows

Table 180. DB2 Information Integrator documentation about federated function on Linux, UNIX, and Windows

Name	Form number	Location
Application Developer's Guide	SC18-7359	• *DB2 PDF Documentation* CD • DB2 Information Integrator Support Web site

Table 180. DB2 Information Integrator documentation about federated function on Linux, UNIX, and Windows (continued)

Name	Form number	Location
C++ API Reference for Developing Wrappers	SC18-9172	• *DB2 PDF Documentation* CD • DB2 Information Integrator Support Web site
Data Source Configuration Guide	N/A	• *DB2 PDF Documentation* CD • DB2 Information Integrator Support Web site
Federated Systems Guide	SC18-7364	• *DB2 PDF Documentation* CD • DB2 Information Integrator Support Web site
Guide to Configuring the Content Connector for VeniceBridge	N/A	DB2 Information Integrator Support Web site
Installation Guide for Linux, UNIX, and Windows	GC18-7036	• *DB2 PDF Documentation* CD • DB2 Information Integrator Support Web site
Java API Reference for Developing Wrappers	SC18-9173	• *DB2 PDF Documentation* CD • DB2 Information Integrator Support Web site
Migration Guide	SC18-7360	• *DB2 PDF Documentation* CD • DB2 Information Integrator Support Web site
Wrapper Developer's Guide	SC18-9174	• *DB2 PDF Documentation* CD • DB2 Information Integrator Support Web site
Release Notes for IBM DB2 Information Integrator Standard Edition, Advanced Edition, and Replication for z/OS	N/A	• In the DB2 Information Center, **Product Overviews** > **Information Integration** > **DB2 Information Integrator overview** > **Problems, workarounds, and documentation updates** • DB2 Information Integrator Installation launchpad • DB2 Information Integrator Support Web site • The *DB2 Information Integrator* product CD

Documentation about enterprise search function on Linux, UNIX, and Windows

Table 181. DB2 Information Integrator documentation about enterprise search function on Linux, UNIX, and Windows

Name	Form number	Location
Administering Enterprise Search	SC18-9283	DB2 Information Integrator Support Web site
Installation Guide for Enterprise Search	GC18-9282	DB2 Information Integrator Support Web site
Programming Guide and API Reference for Enterprise Search	SC18-9284	DB2 Information Integrator Support Web site
Release Notes for Enterprise Search	N/A	DB2 Information Integrator Support Web site

Release notes and installation requirements

Release notes provide information that is specific to the release and fix pack level for your product and include the latest corrections to the documentation for each release.

Installation requirements provide information that is specific to the release of your product.

Table 182. DB2 Information Integrator Release Notes and Installation Requirements

Name	File name	Location
Installation Requirements for IBM DB2 Information Integrator Event Publishing Edition, Replication Edition, Standard Edition, Advanced Edition, Advanced Edition Unlimited, Developer Edition, and Replication for z/OS	Prereqs	• The *DB2 Information Integrator* product CD • DB2 Information Integrator Installation Launchpad
Release Notes for IBM DB2 Information Integrator Standard Edition, Advanced Edition, and Replication for z/OS	ReleaseNotes	• In the DB2 Information Center, **Product Overviews** > **Information Integration** > **DB2 Information Integrator overview** > **Problems, workarounds, and documentation updates** • DB2 Information Integrator Installation launchpad • DB2 Information Integrator Support Web site • The *DB2 Information Integrator* product CD
Release Notes for IBM DB2 Information Integrator Event Publisher for IMS for z/OS	N/A	DB2 Information Integrator Support Web site

Table 182. DB2 Information Integrator Release Notes and Installation Requirements (continued)

Name	File name	Location
Release Notes for IBM DB2 Information Integrator Event Publisher for VSAM for z/OS	N/A	DB2 Information Integrator Support Web site
Release Notes for IBM DB2 Information Integrator Classic Federation for z/OS	N/A	DB2 Information Integrator Support Web site
Release Notes for Enterprise Search	N/A	DB2 Information Integrator Support Web site

To view the installation requirements and release notes that are on the product CD:

- On Windows operating systems, enter:

  `x:\doc\%L`

  x is the Windows CD drive letter and *%L* is the locale of the documentation that you want to use, for example, `en_US`.
- On UNIX operating systems, enter:

  `/cdrom/doc/%L/`

  cdrom refers to the UNIX mount point of the CD and *%L* is the locale of the documentation that you want to use, for example, `en_US`.

