Version 9 Release 1

IBM InfoSphere Optim
Using Optim Designer

Version 9 Release 1

IBM InfoSphere Optim
Using Optim Designer

Note
Before using this information and the product it supports, read the information in "Notices" on page 127.

First Edition

This edition applies to version 9, release 1 of Optim Designer and to all subsequent releases and modifications until otherwise indicated in new editions.

Contents

Chapter 1. Using InfoSphere Optim Designer

Use IBM® InfoSphere® Optim™ Designer to define data models, data privacy policies, data management services, and data stores. You can define objects in Optim Designer and share them in the Optim repository. Use the Repository Explorer to manage objects in the repository. Use the Directory Explorer to view objects and transform requests in an Optim directory.

Data management services
Use a data management service to extract, convert, load, or insert data. A service uses access definitions to define the data to extract. A service also uses table maps and column maps to map source and target data. You can also transform process requests in an Optim Directory into data management services. Use Optim Manager in embedded mode to test services.

Data masking
You can mask data processed by a data management service. Use data masking to transform data such as national ID numbers, credit card numbers, dates, numeric values, and personal information. Use a column map to enter a data masking function or create a column map procedure with a LUA script.

Access definitions
Use an access definition to define data models and subsets of data in matching or different schemas. An access definition identifies selection criteria and relationships used in a data management service. You can define relationships outside of a DBMS and use SQL to define selection criteria.

Optim Manager
You can open Optim Manager from Optim Designer (embedded mode), allowing you to test data management services.

Getting started

To get started, you must use the Repository Explorer to connect to an Optim repository.

Optim Designer workspace

Use the workspace to store repository connection information.

When you open Optim Designer, use the Select Workspace window to enter the location of the Optim Designer workspace.

If you change the location of the workspace, you will need to enter repository connection information.

Optim Perspective

In Optim Designer, the Optim perspective provides tools you need to define data models, data management services, and connect to an Optim directory. When you first open Optim Designer, the Optim perspective is the default perspective.

The Optim perspective includes the following views:

Repository Explorer
Use the Repository Explorer to define data store aliases, access definitions, data management services, and other objects stored in an Optim repository.

Directory Explorer
Use the Directory Explorer to connect to an Optim directory.

To open the Optim perspective, click **Window** > **Open Perspective** > **Other**. In the Open Perspective window, select **Optim**.

Repository Explorer

Use the Repository Explorer to define data store aliases, access definitions, data management services, and other objects stored in an Optim repository.

When Optim Designer is connected to an Optim repository, the Repository Explorer includes the following objects:

Data Store Aliases

A data store alias is a user-defined object associated with a database. When you define a data store alias, you provide parameters that Optim uses to communicate with that database. These parameters include the type and version of the database management system (DBMS) and the properties for both a client connection and a JDBC connection.

A data store alias name serves as a high-level qualifier that allows you to access a specific database to perform requested functions. For example, in an access definition, you must qualify the name of a table with a data store alias name. The referenced data store alias supplies the parameters needed to connect to the database in which the table resides.

File data store aliases are created by data management services and contain data that can be used in a convert, insert, or load service.

Optim Primary Keys

A primary key is the column or columns that contain values that uniquely identify each row in a table. A database table must have a primary key for Optim to insert, update, restore, or delete data from a database table. Optim uses primary keys that are defined to the database. However, you can also define Optim primary keys to supplement the primary keys in the database.

Optim Relationships

Optim uses relationships to determine the data to be retrieved from related tables and relies upon relationships defined to the database, when available. However, you can also define Optim relationships to supplement relationships in the database.

Access Definitions

Use access definitions to specify the tables, relationship traversal, and selection criteria for the data you want to process.

Column Maps

A column map provides specifications needed to match or exclude columns from processing in a data management service. Convert, insert, and load services must reference a table map, which may reference one or more column maps. You can use a column map to define data transformations with functions or column map procedures.

Table Maps

Use a table map to define specifications for correlating source and destination tables of compatible data. You can map tables that have different names, modify table names, exclude tables from a process, or include column maps for greater control over the data.

Services

Use a data management service to extract, convert, load, or insert data. A service uses access definitions to define the data to extract. A service also uses table maps and column maps to map source and target data. You can mask data by applying a data privacy policy to an entity processed by a service. You can also transform process requests in an Optim directory into data management services. Use Optim Manager in embedded mode to test services.

Folders

A Repository Explorer folder contains services, access definitions, column maps, and table maps. Use the folders to organize these objects.

Creating a Repository Explorer folder:

Use the New Folder window to create a folder in the Repository Explorer.

A Repository Explorer folder contains services, access definitions, column maps, and table maps. Use the folders to organize these objects.

To create a Repository Explorer folder.

1. Depending on whether a folder exists, do one of the following steps:
   a. To create an initial folder, click **Click to create a new folder** in the Repository Explorer.
   b. To create an additional folder, right-click the Repository Explorer view and click **New** > **Folder**.

   The New Folder window opens.
2. Enter a folder name and click **OK**.

The new folder is displayed in the Repository Explorer.

Directory Explorer

Use the Directory Explorer to connect to an Optim directory. Only one Optim directory can be associated with the repository.

Use the Directory Explorer to transform process requests into data management services.

The following definitions are available.

Icon	Definition
	access definition
	archive request
	column
	column map
	convert request
	creator ID
	DB alias
	delete request
	extract request
	insert request
	load request
	primary key
	relationship
	restore request

Icon	Definition
	table
	table map
	variable

Using Optim Manager from Optim Designer

Use Optim Manager from Optim Designer to test data management services and Optim interoperability services. Using Optim Manager from Optim Designer is also known as Optim Manager in embedded mode.

Optim Manager is a web application that you can use to test services. Optim Manager is displayed either in the internal browser provided by Optim Designer or in an external browser. You can select the browser that you want Optim Designer to use for Optim Manager by clicking **Window** > **Preferences** > **General** > **Web Browser** in Optim Designer.

Opening Optim Manager from Optim Designer

You can open Optim Manager by testing a service. You can also open Optim Manager by entering the following URL in a web browser: `http://localhost:`*`portnumber`*`/console`, where *`portnumber`* is the port number assigned to Optim Manager. The default port number is 60000.

If the browser displays a message about a page not found when you open Optim Manager, a port conflict may exist and you must change the Optim Manager port number.

You can also open Optim Manager from the Optim Designer toolbar.

Changing the Optim Manager port number

To change the port number, you must edit the following property in the `eclipse.ini` file located in the Optim Designer installation directory:

`-Dorg.eclipse.equinox.http.jetty.http.port=`*`portnumber`*

where *`portnumber`* is the new Optim Manager port number. If Optim Designer is open, you must restart the application to apply the new port number.

Roadmap for configuring Optim Designer

To configure Optim Designer, you must configure IBM InfoSphere Optim for the Optim repository and connect to the repository.

1. Configure IBM InfoSphere Optim for the Optim repository.
2. Connect to the Optim repository.
3. Create a Repository Explorer folder, which contains services, access definitions, column maps, and table maps.

