IBM DB2 Information Integrator

Addendum to the Data Source Configuration Guide: BioRS Wrapper and Life Sciences User-Defined Functions

Version 8

IBM DB2 Information Integrator

Addendum to the Data Source Configuration Guide: BioRS Wrapper and Life Sciences User-Defined Functions

Version 8

Before using this information and the product it supports, be sure to read the general information under “Notices” on page 85.

You can order IBM publications online or through your local IBM representative:

- To order publications online, go to the IBM Publications Center at www.ibm.com/shop/publications/order
- To find your local IBM representative, go to the IBM Directory of Worldwide Contacts at www.ibm.com/planetwide

When you send information to IBM, you grant IBM a nonexclusive right to use or distribute the information in any way it believes appropriate without incurring any obligation to you.

Contents

Chapter 1. Configuring access to BioRS data sources

This chapter explains what BioRS is, how to add BioRS data sources to your federated system, and 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*. Only the elements of a databank that are indexed in the BioRS system can be used in a BioRS query. 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.

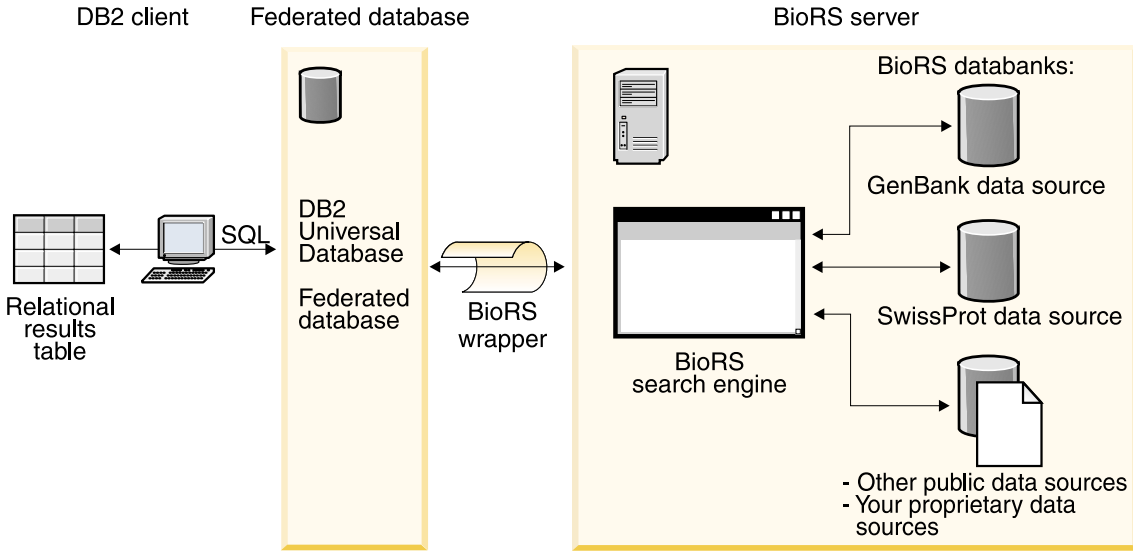


Figure 1. How the BioRS wrapper works

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 DB2® Universal Database 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 to a federated system" on page 3

Related reference:

- "BioRS wrapper - Example queries" on page 19

Adding BioRS to a federated system

You can use a BioRS data source with your federated server by registering custom functions and a BioRS wrapper. You then register a corresponding BioRS server, user mappings, and nicknames to enable your federated server to retrieve and process BioRS data.

You can run the SQL statements from the DB2 Control Center or from the DB2 command line processor. After you add BioRS to your federated system, you can run queries on a BioRS data source.

Procedure:

To add a BioRS data source to a federated server:

1. Register custom functions by using the CREATE FUNCTION statement.
2. Register the BioRS wrapper by using the CREATE WRAPPER statement.
3. Optional: Set the DB2_DJ_COMM environment variable to improve query performance.
4. Register the BioRS server by using the CREATE SERVER statement.
5. Optional: Register authorized users with the CREATE USER MAPPING statement.
6. Register nicknames by using the CREATE NICKNAME statement.
7. Optional: Update cardinality statistics for BioRS columns.

Related tasks:

- “Registering custom functions for the BioRS wrapper” on page 3
- “Registering the BioRS wrapper” on page 4
- “Setting the DB2_DJ_COMM profile variable for the BioRS wrapper” on page 5
- “Registering the server for a BioRS data source” on page 6
- “Registering user mappings for BioRS data sources” on page 6
- “Registering nicknames for BioRS data sources” on page 8
- “Updating BioRS column cardinality statistics” on page 11

Registering custom functions for the BioRS wrapper

Registering custom functions for the BioRS wrapper is part of the larger task of adding BioRS to a federated system. After the custom functions are registered, you must register the wrapper.

You can use the sample file create_function_mappings.ddl to register custom functions. This file is in the sqllib/samples/lifesci/biors directory. The

create_function_mappings.ddl file contains definitions for each custom function. You can run this DDL file to register the custom functions for each DB2 database where the BioRS wrapper is installed.

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 DB2 database where the BioRS wrapper is installed.

Procedure:

To register custom functions, issue the CREATE FUNCTION statement with the AS TEMPLATE keyword.