Accessibility

Accessibility features help users with physical disabilities, such as restricted mobility or limited vision, to use software products successfully. The following list specifies the major accessibility features in DB2® Version 8 products:

- All DB2 functionality is available using the keyboard for navigation instead of the mouse. For more information, see "Keyboard input and navigation."
- You can customize the size and color of the fonts on DB2 user interfaces. For more information, see "Accessible display."
- DB2 products support accessibility applications that use the Java™ Accessibility API. For more information, see "Compatibility with assistive technologies" on page 650.
- DB2 documentation is provided in an accessible format. For more information, see "Accessible documentation" on page 650.

Keyboard input and navigation

Keyboard input

You can operate the DB2 tools using only the keyboard. You can use keys or key combinations to perform operations that can also be done using a mouse. Standard operating system keystrokes are used for standard operating system operations.

For more information about using keys or key combinations to perform operations, see Keyboard shortcuts and accelerators: Common GUI help.

Keyboard navigation

You can navigate the DB2 tools user interface using keys or key combinations.

For more information about using keys or key combinations to navigate the DB2 Tools, see Keyboard shortcuts and accelerators: Common GUI help.

Keyboard focus

In UNIX® operating systems, the area of the active window where your keystrokes will have an effect is highlighted.

Accessible display

The DB2 tools have features that improve accessibility for users with low vision or other visual impairments. These accessibility enhancements include support for customizable font properties.

Font settings

You can select the color, size, and font for the text in menus and dialog windows, using the Tools Settings notebook.

For more information about specifying font settings, see Changing the fonts for menus and text: Common GUI help.

Non-dependence on color

You do not need to distinguish between colors in order to use any of the functions in this product.

Compatibility with assistive technologies

The DB2 tools interfaces support the Java Accessibility API, which enables you to use screen readers and other assistive technologies with DB2 products.

Accessible documentation

Documentation for DB2 is provided in XHTML 1.0 format, which is viewable in most Web browsers. XHTML allows you to view documentation according to the display preferences set in your browser. It also allows you to use screen readers and other assistive technologies.

Syntax diagrams are provided in dotted decimal format. This format is available only if you are accessing the online documentation using a screen-reader.

Related concepts:

- "Dotted decimal syntax diagrams" in the *Infrastructure Topics (DB2 Common Files)*

Related tasks:

- "Keyboard shortcuts and accelerators: Common GUI help"
- "Changing the fonts for menus and text: Common GUI help"

Notices

Trademarks

Index

Contacting IBM

To contact IBM customer service in the United States or Canada, call 1-800-IBM-SERV (1-800-426-7378).

To learn about available service options, call one of the following numbers:
- In the United States: 1-888-426-4343
- In Canada: 1-800-465-9600

To locate an IBM office in your country or region, see the IBM Directory of Worldwide Contacts on the Web at www.ibm.com/planetwide.

Product information

Information about DB2 Information Integrator is available by telephone or on the Web.

If you live in the United States, you can call one of the following numbers:
- To order products or to obtain general information: 1-800-IBM-CALL (1-800-426-2255)
- To order publications: 1-800-879-2755

On the Web, go to www.ibm.com/software/data/integration/db2ii/support.html. This site contains the latest information about:
- The technical library
- Ordering books
- Client downloads
- Newsgroups
- Fix packs
- News
- Links to Web resources

Comments on the documentation

Your feedback helps IBM to provide quality information. Please send any comments that you have about this book or other DB2 Information Integrator documentation. You can use any of the following methods to provide comments:
- Send your comments using the online readers' comment form at www.ibm.com/software/data/rcf.
- Send your comments by e-mail to comments@us.ibm.com. Include the name of the product, the version number of the product, and the name and part number of the book (if applicable). If you are commenting on specific text, please include the location of the text (for example, a title, a table number, or a page number).

Printed in USA

Spine information:

IBM DB2 Information Integrator Data Source Configuration Guide Version 8.2