Roadmap for extracting data

To extract data, use an access definition to define the data to an extract, and create an extract service to extract the data.

1. Define a data store alias for the database that contains the data to extract, as described in “Defining a data store alias” on page 21.

2. Create an access definition that defines the data to extract, as described in “Creating an access definition” on page 24.
3. Edit the access definition to define selection criteria and other data selection options, as described in the following topics:
   “Managing tables in an access definition” on page 24
   “Managing selection criteria” on page 25
   “Managing relationships in an access definition” on page 27
   “Managing variables in an access definition” on page 28
   “Managing point and shoot lists” on page 29
4. Create an extract service, as described in “Creating an extract service” on page 43.
5. Edit the extract service, as described in “Editing an extract service” on page 43.
6. Test the extract service, as described in “Testing a data management service” on page 52.

Roadmap for masking data

To mask data, map the source and target data, apply a masking function or procedure, and create a convert service to transform the data.

1. Create a table map to map the source tables to the target tables that will contain the masked data, as described in “Creating a table map” on page 37.
2. Edit the table map to define the target data, as described in “Editing a table map” on page 37.
3. Create a column map to map the source columns to the target columns that will contain the masked data, as described in “Creating a column map” on page 40.
4. Apply a data masking function or a column map procedure, as described in the following topics:
   “Applying a data masking function” on page 40
   “Working with column map procedures” on page 41
5. Create a convert service, as described in “Creating a convert service” on page 46.
6. Edit the convert service, as described in “Editing a convert service” on page 46.
7. Test the convert service, as described in “Testing a data management service” on page 52.

Sample data

Optim provides sample data for data masking as well as predefined source and target data sources.

Sample database tables and structure

As a group, the database tables include information on customers and orders, with shipping instructions. The sample tables also include information about sales and inventory. Minor differences in data types exist, depending upon the DBMS you use to install the sample tables. The following diagram shows the basic structure of the sample database.

As a group, the database tables include information on customers and orders, with shipping instructions. The sample tables also include information about sales and inventory. Minor differences in data types exist, depending upon the DBMS you use to install the sample tables.

An additional set of tables is also installed with the sample database. Tables in the additional set have the same names as tables in the first set, with the suffix “2” appended. The four tables in this additional set are:

- OPTIM_CUSTOMERS2
- OPTIM_ORDERS2
- OPTIM_DETAILS2
- OPTIM_ITEMS2

Tables in the additional set do not contain data. They are used to demonstrate the facilities in Optim.

OPTIM_SALES table:

The OPTIM_SALES table identifies each salesperson by name, ID number and manager.

The OPTIM_SALES table has the following columns:

SALESMAN_ID
CHAR; up to 6 characters; cannot contain null.

FIRST_NAME
VARCHAR; up to 15 characters; cannot contain null.

LAST_NAME
VARCHAR; up to 15 characters; cannot contain null.

NATIONALITY
VARCHAR; up to 30 characters

NATIONAL_ID
VARCHAR; up to 30 characters

PHONE_NUMBER
VARCHAR; up to 20 characters; cannot contain a null value.

EMAIL_ADDRESS
VARCHAR; up to 70 characters; cannot contain null.

AGE SMALLINT; cannot contain null; has a default value.

SEX CHAR; 1 character; cannot contain null; has a default value.

TERRITORY
VARCHAR; up to 14 characters; cannot contain null.

MANAGER_ID
VARCHAR; up to 6 characters.

Primary keys

The SALESMAN_ID column is the primary key column.

Relationships to other tables

The OPTIM_SALES table is a parent of:

- The OPTIM_CUSTOMERS table, through a foreign key on column SALESMAN_ID.
- The OPTIM_MALE_RATES table, through an Optim data-driven relationship on column AGE when SEX = 'M'.
- The OPTIM_FEMALE_RATES table, through an Optim data-driven relationship on column AGE when SEX = 'F'.
- The OPTIM_STATE_LOOKUP table, through an Optim substring relationship using SUBSTR(SALESMAN_ID,1,2).

OPTIM_CUSTOMERS table:

The OPTIM_CUSTOMERS table contains customer names, ID numbers, and addresses.

The OPTIM_CUSTOMERS table has the following columns:

CUST_ID
CHAR; up to 5 characters; cannot contain null; contains a check constraint.

CUSTNAME
CHAR; up to 20 characters; cannot contain null.

ADDRESS1
VARCHAR; up to 100 characters; cannot contain null.

ADDRESS2
VARCHAR; up to 100 characters; cannot contain null.

LOCALITY
VARCHAR; up to 56 characters

CITY VARCHAR; up to 60 characters

STATE
VARCHAR; up to 30 characters

COUNTRY_CODE
CHAR; up to 2 characters

POSTAL_CODE
VARCHAR; up to 15 characters

POSTAL_CODE_PLUS4
CHAR; up to 4 characters; can contain a null value.

EMAIL_ADDRESS
VARCHAR; up to 70 characters

PHONE_NUMBER
VARCHAR; up to 20 characters

YTD_SALES
DECIMAL; dollar amount; cannot contain null; has a default value.

SALESMAN_ID
CHAR; up to 6 characters

NATIONALITY
VARCHAR; up to 30 characters

NATIONAL_ID
VARCHAR; up to 30 characters

CREDITCARD_NUMBER
VARCHAR; 19 characters

CREDITCARD_TYPE
VARCHAR; up to 30 characters

CREDITCARD_EXP
CHAR; 4 characters

CREDITCARD_CVV
VARCHAR; up to 4 characters

DRIVER_LICENSE
VARCHAR; up to 30 characters

CUSTOMER_INFO
XMLTYPE

Primary keys

The CUST_ID column is the primary key column.

Relationships to other tables

The OPTIM_CUSTOMERS table is a parent of:

- The OPTIM_ORDERS table, through a foreign key on column CUST_ID.
- The OPTIM_SHIP_TO table, through an Optim relationship on column CUST_ID.

The OPTIM_CUSTOMERS table is a child of:

- The OPTIM_SALES table, through its foreign key on column SALESMAN_ID.

OPTIM_ORDERS table:

The OPTIM_ORDERS table contains information for orders, including order number, customer ID, and salesman.

The OPTIM_ORDERS table has the following columns:

ORDER_ID
DECIMAL; order ID number; cannot contain null.

CUST_ID
CHAR; customer ID number; up to 5 characters; cannot contain null.

ORDER_DATE
TIMESTAMP; date of order; cannot contain null; has default value.

ORDER_TIME
TIMESTAMP; time of day; cannot contain null; has default value.

FREIGHT_CHARGES
DECIMAL; dollar amount

ORDER_SALESMAN
CHAR; up to 6 characters

ORDER_POSTED_DATE
TIMESTAMP; cannot contain null; has default value.

ORDER_SHIP_DATE
CHAR; date when order is shipped; up to 8 characters; cannot contain null; has default value.

Primary keys

The ORDER_ID column is the primary key column.