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;
```

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

Related tasks:

- "Registering the BioRS wrapper" on page 4

Related reference:

- "CREATE FUNCTION (Sourced or Template) statement" in the *SQL Reference, Volume 2*
- "Custom functions and BioRS queries" on page 14
- "BioRS wrapper - Example queries" on page 19
- "Custom function table - BioRS wrapper" on page 30

Registering the BioRS wrapper

Registering the BioRS wrapper is part of the larger task of adding BioRS to a federated system. You must register the wrapper to access a data source. Wrappers are mechanisms that federated servers use to communicate with and retrieve data from the data sources. Wrappers are installed on your system as library files. Table 1 on page 5 lists the default BioRS library files and the supported operating system for each file.

Table 1. BioRS library files and supported operating systems

BioRS library file	Operating system
libdb2lsbiors.a	IBM AIX Version 4.3.3 or later
db2lsbiors.dll	Microsoft Windows NT Version 4 Microsoft Windows 2000 Microsoft Windows XP

Procedure:

To register the BioRS wrapper, issue the CREATE WRAPPER statement.

For example, to register a BioRS wrapper on AIX called `wrap_biors` from the default library file, `libdb2lsbiors.a`, issue the following statement:

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

Related tasks:

- "Checking the nonrelational wrapper and life sciences user-defined function libraries" in the *DB2 Information Integrator Installation Guide*
- "Setting the DB2_DJ_COMM profile variable for the BioRS wrapper" on page 5

Related reference:

- "CREATE WRAPPER statement" in the *SQL Reference, Volume 2*

Setting the DB2_DJ_COMM profile variable for the BioRS wrapper

Setting the DB2_DJ_COMM DB2 profile variable for the BioRS wrapper is part of the larger task of adding BioRS to a federated system. To improve performance when BioRS data sources are accessed, you can optionally set the DB2_DJ_COMM DB2 profile variable. This variable specifies whether the federated server loads the wrapper upon initialization.

Processor usage increases when the federated server loads the wrapper libraries during database startup. To avoid excessive usage, specify only the libraries that you intend to access.

Procedure:

To set the DB2_DJ_COMM DB2 profile variable, issue the **db2set** command with the wrapper library that corresponds to the wrapper that you specified in the associated CREATE WRAPPER statement.

For example:

```
db2set DB2_DJ_COMM='libdb2lsbiors.a'
```

Ensure that there are no spaces on either side of the equal sign (=).

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

Related concepts:

- "Environment Variables and the Profile Registry" in the *Administration Guide: Implementation*

Related tasks:

- "Registering the server for a BioRS data source" on page 6

Related reference:

- "db2set - DB2 Profile Registry Command" in the *Command Reference*

Registering the server for a BioRS data source

Registering the server 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.

Procedure:

To register the BioRS server to the federated system, issue a CREATE SERVER statement.

For example:

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

Related tasks:

- "Registering nicknames for BioRS data sources" on page 8

Related reference:

- "CREATE SERVER statement options - BioRS wrapper" on page 37

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 wrapper.

Related tasks:

- “Registering nicknames for BioRS data sources” on page 8

Related reference:

- “CREATE USER MAPPING statement” in the *SQL Reference, Volume 2*

• “CREATE USER MAPPING statement options - BioRS wrapper” on page 38

Registering nicknames for BioRS data sources

Registering nicknames for BioRS data sources is part of the larger task of adding BioRS to a federated system. After you register a server, 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 nicknames.

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

Prerequisites:

- If a BioRS databank name does not conform to DB2 federated syntax, you must use the REMOTE_OBJECT nickname option when you register the nickname.
- If a BioRS element name does not conform to DB2 federated syntax, 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.

A federated nickname equates directly to a BioRS databank. When you create a federated 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. These data types can be mapped only to the CLOB, CHAR, or VARCHAR federated system data types.

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 underlying BioRS databank SwissProt is the name of the nickname.

Using this simple CREATE NICKNAME syntax limits you to one family of nicknames per DB2 schema. You can use other names by specifying the REMOTE_OBJECT option. This nickname option specifies the name of the BioRS object type to be associated with the nickname. The name that is specified 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');
```

The underlying BioRS databank is SwissProt, and the name of the nickname is NewSP.

Related concepts:

- "BioRS statistical information" on page 26

Related tasks:

- "Updating BioRS nickname cardinality statistics" on page 27

Related reference:

- "The BioRS AllText element" on page 29
- "CREATE NICKNAME statement - Examples for BioRS wrapper" on page 9
- "CREATE NICKNAME statement syntax - BioRS wrapper" on page 35
- "Considerations for altering nicknames - BioRS wrapper" on page 30

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 8

Related reference:

- "CREATE NICKNAME statement syntax - BioRS wrapper" on page 35
- "Considerations for altering nicknames - BioRS wrapper" on page 30

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 26

Related tasks:

- "Updating BioRS nickname cardinality statistics" on page 27
- "Updating BioRS _ID_ column cardinality" on page 28

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 18
- “BioRS statistical information” on page 26

Related reference:

- “Custom functions and BioRS queries” on page 14
- “BioRS wrapper - Example queries” on page 19

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 2.

Table 2. 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 2. 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 2 on page 15.