Relationships to other tables

The OPTIM_ORDERS table is a parent of the OPTIM_DETAILS table, through a foreign key on column ORDER_ID.

The OPTIM_ORDERS table is a child of the OPTIM_CUSTOMERS table, through its foreign key on column CUST_ID.

OPTIM_DETAILS table:

The OPTIM_DETAILS table contains additional information for each order in the OPTIM_ORDERS table.

The OPTIM_DETAILS table has the following columns:

ORDER_ID
DECIMAL; order ID number; cannot contain null.

ITEM_ID
CHAR; up to 5 characters; item ID number; cannot contain null.

ITEM_QUANTITY
DECIMAL; number of items; cannot contain null.

DETAIL_UNIT_PRICE
DECIMAL; unit price; dollar amount; cannot contain null.

Primary keys

The ORDER_ID and ITEM_ID columns are the primary key.

Relationships to other tables

The OPTIM_DETAILS table is a child of:

- The OPTIM_ORDERS table, through its foreign key on column ORDER_ID.
- The OPTIM_ITEMS table, through its foreign key on column ITEM_ID.

OPTIM_ITEMS table:

The OPTIM_ITEMS table contains information about each item for an order, including description, price, and quantity in inventory.

The OPTIM_ITEMS table has the following columns:

ITEM_ID
CHAR; up to 5 characters; cannot contain null.

ITEM_DESCRIPTION
VARCHAR; up to 72 characters; cannot contain null.

CATEGORY
VARCHAR; up to 14 characters; cannot contain null.

RATING
CHAR; up to 4 characters; cannot contain null.

UNIT_PRICE
DECIMAL; dollar amount; cannot contain null.

ON_HAND_INVENTORY
INTEGER; cannot contain null.

Primary keys

The ITEM_ID column is the primary key column.

Relationships to other tables

The OPTIM_ITEMS table is a parent of the OPTIM_DETAILS table, through a foreign key on column ITEM_ID.

OPTIM_SHIP_TO table:

The OPTIM_SHIP_TO table contains order shipping information.

The OPTIM_SHIP_TO table has the following columns:

CUST_ID
CHAR; up to 5 characters; cannot contain null.

SHIP_ID
DECIMAL; cannot contain null.

ADDRESS1
VARCHAR; up to 100 characters

ADDRESS2
VARCHAR; up to 100 characters

LOCALITY
VARCHAR; up to 56 characters

CITY VARCHAR; up to 30 characters

STATE
VARCHAR; up to 30 characters

COUNTRY_CODE
CHAR; 2 character abbreviation

POSTAL_CODE
VARCHAR; up to 15 characters

POSTAL_CODE_PLUS4
CHAR; 4 characters

IN_CARE_OF
VARCHAR; up to 31 characters

SHIPPING_CHANGE_DT
TIMESTAMP; cannot contains nulls; has default value.

Primary keys

The SHIP_ID column is the primary key column.

Relationships to other tables

The OPTIM_SHIP_TO table is a parent of the OPTIM_SHIP_INSTR table, through an Optim relationship on column SHIP_ID.

The OPTIM_SHIP_TO table is a child of the OPTIM_CUSTOMERS table, through its Optim relationship on column CUST_ID.

OPTIM_SHIP_INSTR table:

The OPTIM_SHIP_INSTR table contains detailed information for order shipping.

The OPTIM_SHIP_INSTR table has the following columns:

SHIP_ID
DECIMAL

SHIP_INSTR_ID
INTEGER

ORDER_SHIP_INSTR
VARCHAR; up to 254 characters

SHIP_UPDATED
TIMESTAMP; cannot contain null; has default value.

Primary keys

The SHIP_INSTR_ID column is the primary key column.

Relationships to other tables

The OPTIM_SHIP_INSTR table is a child of the OPTIM_SHIP_TO table, through its Optim relationship on column SHIP_ID.

OPTIM_MALE_RATES table:

The OPTIM_MALE_RATES table contains insurance rates, based on age.

The OPTIM_MALE_RATES table has the following columns:

AGE SMALLINT

RATE_PER_1000
DECIMAL; rate in dollar amount

Primary keys

The RATE_PER_1000 column is the primary key column.

Relationships to other tables

The OPTIM_MALE_RATES table is a child of the OPTIM_SALES table, through its Optim data-driven relationship on column AGE.

OPTIM_FEMALE_RATES table:

The OPTIM_FEMALE_RATES table contains insurance rates based on age.

The OPTIM_FEMALE_RATES table has the following columns:

AGE SMALLINT

RATE_PER_1000
DECIMAL; rate in dollar amount

Primary keys

The RATE_PER_1000 column is the primary key column.

Relationships to other tables

The OPTIM_FEMALE_RATES table is a child of the OPTIM_SALES table, through its Optim data-driven relationship on column AGE.

OPTIM_STATE_LOOKUP table:

The OPTIM_STATE_LOOKUP table contains state codes and corresponding abbreviations.

The OPTIM_STATE_LOOKUP table has the following columns:

DIST_CODE
CHAR; 3 characters; cannot contain a null value.

DISTRICT
CHAR; 2 characters; cannot contain a null value.

Primary keys

The OPTIM_STATE_LOOKUP table does not have a primary key.

Relationships to other tables

The OPTIM_STATE_LOOKUP table is a child of the OPTIM_SALES table through a substring relationship on column DISTRICT using SUBSTR(SALESMAN_ID,1,2).

Data privacy tables

Data privacy tables are available to clients who have an Optim Data Privacy License. Use the tables to mask company and personal data such as employee names, customer names, social security numbers, credit card numbers, and email addresses. Use the tables to generate transformed data that is both unique and valid within the context of the application.

You can use the data privacy tables to:

- Prevent internal privacy breaches by de-identifying or masking the data available to developers, quality assurance testers, and other personnel.
- Improve privacy compliance initiatives by substituting customer data with contextually accurate, but fictionalized data.
- Protect confidential customer information and employee data in your application development and testing environments.
- Ensure valid test results by propagating masked elements across related tables to ensure the referential integrity of the database.

Each category of personal data is provided in a separate table for the following countries (abbreviations are in parentheses): Australia (AU), Canada (CA), France (FR), Germany (DE), Italy (IT), Japan (JP), Spain (ES), United Kingdom (UK), and United States (US). Each table includes a column of sequential numbers that is used with lookup policies that use hash values to select a row in the lookup table.

Each table name is composed of a country abbreviation prefix and the category (*countryabbreviation_category*). For example, the address table for Canada is named CA_ADDRESSES and the address table for Germany is named DE_ADDRESSES.

The schema includes the following categories:

ADDRESSES
Tables that include columns for street address, city, locality (e.g., state or province), and postal code.

FIRSTNAME
Tables that include a column with male and female given names.

FIRSTNAME_F
Tables that include a column with female given names.

FIRSTNAME_M
Tables that include a column with male given names.

LASTNAME
Tables that include a column with family names.

PERSON
Tables that include columns for birth date, given name, family name, gender, phone number, national ID number, company name, and email address.

CCN Tables that include a credit card numbers for the associated issuer (MasterCard, VISA, etc.).

DOMAIN_NAMES
Table that includes domain names for masking email addresses.

Creating sample data tables

Use the Configuration utility to create sample data tables.

To create sample data tables:

1. From Optim Designer, click **Utilities** > **Configure** The Configuration utility opens.
2. Click **Tasks** > **Load/Drop Sample Data** The Load/Drop Sample Data wizard opens.
3. Complete the steps of the wizard.

   You must select an Optim directory. The default Optim directory name for the Optim repository is PODREPO.

   When prompted to select a DB alias, select a data store alias name.

Creating data privacy tables

Use the Configuration utility to create data privacy tables.

To create data privacy tables:

1. From Optim Designer, click **Utilities** > **Configure** The Configuration utility opens.
2. Click **Tasks** > **Load/Drop Data Privacy Data** The Load/Drop Data Privacy Data wizard opens.
3. Complete the steps of the wizard.

   You must select an Optim directory. The default Optim directory name for the Optim repository is PODREPO.

   When prompted to select a DB alias, select a data store alias name.

Database support

Optim Designer provides support for multiple database management systems.

Optim Designer supports the following databases:

- DB2® for z/OS® V8.1, V9.1, V10.1
- DB2 for Linux, UNIX, and Windows V8.2, V9.1, V9.5, V9.7
- DB2 for i V5.4
- Informix® V10
- Microsoft SQL Server 2005, 2008
- Oracle V10.2, V11, V11.2
- Sybase V12.5, V15, V15.5
- Teradata V2.6, V12, V13
- IBM Informix V11.5, V.11.7

DB2 Prerequisites
To allow Optim to obtain full JDBC metadata from an instance of DB2 z/OS, the DESCSTAT value in ZPARMS must be set to YES. Also, you must run job DSNTIJMS to install the stored procedures needed by JDBC, bind the necessary packages, and set security permissions. In addition, workload manager (WLM) definitions are needed to ensure that the WLM can start the stored procedure address space when requested by DB2.

Accessibility Features

Accessibility features help people with a physical disability, such as restricted mobility or limited vision, or with other special needs, to use software products successfully.

Optim Designer uses accessibility features available with the Eclipse environment.

Accessibility features help people with a physical disability, such as restricted mobility or limited vision, or those with special needs to use software products successfully.

Chapter 2. Configuring IBM InfoSphere Optim for the Optim repository

Use the Configuration utility and system files to configure IBM InfoSphere Optim for the Optim repository.

To configure IBM InfoSphere Optim for the Optim repository, complete the following steps:

1. Edit system files
2. Sign the Optim exit
3. Enter the product license key
4. Access the repository Optim directory
5. Configure options

Editing system files

You must edit the Windows `hosts` and `services` files with connection information for the Optim repository.

1. Use a text editor to open the `hosts` file, located in the following default Windows path: `C:\Windows\system32\drivers\etc\hosts`.
2. Enter the following information: `IP_address optimrepository`, where `IP_address` is the repository IP address, and `optimrepository` is the repository host name. The default host name is `optimrepository`.

   For example: `192.0.2.0 optimrepository`.
3. Save the `hosts` file.
4. Use a text editor to open the `services` file, located in the following default Windows path: `C:\Windows\system32\drivers\etc\services`.
5. Enter the following information: `optimrepo port/tcp`, where `optimrepo` is the repository server name (`optimrepo` is the default), and `port/tcp` is the repository port number and protocol (`9088/tcp` is the default).

   For example: `optimrepo 9088/tcp`.
6. Save the `services` file.

Setting the IBM InfoSphere Optim location

By default, the Optim Designer configuration includes the default path of the IBM InfoSphere Optim`BIN` directory, `C:\IBM\InfoSphere\Optim\RT\BIN`. If IBM InfoSphere Optim is installed in a different path, use the Optim Designer preferences to enter the path.

To enter the IBM InfoSphere Optim `BIN` directory location in the Optim Designer preferences:

1. From Optim Designer, click **Window** > **Preferences**. The Preferences window opens.
2. In the navigation tree, expand the **Optim** node and then click **Optim Distributed**. The Optim Distributed editor opens.
3. In the **Command-line directory** field, enter the path to the IBM InfoSphere Optim `BIN` directory or click **Browse** to select the path.
4. Click **OK**.

Signing an Optim exit

Use the Sign Optim Exit window in the Configuration utility to sign an Optim exit, which validates that a user is authorized to use Optim. The Optim exit must be signed.

A default Optim exit, which allows access to all actions by all users, is supplied with Optim and can be used if your company security policies allow. To sign the default Optim exit, you must enter the unique password that was included when you received your company ID and license key.

For information about creating a user supplied exit, refer to the *IBM InfoSphere Optim Installation and Configuration Guide*.

The Sign Optim Exit window opens automatically when the Configuration utility is first opened.

To open the Configuration utility from Optim Designer, click **Utilities** > **Configure**.

To open the Sign Optim Exit window, click **Options** > **Sign Optim Exit** from the Configuration utility.

Entering the product license key

Use the Specify Product License Key window in the Configuration utility to enter the product license key.

The product license key provides an internal control that determines the features and number of users your company is licensed to use Optim. This key may be changed from time to time when you upgrade the product.

To enter the license key, you can copy it from the email sent to you by IBM.

The Specify Product License Key window opens automatically the first time the Configuration utility is used.

To open the Configuration utility from Optim Designer, click **Utilities** > **Configure**.

To open the Specify Product License Key window, click **Options** > **License** from the Configuration utility.

Accessing the repository Optim directory

Use the Access Optim Directory wizard in the Configuration utility to allow IBM InfoSphere Optim to access the Optim repository.

When you first open Optim Designer, a prompt will indicate that you must establish access to the Optim repository and the Configuration utility will automatically open. Use the Configuration utility to open the Access Optim Directory wizard.

The default Optim directory name for the Optim repository is PODREPO.

To access the Optim repository:

1. From Optim Designer, click **Utilities** > **Configure**. The Configuration utility opens.
2. Click **Tasks** > **Access Optim Directory** The Access Optim Directory wizard opens.
3. Complete the steps of the wizard.

   Edit the connection string and user credentials.

   The format of the connection string is *SERVER*|*PROTOCOL*|*HOST*|*PORT*. The default connection string is `optimrepo|olsoctcp|optimrepository|9088`.

   a. Click **Yes** when prompted to verify the Optim directory name.

4. Click **File** > **Exit** to close the Configuration utility.

Configuring options

Use the Configure Options wizard in the Configuration utility to create a product configuration file and specify working directories.

A product configuration file is required for each installation of IBM InfoSphere Optim.

The Temporary Work Directory stores internal work files and trace files.

The Data Directory stores target data files associated with file data stores.