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;
```

If you specify the BIORS.SEARCH_TERM function, you cannot use any other custom functions in a query. However, you can use any combination of the BIORS.CONTAINS, BIORS.CONTAINS_GE, and BIORS.CONTAINS_LE functions in the same query.

Related concepts:

- "Pushdown analysis" in the *Federated Systems Guide*
- "Guidelines for optimizing BioRS wrapper performance" on page 13
- "Equijoin predicates for the BioRS wrapper" on page 18

Related tasks:

- "Registering custom functions for the BioRS wrapper" on page 3

Related reference:

- "BioRS wrapper - Example queries" on page 19
- "Custom function table - BioRS wrapper" on page 30

Equijoin predicates for the BioRS wrapper

You can specify predicates for the BioRS engine using the four 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.

Suppose 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 13

Related reference:

- “Custom functions and BioRS queries” on page 14
- “BioRS wrapper - Example queries” on page 19

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 3 illustrate how you can structure your queries to optimize the workload between the federated system and the BioRS server.

Table 3. 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 3 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 4 show the use of wildcard characters in the BIORS.CONTAINS function. All of the query results in Table 4 are identical, even though different wildcard characters are used.

Table 4. 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.

Table 4. Sample queries that use wildcards in the BIORS.CONTAINS function (continued)

Query	Result
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 5 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 5 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 5. 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.

Table 5. Sample queries that access BioRS Author data type columns (continued)

Query	Result
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 6 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 6. 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 6. 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 7 on page 24 show how you can use the BIORS.CONTAINS_LE and the BIORS.CONTAINS_GE functions.

Table 7. 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 8 show how you can use the BIORS.SEARCH_TERM function to specify a search term using the BioRS format.

Table 8. 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.

Table 8. Sample queries that use the BIORS.SEARCH_TERM function (continued)

Query	Result
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 13
- "Equijoin predicates for the BioRS wrapper" on page 18

Related reference:

- "Custom functions and BioRS queries" on page 14
- "CREATE NICKNAME statement - Examples for BioRS wrapper" on page 9
- "CREATE NICKNAME statement syntax - BioRS wrapper" on page 35

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 27
- "Updating BioRS nickname cardinality statistics" on page 27
- "Updating BioRS column cardinality statistics" on page 11
- "Updating BioRS _ID_ column cardinality" on page 28

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 26

Related tasks:

- "Updating BioRS nickname cardinality statistics" on page 27
- "Updating BioRS column cardinality statistics" on page 11
- "Updating BioRS _ID_ column cardinality" on page 28

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 26

Related tasks:

- "Determining BioRS databank cardinality statistics" on page 27
- "Updating BioRS column cardinality statistics" on page 11
- "Updating BioRS _ID_ column cardinality" on page 28

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 26

Related tasks:

- "Determining BioRS databank cardinality statistics" on page 27
- "Updating BioRS nickname cardinality statistics" on page 27
- "Updating BioRS column cardinality statistics" on page 11

The BioRS AllText element

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

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

If a column that maps the AllText element is referenced in the SELECT list of a query, a value of NULL will always be returned.

Related tasks:

- "Registering nicknames for BioRS data sources" on page 8

Related reference:

- "BioRS wrapper - Example queries" on page 19

Considerations for altering nicknames - BioRS wrapper

You can modify previously registered BioRS nicknames using the ALTER NICKNAME statement. With the ALTER NICKNAME statement, you can:

- Change the name of a column
- Change the data type of a column
- Add, change, or delete options for a column

Restrictions:

You cannot change the name of the BioRS databank that is referenced by or used in a nickname. If the name of a BioRS databank that is used in a nickname changes, you must drop and then recreate the entire nickname.

If you specified the REMOTE_OBJECT option, you cannot change or delete the option value.

If you change the data type of a column, the new data type must be compatible with the the data type of the corresponding BioRS element.

If you change the element name of a column 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.

Related reference:

- "ALTER NICKNAME statement" in the *SQL Reference, Volume 2*

Custom function table - BioRS wrapper

Table 9 on page 31 provides examples of how you can register each of the four BioRS custom functions.

To assist you in registering custom functions, the sample file create_function_mappings.ddl is provided in the sqllib/samples/lifesci/biors directory. The create_function_mappings.ddl file contains definitions for each custom function. You can run this DDL file to register the custom functions for each DB2 database that has the BioRS wrapper installed.

Table 9. Custom functions for the BioRS wrapper

Function name	Description
CONTAINS (col VARCHAR(), term VARCHAR()), CONTAINS (col VARCHAR(), term CHAR()) CONTAINS (col VARCHAR(), term DATE) CONTAINS (col VARCHAR(), term TIMESTAMP)	Searches an indexed column using the given expression. **col** Indexed column. **term** Search term.
CONTAINS_LE (col VARCHAR(), term VARCHAR()), CONTAINS_LE (col VARCHAR(), term SMALLINT) CONTAINS_LE (col VARCHAR(), term BIGINT) CONTAINS_LE (col VARCHAR(), term DECIMAL) CONTAINS_LE (col VARCHAR(), term DOUBLE) CONTAINS_LE (col VARCHAR(), term REAL)	Searches an indexed column using the given expression. **col** Indexed column. **term** Search term.
CONTAINS_GE (col CHAR(), term CHAR()) CONTAINS_GE (col CHAR(), term DATE) CONTAINS_GE (col CHAR(), term TIMESTAMP) CONTAINS_GE (col CHAR(), term INTEGER) CONTAINS_GE (col CHAR(), term SMALLINT) CONTAINS_GE (col CLOB(), term DATE)	Searches an indexed column using the given expression. **col** Indexed column. **term** Search term.
SEARCH_TERM (col VARCHAR(), term VARCHAR()) SEARCH_TERM (col VARCHAR(), term CHAR()) SEARCH_TERM (col CHAR(), term VARCHAR()) SEARCH_TERM (col CHAR(), term CHAR())	Passes a BioRS search term to the BioRS search engine. **col** Indexed column. **term** Search term.

Related tasks:

- "Registering custom functions for the BioRS wrapper" on page 3

Messages for the BioRS wrapper

This topic explains messages that you might receive when you work with the wrapper for BioRS. For more information about messages, see the *DB2 Message Reference*.

Table 10. 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.
SQL401N	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.

Table 10. Messages issued by the wrapper for BioRS (continued)

Error Code	Message	Explanation
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.

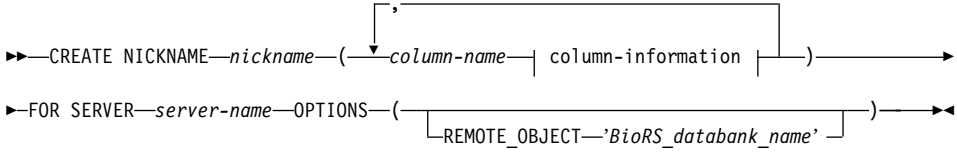
Table 10. Messages issued by the wrapper for BioRS (continued)

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.

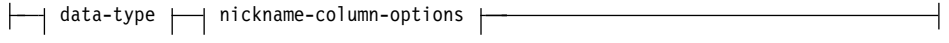
Table 10. Messages issued by the wrapper for BioRS (continued)

Error Code	Message	Explanation
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.

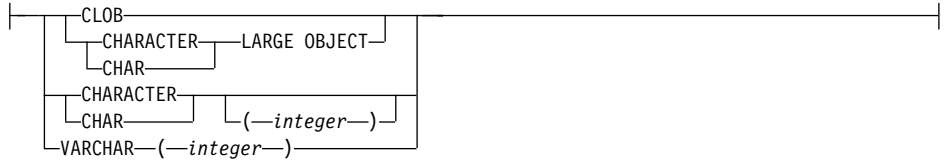
CREATE NICKNAME statement syntax - BioRS wrapper



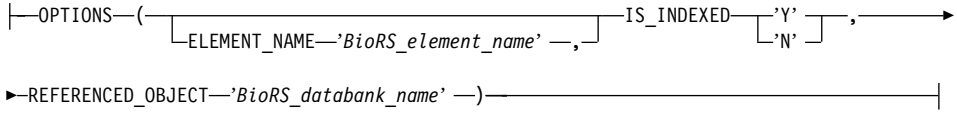
column-information:



data-type:



nickname-column-options:



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 8

Related reference:

- "CREATE NICKNAME statement" in the *SQL Reference, Volume 2*
- "CREATE NICKNAME statement - Examples for BioRS wrapper" on page 9

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*. You must specify this option.

PORT Specifies the number of the port to be used to connect to the BioRS server. The default value is 5014. You must specify this option.

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 for a BioRS data source" on page 6
- "Registering nicknames for BioRS data sources" on page 8

Related reference:

- "CREATE SERVER statement" in the *SQL Reference, Volume 2*
- "CREATE NICKNAME statement syntax - BioRS wrapper" on page 35

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 6

Related reference:

- "CREATE USER MAPPING statement" in the *SQL Reference, Volume 2*

Chapter 2. 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

The life sciences user-defined functions provide researchers with algorithms they commonly use to analyze data. The functions are available on Windows® NT, AIX, and Linux 32–bit platforms, except the GeneWise function which is available on AIX® and Linux platforms.

The life sciences user-defined functions use the standard single-letter codes and the IUPAC-IUB ambiguity codes to represent amino acids and nucleotides.

Before you can use the life sciences user-defined functions, you must register them. After you've registered them, you can remove them if needed.

Related tasks:

- "Registering life sciences user-defined functions" on page 40
- "Removing life sciences user-defined functions" on page 41

Related reference:

- "Life sciences user-defined functions by functional category" on page 39

Life sciences user-defined functions by functional category

Table 11 lists the life sciences user-defined functions by functional category. It also provides a brief description of each category.

Table 11. 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.

Table 11. Life sciences user-defined functions (continued)

Functional category	User-defined functions	Description
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.
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 39

Related tasks:

- "Registering life sciences user-defined functions" on page 40
- "Removing life sciences user-defined functions" on page 41

Registering life sciences user-defined functions

Before you can use the life sciences user-defined functions, you must register them.

Procedure:

To register the life sciences user-defined functions, use the enable_LSFunctions command available on Windows NT and AIX 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 database to which you will register the functions.

userID
A valid user ID for the database.

password
A valid password for the user ID.

force A flag you can use to remove the functions and register them again. You can use this flag to register the functions again if they get corrupted or dropped accidentally.

The command creates the functions with the schema name DB2LS.

The following example shows sample output for the enable_LSFunctions command:

```
C:> enable_LSFunctions -n test -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 output for the enable_LSFunctions command when you use the force flag and the functions are already registered:

```
C:> enable_LSFunctions -n test -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 ......
```

Removing life sciences user-defined functions

If you no longer want the life sciences user-defined functions on your system, you can remove them.

Procedure:

To remove the life sciences user-defined functions, use the disable_LSFunctions command available on Windows NT and AIX 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 database from which you want to remove the functions.

userID
A valid user ID for the database.

password
A valid password for the user ID.

The following example shows the output from the disable_LSFunctions command:

```
C:>disable_LSFunctions -n test -u db2admin -p db2admin
a

(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 ......
```

Back translation user-defined functions

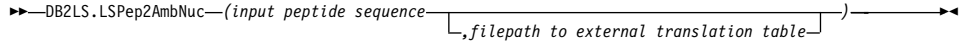
Use the back–translation user-defined functions to convert a peptide sequence to a nucleotide sequence. Back translation is the inverse of translation.

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.

LSPep2AmbNuc user-defined function



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 12 by default.

Table 12. 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
P	Pro	CCX
Q	Gln	CAR
R	Arg	MGX
S	Ser	WSX
T	Thr	ACX
V	Val	GTX
W	Trp	TGG

Table 12. Default translation table (continued)

Amino acid symbol	Abbreviation	Codon
X	Xxx	XXX
Y	Tyr	TAY
Z	Glx	SAR
*	End	TRR

Related reference:

- "LSPep2AmbNuc user-defined function - error messages" on page 45
- "LSPep2ProbNuc user-defined function" on page 46
- "LSPep2AmbNuc user-defined function - example" on page 44

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 13. 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 42
- "LSNuc2Pep user-defined function – example" on page 75

LSPep2AmbNuc user-defined function - error messages

Table 14. 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.

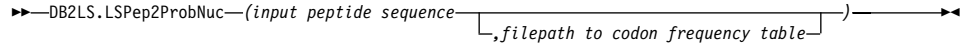
Table 14. Messages issued by the LSPep2AmbNuc user-defined function (continued)

Error code	Message	Explanation
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.
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 42

LSPep2ProbNuc user-defined function



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 42
- "LSPep2ProbNuc user-defined function - error messages" on page 48
- "LSPep2ProbNuc user-defined function - example" on page 47

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 we have the following data in a frequency table:

Table 15. 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 46
- “LSPep2ProbNuc user-defined function - error messages” on page 48
- “LSPep2AmbNuc user-defined function - example” on page 44

LSPep2ProbNuc user-defined function - error messages

Table 16. 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.
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.

Table 16. 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 ″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.

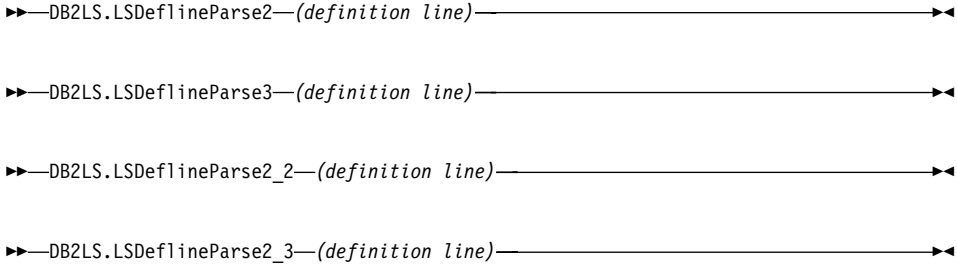
Related reference:

- “LSPep2ProbNuc user-defined function” on page 46
- “LSPep2ProbNuc user-defined function - example” on page 47

Defline parsing user-defined functions

Defline parsing user-defined functions parse elements of a definition line, for example, to enable joins with other data sources on sequence identifiers parsed out of the defline, or to evaluate predicates on portions of the defline, such as `'species = "human"'`. The defline functions cover the most common defline formats. Examples include definition line elements the BLAST wrapper returns or that are present in a FASTA format data file.

LSDeflineParse user-defined functions



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 schema name is DB2LS.

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.

LSDeflineParse2 parses a defline that has a two-field NSID. The result of the function is a table with four columns:

Table 17. 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.

LSDeflineParse3 parses a defline that has a three-field NSID. The result of the function is a table with five columns:

Table 18. LSDeflineParse3 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.
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.

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 19. 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.

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 20. 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.
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.