To configure options:

1. From Optim Designer, click **Utilities** > **Configure**. The Configuration utility opens.
2. Click **Tasks** > **Configure Options** The Configure Options wizard opens.
3. On the Specify Optim Directory page, select **Use Existing Optim Directory and Registry Entry**, and from **Name**, select the Optim directory name for the Optim repository. The default Optim directory name for the Optim repository is PODREPO. Click **Proceed**.
4. On the Initialize Security page, ensure that **Initialize** is not selected. Click **Skip**.
5. On the Enable/Disable Optim Server Feature page, click **Skip**.
6. On the Enable/Disable Archive ODBC Feature page, click **Skip**.
7. On the Specify Product Configuration File page, do the following steps:
   a. Select **Create New File**.
   b. Enter a **File Name** for the configuration file.
   c. In the **Password** field, enter the password that was included when you received your company ID and license key.
   d. Click **Proceed**.
8. On the Modify Product Options page, click **Proceed**.
9. On the Modify Personal Options page, click **Personal Options**. The Personal Options window opens.
10. Select the **General** tab and do the following steps:
    a. Enter a **Temporary Work Directory**.
    b. Enter a **Data Directory**.
    c. Click **OK** to return to the Modify Personal Options page.
    d. Click **Proceed**.
11. On the Complete page, click **Close**.
12. Click **File** > **Exit** to close the Configuration utility.

Chapter 3. Managing data source connections

Use the Repository Explorer to create data store aliases. Use the Directory Explorer to connect to an Optim directory.

Working with the Optim repository

The Optim repository contains the definitions, such as data store aliases, services, and access definitions, that are shared among the Optim components.

Only one Optim directory can be associated with the repository.

Connecting to the Optim repository

Use the New Repository Connection window to define a connection to the Optim repository.

The Optim repository must be running to create a connection.

To define a connection to the Optim repository:

1. In the Repository Explorer, right-click **Optim Repository** and click **New** > **Repository Connection**. The New Repository Connection window opens.
2. Enter connection information and the user credentials for the repository.
3. Click **Test Connection** to verify the connection.
4. Click **OK**.

When Optim Designer connects to the repository, the Repository Explorer displays the contents of the repository.

Editing a repository connection

Use the Edit Repository Connection window to define a connection to the Optim repository.

To define a connection to the Optim repository:

1. In the Repository Explorer, right-click **Optim Repository** and click **Open**. The Edit Repository Connection window opens.
2. Enter connection information and the user credentials for the repository.
3. Click **Test Connection** to verify the connection.
4. Click **OK**.

When Optim Designer connects to the repository, the Repository Explorer displays the contents of the repository.

Changing a repository connection

Use the Switch Repository window to connect to a different Optim repository.

The Optim repository must be running to change a connection.

To change a connection to the Optim repository:

1. In the Repository Explorer, right-click **Optim Repository** and click **Switch Repository**. The Switch repository window opens.
2. Select a repository connection.

3. Click **OK**.

When Optim Designer connects to the repository, the Repository Explorer displays the contents of the repository.

Associating an Optim Directory with the Repository

Use the Associate Directory window to associate an Optim directory with the repository.

An Optim directory connection must be defined in the registry of the Optim Designer computer. Use the Optim Configuration program to define a connection in the registry.

After an initial Optim directory connection, any additional connections may include data store aliases and other definitions that may conflict with the existing Optim directory connection. If you transform requests from the new connection, any existing data store aliases and definitions in the repository will not be overwritten. You will need to edit objects in conflict during the transform process from the new connection.

To associate an Optim directory with the repository:

1. In the Directory Explorer, click **Click to add an Optim directory connection**. The Associate Directory window opens.
2. Select an Optim directory.
3. Click **Finish**.

When Optim Designer connects to the Optim directory, the Directory Explorer displays the contents of the Optim directory.

Transforming a request into a service

Use the Transform Request to Optim Service wizard to transform a request in an Optim directory into a data management service.

An Optim directory must be associated with the repository.

To transform a request in an Optim directory into a data management service:

1. In the Directory Explorer, click **Transform**. The Transform Request to Optim Service wizard opens.
2. On the DB Alias and Database Connections page, review the connection information. If a data store alias is not defined for a DB alias associated with the request, click **Edit Connection** to define a data store alias. Click **Next**.
3. If the transform is successful, click **Finish**. If the transform is not successful, click **Next** to review error information.

A folder with the name of the Optim directory that contained the request will be created in the Repository Explorer. The transformed request will be listed as a service under the folder.

Working with a data store alias

A data store alias is a user-defined object associated with a database. When you define a data store alias, you provide parameters that Optim uses to communicate with that database. These parameters include the type and version of the database management system (DBMS) and the properties for both a client connection and a JDBC connection.

A data store alias name serves as a high-level qualifier that allows you to access a specific database to perform requested functions. For example, in an access definition, you must qualify the name of a table with a data store alias name. The referenced data store alias supplies the parameters needed to connect to the database in which the table resides.

A data store alias identifies a specific database and serves as a prefix in the fully qualified names of primary keys, database tables, and relationships. Data store aliases are essential elements in managing your databases. The following rules apply to data store alias names:

- Each database can have only one data store alias.
- Each data store alias name must be unique.
- Objects in the repository cannot have the same name as a data store alias.

Defining a data store alias

Use the New Data Store Alias wizard to define a new data store alias.

To define a data store alias, you need both the connection string for a client connection and the .jar file for a JDBC connection.

To define a data store alias:

1. In the Repository Explorer, right-click **Data Store Aliases** and click **New Data Store Alias**. The New Data Store Alias wizard opens.
2. Complete the steps of the wizard.

   You must enter properties for both a client connection and a JDBC connection. You must also identify the character set used by the database and how to manage packages for accessing Optim objects.

Editing a data store alias

Use the Connection Properties wizard to edit a data store alias.

To edit a data store alias:

1. In the Repository Explorer, expand the **Data Store Aliases** node, right-click the data store alias and click **Open**. The Connection Properties wizard opens.
2. Complete the steps of the wizard.

Connecting to a data store alias

Use the Repository Explorer to connect to a data store alias.

To connect to a data store alias:

In the Repository Explorer, expand the **Data Store Aliases** node, right-click the data store alias and click **Connect**.

Chapter 4. Managing data models

You can create data models that define data to extract and map target data for data management services. You can also create Optim relationships and primary keys to supplement relationships and primary keys in the database.

Working with access definitions

Use access definitions to specify the tables, relationship traversal, and selection criteria for the data you want to process.

An access definition includes the following items.

Tables An access definition must reference at least one table, view, alias, or synonym. The table, view, alias, or synonym from which rows are selected first is called the start table. You can enter the name of a start table and easily include the names of all tables related to the start table (to a maximum of 24,000 tables).

Start Table
The start table is the first table to use when extracting data. You can specify any table in the access definition as the start table, except a reference table. If you do not explicitly specify a start table, the first table in the table list is the start table.

Reference Table
Tables from which all rows are selected, unless selection criteria are specified for the reference table. Specify any table as a reference table, except the start table.

Relationships
Relationships determine the traversal path for selecting data from tables. By default, relationships are traversed from parent to child, but you can control the direction of traversal using settings in the access definition. Relationships among tables referenced by the access definition are listed on the relationship tab (to a maximum of 24,000 tables). You can select relationships to be used in processing and the direction in which they are traversed.

Selection Criteria
Selection criteria define a specific set of data to use from the tables in an access definition. You can specify SQL operators and values, and use substitution variables with default values.

Point and Shoot
Use a point and shoot list to select specific rows from a start table that will be included in a service.

Variables
Variables are user-defined default values specified in an access definition. You can use these substitution variables to specify column selection criteria or to create an SQL WHERE clause.

Additional Parameters
Use additional parameters for extracting rows that correspond to a particular column value in the start table or for using a specified sampling rate (every nth row).

Naming conventions

A fully qualified access definition name is in the form *identifier.name*, where:

identifier
Qualifier assigned to the access definition (1 to 8 characters).

name Base name assigned to the access definition (1 to 12 characters).

A logical set of naming conventions can identify the use for each access definition and be used to organize them for easy access.

Creating an access definition

Use the New Access Definition wizard to create an access definition.

Before creating an access definition, a data store alias must exist for the database that contains the start table.

To create an access definition:

1. Expand a folder in the Repository Explorer to contain the access definition, right-click **Access Definitions** and click **New Access Definition**. The New Access Definition wizard opens.
2. Complete the steps of the wizard.

   You must select a data store alias and a start table from the alias. You can choose to add tables related to the start table.

The new access definition is displayed in the Access Definition Editor.

Managing tables in an access definition

Use the **Tables** tab in the Access Definition editor to add or remove tables in an access definition. You can also change tables to related or reference tables.

Adding tables to an access definition

Use the Add Tables wizard to add tables in a data store alias to an access definition.

To add tables to an access definition:

1. Expand a folder in the Repository Explorer that contains the access definition, expand the **Access Definitions** node, and double-click the access definition. The Access Definition Editor opens.
2. Select the **Tables** tab.
3. Click **Add Tables**. The Add Tables wizard opens.
4. Complete the steps of the wizard.

   You must select a data store alias that contains the tables to add. You can add reference tables or tables related to a selected table.
5. Save the access definition.

Changing tables to reference or related

Use the Access Definition Editor to change table in an access definition to related or reference tables.

To change tables to reference or related:

1. Expand the folder in the Repository Explorer that contains the access definition, expand the **Access Definitions** node, and double-click the access definition. The Access Definition Editor opens.
2. Select the **Tables** tab.
3. Select a table to change to related or reference.
4. Click **Change to Reference** or **Change to Related**. The **Type** column will show the new table type.
5. Save the access definition.

Removing tables from an access definition

Use the Access Definition Editor to remove tables from an access definition.

To remove tables from an access definition:

1. Expand the folder in the Repository Explorer that contains the access definition, expand the **Access Definitions** node, and double-click the access definition. The Access Definition Editor opens.
2. Select the **Tables** tab.
3. Select a table to remove.
4. Click **Remove Table**.
5. Save the access definition.

Viewing and editing traversal steps

Use the Traversal Steps window to view and edit the traversal steps of an access definition.

You can evaluate the tables, relationships, and the steps performed to select data. This evaluation can ensure that you retrieve the desired set of data.

The steps in a process may be repeated any number of times according to relationships you select and criteria you specify.

Any table may be revisited several times in successive steps. Cycles may also be involved. A cycle causes a set of tables to be traversed repeatedly until a complete pass through the cycle does not result in selecting additional rows.

When a relationship is traversed to select parent rows for child rows already selected, any selection criteria for the parent table are ignored.

To view and edit traversal steps:

1. Expand a folder in the Repository Explorer that contains the access definition, expand the **Access Definitions** node, and double-click the access definition. The Access Definition Editor opens.
2. Select the **Tables** tab.
3. Click **Show Steps**. The Traversal Steps window opens.
4. Review the steps and edit the order in which tables are selected.
5. Click **OK**.
6. Save the access definition.

Managing selection criteria

Use selection criteria to focus on a specific set of related data by defining an SQL WHERE clause and using substitution variables with default values.

Selection criteria must conform to SQL syntax and include relational or logical operators. Logical operators and syntax vary among DBMSs. Refer to the appropriate DBMS documentation for information.

To select the desired set of data for a table, you may need a combination of `AND` and `OR` logical operators.

If a Point and Shoot list is also used, it is logically included with the other criteria with an `OR` logical operator.

When a relationship is traversed from child to parent, any selection criteria for the parent table are ignored.

Date criteria

A unique operator, **`BEFORE`**, allows you to select data on the basis of values in a DATE column. The syntax for this operator is:

`BEFORE` (*n*`D` *n*`W` *n*`M` *n*`Y`)

Use the **D**, **W**, **M**, and **Y** arguments in any combination to indicate the number of days, weeks, months, or years subtracted from the date at run time. If no arguments are specified, the current date is used. Rows with a date older than the calculated date are extracted or archived. The *n* multiplier is an integer and can optionally be preceded by + or -.

Defining selection criteria

Use the Table Specification window to define selection criteria with an SQL WHERE clause.

To define selection criteria:

1. Expand a folder in the Repository Explorer that contains the access definition, expand the **Access Definitions** node, and double-click the access definition. The Access Definition Editor opens.
2. Select the **Tables** tab.
3. Select the table to which you will add selection criteria.
4. Click **Add Selection Criteria**. The Table Specification window opens.
5. Select the **Selection Criteria** tab. The tab displays a WHERE clause for the table.
6. Enter selection criteria.

   To display a list of columns and operators, right-click the WHERE clause and click **Content Assist**.

   To add a variable, select a **Variable delimiter** and click **Insert Variable**.

   Click **Check Syntax** to verify the syntax and identify errors.
7. Click **OK**.

   If the syntax is not valid, a prompt will open to identify to error. You cannot save the criteria if it contains errors.
8. Save the access definition.

Specifying data grouping options

Use the **Data group properties** tab in the Access Definition Editor to specify group selection, which extracts a number of rows based on values in a particular column in the start table. Rows in each group have the same value in the selected column.

When criteria are used in combination with group selection parameters, the criteria are applied first with group selection parameters applied to the result.

To specify data grouping options:

1. Expand a folder in the Repository Explorer that contains the access definition, expand the **Access Definitions** node, and double-click the access definition. The Access Definition Editor opens.
2. Select the **Data group properties** tab.
3. Select a column and specify grouping options.
4. Save the access definition.

Specifying data sampling and row limit options

Use the **Tables** tab in the Access Definition Editor to specify options for extracting a sampling of rows or for limiting the number of rows to extract.

If a Point and Shoot list is used to select start table rows, the extract service ignores any data sampling or row limit parameters for the start table.