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 21. 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 52

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 22. LSDeflineParse2 user-defined function result data

Column name	Data
ROWID	1
TAG	gi

Table 22. LSDeflineParse2 user-defined function result data (continued)

Column name	Data
IDENTIFIER	12346
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 23. 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 24. 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 25. 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 26. 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 27. LSDeflineParse3_3 user-defined function result data

Column name	Data
ROWID	1
TAG1	dbj
ACCESSION1	AAD55586.1
LOCUS1	AF055084_1
TAG2	gp
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 28. 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

Table 28. LSDeflineParse2_3 user-defined function result data (continued)

Column name	Data	Data
LOCUS	CHTAHSRA_4	CHTAHSRA_4
DESCRIPTION	hypothetical protein 185 – wheat chloroplast	hypothetical protein 185 – wheat chloroplast

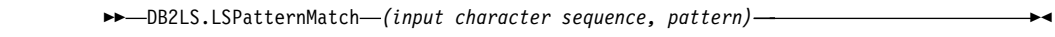
Related reference:

- "LSDeflineParse user-defined functions" on page 49

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



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 57
- "LSPrositePattern user-defined function" on page 59

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
```

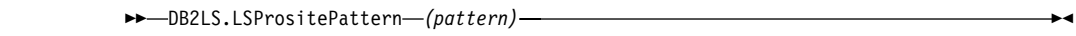
This second query will return the blast rows that have a match together with the matched string and their position in the sequence.

BlastOutput is a view over a BlastN nickname.

Related reference:

- "LSPrositePattern user-defined function - example" on page 59
- "LSPatternMatch user-defined function" on page 56
- "LSPrositePattern user-defined function" on page 59

LSPrositePattern user-defined function



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 59
- "LSPatternMatch user-defined function" on page 56

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 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 57
- "LSPrositePattern user-defined function" on page 59

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/.

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:

1. Download the Wise2 package version 2.1.20c from http://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.
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.

   The For example, if Wise2 package is installed in /usr/wise2.1.20c/, then add WISECONFIGDIR=/usr/wise2.1.20c/wisecfg/ into db2dj.ini file.
7. Run **djxlinkLSGeneWise** from the sqllib/bin directory and check its output.
8. Check djxlinkLSGeneWise.out in sqllib/lib directory.
9. If no errors were reported, then the library was successfully built.

Related reference:

- "LSGeneWise user-defined function" on page 61

LSGeneWise user-defined function

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 29 on page 62 shows the one row output table the LSGeneWise function returns.

Table 29. 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 61

Related reference:

- "LSGeneWise user-defined function – example" on page 63

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 30. 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
TRANSLATED_DNA	KIFVGGIKEDTEEHHLRDYFEQYGKIEVIEIMTDRGSGK KRGFAxVTFDDHDSVDKIVIQKYHTVNGHNCEVRKAL SKQEMASASSSQRGRSGS------ GNFGGGRGGGFGGNDNFGRGGN
DNA	aagatatttgttggtggcattaaagaagacactgaagaacatcacctaagagat...

Related tasks:

- "Linking to GeneWise" on page 61

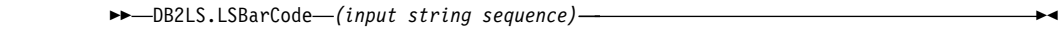
Related reference:

- "LSGeneWise user-defined function" on page 61

Motifs user-defined functions

Motif user-defined functions match patterns in nucleotide or amino acid sequences.

LSBarCode user-defined function



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 64
- "LSMultiMatch user-defined function" on page 66
- "LSMultiMatch3 user-defined function" on page 67

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 31.

Table 31. 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

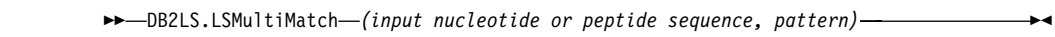
The previous query was adapted from Extending traditional query-based integration approaches for functional characterization of post-genomic data. (2001) Barbara A Eckman, Anthony S Kosky and Leonardo A Laroco Jr. *Bioinformatics* 17(7), 587-601.

Related reference:

- "LSMultiMatch3 user-defined function – example" on page 68

- "LSBarCode user-defined function" on page 64

LSMultiMatch user-defined function



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 66
- "LSBarCode user-defined function" on page 64
- "LSMultiMatch3 user-defined function" on page 67

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 32. Result of LSMultiMatch returning multiple rows

POSITION	MATCH
7	not

Table 32. Result of LSMultiMatch returning multiple rows (continued)

POSITION	MATCH
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:

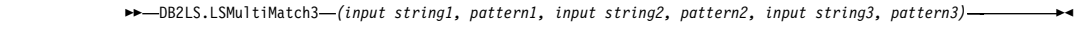
Table 33. Search Entrez data

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 66
- "LSBarCode user-defined function" on page 64
- "LSMultiMatch3 user-defined function" on page 67

LSMultiMatch3 user-defined function



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 68
- "LSMultiMatch user-defined function" on page 66
- "LSBarCode user-defined function" on page 64

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 34.

Table 34. 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 35. 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',
   '.','01234567890123456789012345678901234567890123456789','\d')) as f
```

The results are in the following table:

Table 36. Result of a multi-match using three inputs

POSITION	MATCH_1	MATCH_2	MATCH_3
1	cbcc	a	0
7	cccbbbccc	g	6

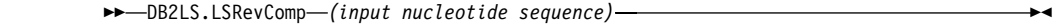
Related reference:

- "LSBarCode user-defined function — example" on page 64
- "LSBarCode user-defined function" on page 64
- "LSMultiMatch3 user-defined function" on page 67

Reverse user-defined functions

Reverse user-defined functions reverse a nucleotide or amino acid sequence.