To specify options for data sampling and row limits:

1. Expand a folder in the Repository Explorer that contains the access definition, expand the **Access Definitions** node, and double-click the access definition. The Access Definition Editor opens.
2. Select the **Tables** tab.
3. Enter the following options:

Every Nth Number of Rows
Enter a numeric value to specify a sampling factor for a table. For example, if you enter 5, the process extracts every 5th row in the table, beginning with the 5th row. Valid values are 1 through 9999.

Row Limit
Enter a numeric value to limit the number of rows extracted from a table. Valid values are 1 through 999999999999.

4. Save the access definition.

Managing relationships in an access definition

Use the **Relationships** tab in the Access Definition Editor to add or remove relationships in an access definition. You can also specify relationship traversal and table access options.

Selecting relationship traversal options

Use the Access Definition Editor to manage relationship traversal options in an access definition.

You can specify the following options:

- The maximum number of rows from the child table to be selected for a relationship.
- The maximum number of key lookups performed at one time for a table.

  Increasing the key lookup limit may significantly improve performance. For example, if you specify 5 as the key lookup limit and the key has a single column, 5 key values are searched in a single request to the DBMS.
- The traversal paths:
  - Traverse the relationship from child to parent, in order to select a parent row for each child row to ensure the relational integrity. (Option 1)
  - Traverse the relationship to select additional child rows for each parent row selected as a result of a traversal from child to parent. (Option 2)

  Options 1 and 2 are relevant when the start table is a child table or when a table has more than one parent table that is referenced by the access definition.

  Option 2 is relevant only if you traverse a relationship from child to parent. For example, if a process traverses from child to parent (option 1) and a parent row is selected, option 2 causes the process to select additional child rows for that parent row.

  If you select option 2 for a relationship, consider a child limit on the number of child rows to extract.

To manage relationship traversal options in an access definition:

1. Expand the folder in the Repository Explorer that contains the access definition, expand the **Access Definitions** node, and double-click the access definition. The Access Definition Editor opens.
2. Select the **Relationships** tab and then the **Traversal Options** tab.
3. Enter traversal options.
4. Save the access definition.

Selecting table access options

Use the Access Definition Editor to select the method for accessing the parent or child table for each relationship.

You can specify the following options:

Default
Optim determines the best method. A key lookup is used when a DBMS index is available, and a scan when an index is not available. However, if accessing a significant portion of the table, the default is to scan, even if an index exists.

Force scan
Read all rows in a table at one time.

Force key lookup
Locate rows using a WHERE clause to search for primary or foreign key values.

Note: Override the default method only if the statistical information in the process report indicates that the default method is less efficient.

To manage table access options:

1. Expand the folder in the Repository Explorer that contains the access definition, expand the **Access Definitions** node, and double-click the access definition. The Access Definition Editor opens.
2. Select the **Relationships** tab and then the **Table Access** tab.
3. Enter table access options.
4. Save the access definition.

Managing variables in an access definition

Variables are user-defined default values specified in an access definition. You can use these substitution variables to create an SQL WHERE Clause.

By assigning variables, you provide values for the variables each time the access definition is processed. As an option, you can provide default values for substitution variables. The variables are saved with the access definition.

Default values

When you create a variable, you can enter an optional default value to be used when no value is specified for the variable at run time.

Default values must be of the appropriate data type and size for the column and must conform to SQL syntax. For example, assume a variable name is **ST** (state), the variable delimiter is a colon (**:**), and the column requires character data.

- If you use the variable with single quotes in the selection criteria, you must specify the value without single quotes:

Selection Criteria	Value
= ':ST'	CA

- If you use the variable without single quotes in the selection criteria, you must specify the value with single quotes:

Selection Criteria	Value
= :ST	'CA'

Note: Default values are not validated until run time. If a value has the incorrect data type or size for the column or does not conform to SQL syntax, processing errors may result.

Prompt string

You must enter the text that prompts for the variable value at run time. Type the prompt string exactly as you want it to appear in the process request dialog (up to 50 characters). This prompt is displayed before you run the process.

Creating a variable

Use the Add a Variable window to create a variable in an access definition.

To create a variable:

1. Expand the folder in the Repository Explorer that contains the access definition, expand the **Access Definitions** node, and double-click the access definition. The Access Definition Editor opens.
2. Select the **Variables** tab.
3. Click **Add**. The Add a Variable window opens.
4. Enter information for the variable. You must enter a variable name and prompt text. Click **OK**.
5. Save the access definition.

Managing point and shoot lists

Use a point and shoot list to select specific rows from a start table that will be included in a service.

A point and shoot list is included in an access definition. When you use point and shoot to select rows from the start table, the primary keys for these rows are stored in a point and shoot file. The service request uses the primary keys to identify the rows to process first.

Creating a point and shoot list file

Use the New Point and Shoot File window to create a point and shoot list file for a table in a data store alias.

To create a point and shoot list file:

1. In the Repository Explorer, expand the **Data Store Aliases** node and the target data store alias until the table to which you will add the point and shoot file is displayed.
2. Expand the table, right-click **Point and Shoot Lists** and click **New Point and Shoot** The New Point and Shoot File window opens.
3. Enter a file name and click **OK**.

The point and shoot list file is displayed under **Point and Shoot Lists**.

You must use the Point and Shoot Editor to select rows or the point and shoot list.

Selecting a point and shoot list

Use the Access Definition Editor to select a point and shoot list for an access definition.

To select a point and shoot list, a point and shoot file must be available for the start table.

To select a point and shoot list:

1. Expand the folder in the Repository Explorer that contains the access definition, expand the **Access Definitions** node, and double-click the access definition. The Access Definition Editor opens.
2. Select the **Point and Shoot** tab.
3. Select a point and shoot list.
4. Save the access definition.

Working with Optim relationships

Optim uses relationships to determine the data to be retrieved from related tables and relies upon relationships defined to the database, when available. However, you can also define Optim relationships to supplement relationships in the database.

With Optim relationships, a number of the database restrictions are relaxed. For example:

- Primary keys and foreign keys are not required.
- Corresponding columns need not be identical, but must be compatible.
- At least one of a pair of corresponding columns must be specified by column name. However, you can use an expression to evaluate or define the value in the second column. Expressions can include string literals, numeric constants, NULL, concatenation, and substrings.

The more flexible Optim relationships are called “extended” relationships. Extended relationships can replicate implicit or application-managed relationships in your database, allowing you to manipulate sets of relational data in the same manner as in your production environment.

In addition, an Optim relationship can be stored in the Optim Directory as:

- An **explicit** relationship, used for a single pair of tables.
- A **generic** relationship, used for one or more pairs of tables that have the same base name, column names, and attributes, but different Creator IDs.

Generic relationships are useful when several sets of tables differ only by Creator ID. (For example, in a test environment, each programmer may use a separate copy of the same production tables. Each set of tables can be distinguished by the Creator ID.) Using generic relationships, you define one set of relationships that applies to all sets of tables. Also, when a set of these tables is added, the generic relationships automatically apply.