LSRevComp user-defined function



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 70
- "LSRevNuc user-defined function" on page 72
- "LSRevPep user-defined function" on page 73

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 70

LSRevNuc user-defined function

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 72
- "LSRevComp user-defined function" on page 70
- "LSRevPep user-defined function" on page 73

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='gtaatacgtaggggctagcgcgggcaaactgaagataaagc'
```

The following results table shows the reversed nucleotide sequences the query returns:

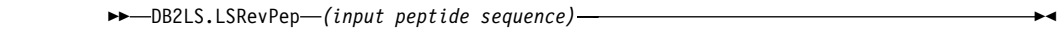
```
HSP_H_SEQ                                          REV_HSP_H_SEQ

-------------------------------------------------- --------------------------
```

```
CGCGGGCAAACTGAAGATAAAGC                CGAAATAGAAGTCAAACGGGCGC

GCGCTAGCCCCCTACGTATTAC                 CATTATGCATCCCCCGATCGCG

GTAATACGTAGGGGGCTAGCG                  GCGATCGGGGGATGCATAATG

GTAATACGTAGGGGGCTAGCG                  GCGATCGGGGGATGCATAATG

GTAATACGTAGGGGGCTAGCG                  GCGATCGGGGGATGCATAATG


  5 record(s) selected.
```

Related reference:

- "LSRevNuc user-defined function" on page 72

LSRevPep user-defined function



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 73
- "LSRevComp user-defined function" on page 70
- "LSRevNuc user-defined function" on page 72

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.
```

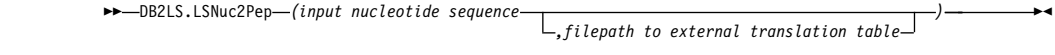
Related reference:

- "LSRevPep user-defined function" on page 73

Translate

The translation user-defined functions convert a nucleotide sequence into a peptide sequence.

LSNuc2Pep user-defined function



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 75
- "LSTransAllFrames user-defined function" on page 76

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 37.

Table 37. 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..')
```

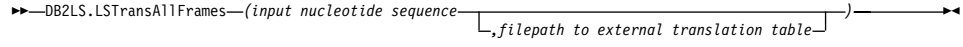
Notice that the previous example returns the same result as the following query:

```
select * from table (DB2LS.LSTransAllFrames ('TTT..')) as t
where t.readframe = -1
```

Related reference:

- "LSRevNuc user-defined function - example" on page 72
- "LSTransAllFrames user-defined function - example" on page 77
- "LSNuc2Pep user-defined function" on page 74

LSTransAllFrames user-defined function



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 77
- "LSNuc2Pep user-defined function" on page 74

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 38. Result of translating a nucleotide sequence

READFRAME	PEPTIDE
1	FFLLSSSSYFLCC*C
2	FSYCLPHRISYVAD

Table 38. Result of translating a nucleotide sequence (continued)

READFRAME	PEPTIDE
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 39. Readframe function usage

READFRAME	PEPTIDE
–2	HQQHKKYDEEDNKK

Related reference:

• "LSNuc2Pep user-defined function – example" on page 75

- “LSRevNuc user-defined function - example” on page 72
- “LSTransAllFrames user-defined function” on page 76

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 46
- “Codon frequency table - example” on page 79

Codon frequency table - example

Figure 2 on page 80 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 2. Sample codon frequency table

Related reference:

- "LSPep2ProbNuc user-defined function" on page 46
- "Codon frequency table format" on page 79

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.

Translation table - example

Figure 3 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 3. Sample translation table

Accessibility

Users with physical disabilities, such as restricted mobility or limited vision, can use software products successfully by using accessibility features. These are the major accessibility features in DB2 Information Integrator Version 8:

- You can operate all features by using the keyboard instead of the mouse.
- You can customize the size and color of your fonts.
- You can receive either visual or audio alert cues.
- DB2 supports accessibility applications that use the Java™ Accessibility API.
- DB2 documentation is provided in an accessible format.

Keyboard input and navigation

You can operate the DB2 database tools, such as Control Center, Data Warehouse Center, and Replication Center, by using only the keyboard. You can use keys or key combinations instead of a mouse to perform most operations.

In UNIX-based systems, the position of the keyboard focus is highlighted. This highlighting indicates which area of the window is active and where your keystrokes will have an effect.

Accessible display

The DB2 database tools have features that enhance the user interface and improve accessibility for users with low vision. These accessibility enhancements include support for customizable font properties.

Font settings

For the DB2 database tools, you can use the Tools Settings notebook to select the color, size, and font for the text in menus and windows.

Nondependence on color

You do not need to distinguish between colors to use any of the functions in this product.

Alternative alert cues

You can specify whether you want to receive alerts through audio or visual cues, using the Tools Settings notebook.

Compatibility with assistive technologies

The DB2 Information Integrator graphical interface supports the Java Accessibility API, enabling the use of screen readers and other assistive technologies that are used by people with disabilities.

Accessible documentation

Documentation for the DB2 family of products is available in HTML format. You can view documentation according to the display preferences set in your browser. You can use screen readers and other assistive technologies.

Notices

This information was developed for products and services offered in the U.S.A. IBM may not offer the products, services, or features discussed in this document in all countries. Consult your local IBM representative for information on the products and services currently available in your area. Any reference to an IBM product, program, or service is not intended to state or imply that only that IBM product, program, or service may be used. Any functionally equivalent product, program, or service that does not infringe any IBM intellectual property right may be used instead. However, it is the user's responsibility to evaluate and verify the operation of any non-IBM product, program, or service.

IBM may have patents or pending patent applications covering subject matter described in this document. The furnishing of this document does not give you any license to these patents. You can send license inquiries, in writing, to:

IBM Director of Licensing
IBM Corporation
North Castle Drive
Armonk, NY 10504-1785
U.S.A.

For license inquiries regarding double-byte (DBCS) information, contact the IBM Intellectual Property Department in your country/region or send inquiries, in writing, to:

IBM World Trade Asia Corporation
Licensing
2-31 Roppongi 3-chome, Minato-ku
Tokyo 106, Japan

The following paragraph does not apply to the United Kingdom or any other country/region where such provisions are inconsistent with local law: INTERNATIONAL BUSINESS MACHINES CORPORATION PROVIDES THIS PUBLICATION "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF NON-INFRINGEMENT, MERCHANTABILITY, OR FITNESS FOR A PARTICULAR PURPOSE. Some states do not allow disclaimer of express or implied warranties in certain transactions; therefore, this statement may not apply to you.

This information could include technical inaccuracies or typographical errors. Changes are periodically made to the information herein; these changes will

be incorporated in new editions of the publication. IBM may make improvements and/or changes in the product(s) and/or the program(s) described in this publication at any time without notice.

Any references in this information to non-IBM Web sites are provided for convenience only and do not in any manner serve as an endorsement of those Web sites. The materials at those Web sites are not part of the materials for this IBM product, and use of those Web sites is at your own risk.

IBM may use or distribute any of the information you supply in any way it believes appropriate without incurring any obligation to you.

Licensees of this program who wish to have information about it for the purpose of enabling: (i) the exchange of information between independently created programs and other programs (including this one) and (ii) the mutual use of the information that has been exchanged, should contact:

IBM Corporation
J46A/G4
555 Bailey Avenue
San Jose, CA 95141-1003
U.S.A.

Such information may be available, subject to appropriate terms and conditions, including in some cases payment of a fee.

The licensed program described in this document and all licensed material available for it are provided by IBM under terms of the IBM Customer Agreement, IBM International Program License Agreement, or any equivalent agreement between us.

Any performance data contained herein was determined in a controlled environment. Therefore, the results obtained in other operating environments may vary significantly. Some measurements may have been made on development-level systems, and there is no guarantee that these measurements will be the same on generally available systems. Furthermore, some measurements may have been estimated through extrapolation. Actual results may vary. Users of this document should verify the applicable data for their specific environment.

Information concerning non-IBM products was obtained from the suppliers of those products, their published announcements, or other publicly available sources. IBM has not tested those products and cannot confirm the accuracy of performance, compatibility, or any other claims related to non-IBM products. Questions on the capabilities of non-IBM products should be addressed to the suppliers of those products.

All statements regarding IBM's future direction or intent are subject to change or withdrawal without notice, and represent goals and objectives only.

This information may contain examples of data and reports used in daily business operations. To illustrate them as completely as possible, the examples include the names of individuals, companies, brands, and products. All of these names are fictitious, and any similarity to the names and addresses used by an actual business enterprise is entirely coincidental.

COPYRIGHT LICENSE:

This information may contain sample application programs, in source language, which illustrate programming techniques on various operating platforms. You may copy, modify, and distribute these sample programs in any form without payment to IBM for the purposes of developing, using, marketing, or distributing application programs conforming to the application programming interface for the operating platform for which the sample programs are written. These examples have not been thoroughly tested under all conditions. IBM, therefore, cannot guarantee or imply reliability, serviceability, or function of these programs.

Each copy or any portion of these sample programs or any derivative work must include a copyright notice as follows:

© (*your company name*) (*year*). Portions of this code are derived from IBM Corp. Sample Programs. © Copyright IBM Corp. *_enter the year or years_*. All rights reserved.

Trademarks

The following terms are trademarks of International Business Machines Corporation in the United States, other countries, or both:

IBM
AIX
DB2
Domino
Informix
Lotus
Lotus Notes
QuickPlace
WebSphere

The following terms are trademarks or registered trademarks of other companies:

Microsoft, Windows, Windows NT, and the Windows logo are trademarks of Microsoft Corporation in the United States, other countries, or both.

UNIX is a registered trademark of The Open Group in the United States and other countries.

Java and all Java-based trademarks and logos are trademarks or registered trademarks of Sun Microsystems, Inc. in the United States, other countries, or both.

Other company, product or service names may be trademarks or service marks of others.

Index

Contacting IBM

To contact IBM in the United States or Canada, call one of the following numbers:

- For customer service: 1-800-IBM-SERV (1-800-426-7378)
- For DB2 marketing and sales: 1-800-IBM-4YOU (1-800-426-4968)

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. This site contains the latest information on the technical library, ordering books, client downloads, newsgroups, fix packs, news, and links to Web resources.

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.

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 electronic mail (e-mail) to comments@us.ibm.com. Be sure to 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 U.S.A.