Restrictions

Although the rules for creating Optim relationships are more flexible than the rules for creating database-defined relationships, there are some restrictions:

- You must reference at least one column for each table in the relationship.
- You can reference a maximum of 64 columns for any table in the relationship.
- You cannot match a literal or constant to a literal or constant.
- You cannot use a Large Object (LOB) or SQL variant column.
- The total length of all values specified in either the parent table or the child table cannot exceed 3584 bytes.
- You cannot create a relationship using an SQL Variant column.

In a Relationship definition for a multi-byte or Unicode database:

- You cannot use the Substring Function.
- You cannot concatenate character data (CHAR or NCHAR) with binary (RAW).
- If Oracle character semantics are used for any CHAR column, all CHAR columns in the relationship must have character semantics or an NCHAR data type.

EXAMPLES:

Parent	Supported/Not Supported	Child	Description
CHAR		CHAR	Supported, semantics must match
NCHAR		NCHAR	Supported, semantics irrelevant.
CHAR		NCHAR	Not Supported

Parent	Supported/Not Supported	Child	Description
CHAR		VARCHAR	Supported, semantics must match.
NCHAR		NVARCHAR	Supported, semantics irrelevant.
CHAR||NCHAR		NCHAR||CHAR	Supported, if character semantics; not supported if byte semantics.
CHAR||NCHAR		NCHAR||NCHAR	Supported, if character semantics; not supported if byte semantics.

Compatibility rules for relationships

When you define an Optim relationship, the corresponding values must be compatible.

Column Type	Is Compatible With
Character Column	• Character Column • Numeric Column • String Literal • Character Expression
Numeric Column	• Numeric Column • Numeric Constant • Character Column
Binary Column	• Binary Column • Hexadecimal Literal • Binary Expression
Boolean Column	• Boolean Column • Boolean Constant (True or False)
Date Time Column	Date Time Column
Date Column	Date Column
Time Column	Time Column
Interval Column	Interval Column

Note:

- In processing, a value is converted to the data type needed to select related rows. By default, the result of converting a numeric value to a character data type is right-justified with leading zeros. Special registry settings allow you to change the default to left justification with leading or trailing spaces. Also, a character to numeric pairing requires a scale equal to 0 for the numeric column.
- You can use NULL for any null eligible column.
- Unicode or multi-byte columns must be of the same character set.

Data types

The following classes of data and associated data types are supported. These data classes are important for data compatibility when you use column values in relationships.

Character class

DBMS	Data Types
DB2	CHAR, VARCHAR, CLOB
Oracle	CHAR, VARCHAR2, LONG, CLOB, NCLOB, NCHAR, NVARCHAR
Sybase ASE	CHAR, VARCHAR, TXT
SQL Server	CHAR, VARCHAR, TXT
Informix	CHAR, VARCHAR, TXT

Note: Single-byte character columns are not compatible with multi-byte or Unicode character columns.

Numeric class

DBMS	Data Types
DB2	INTEGER, SMALLINT, DECIMAL, FLOAT, DOUBLE
Oracle	NUMBER, FLOAT
Sybase ASE	TINYINT, INT, SMALLINT, DECIMAL, FLOAT, REAL, MONEY, SMALL MONEY
SQL Server	TINYINT, INT, SMALLINT, DECIMAL, FLOAT, REAL, MONEY, SMALL MONEY
Informix	INTEGER, SMALLINT, DECIMAL, FLOAT, REAL, DOUBLE PRECISION, SMALLFLOAT, SERIAL, MONEY, NUMERIC

Binary class

DBMS	Data Types
DB2	CHAR (for Bit Data), VARCHAR (for Bit Data), BLOB
Oracle	RAW, LONG RAW
Sybase ASE	BINARY, VARBINARY, IMAGE
SQL Server	BINARY, VARBINARY, IMAGE
Informix	BYTE

Boolean class

DBMS	Data Types
Sybase ASE	BOOLEAN (TRUE or FALSE)

Datetime

DBMS	Data Types
DB2	TIMESTAMP
Oracle	DATE, TIMESTAMP, TIMESTAMP WITH LOCAL TIME ZONE, TIMESTAMP WITH TIME ZONE
Sybase ASE	DATETIME, SMALL DATE TIME
SQL Server	DATETIME, SMALL DATE TIME
Informix	DATE, DATETIME

Date class

DBMS	Data Types
DB2	DATE
Oracle	DATE
Informix	DATE

Time class

DBMS	Data Types
DB2	TIME

Interval class

DBMS	Data Types
Oracle	YEAR/MONTH INTERVAL, DAY/SECOND INTERVAL
Informix	YEAR/MONTH INTERVAL, DAY/TIME INTERVAL

Creating an Optim Relationship

Use the New Optim Relationship wizard to create a new Optim relationship.

Before creating an Optim relationship, a data store alias must exist for the tables in the relationship.

To create an Optim relationship:

1. In the Repository Explorer, right-click **Optim Relationships** and click **New Optim Relationship**. The New Optim Relationship wizard opens.
2. Complete the steps of the wizard.
   You must select the parent and child tables in the relationship.

The new Optim relationship is displayed in the Relationship Editor.

You must use the Relationship Editor to define a column expression that identifies the parent and child columns in the relationship.

Editing an Optim relationship

Use the Relationship Editor to define a column expression that identifies the parent and child columns in an Optim relationship. You can also use the editor to create a generic relationship and edit columns.

Creating a column expression

Use the Add Column Expression window to create a column expression that identifies the parent and child columns in an Optim relationship.

To create a column expression in an Optim relationship:

1. In the Repository Explorer, expand the **Optim Relationships** node and double-click the relationship to edit. The Relationship Editor opens.
2. Click **Add Column Expression...**. The Add Column Expression window opens.
3. Click **Select Parent Column**. The Select a Parent Column window opens.
4. Select a parent column and click **OK** to return to the Add Column Expression window.
5. Click **Select Child Column**. The Select a Child Column window opens.
6. Select a child column and click **OK** to return to the Add Column Expression window.
7. Click **OK** to return to the editor. The parent and child columns are listed in the editor.

8. Save the relationship.

Editing the columns in a relationship

Use the Select a Parent Column and Select a Child Column windows to edit the columns in an Optim relationship.

To edit the columns in an Optim relationship:

1. In the Repository Explorer, expand the **Optim Relationships** node and double-click the relationship to edit. The Relationship Editor opens.
2. Select the row that contains a column to edit. To edit a parent column:
   a. Click **Select Parent Column**. The Select a Parent Column window opens.
   b. Select a parent column and click **OK** to return to the editor.

   To edit a child column:
   a. Click **Select Child Column**. The Select a Child Column window opens.
   b. Select a child column and click **OK** to return to the editor.

   The selected items are listed in the editor.
3. Save the relationship.

Changing the column order in a relationship

Use the Relationship Editor to change the order of columns in an Optim relationship.

To change the order of columns in an Optim relationship:

1. In the Repository Explorer, expand the **Optim Relationships** node and double-click the relationship to edit. The Relationship Editor opens.
2. Select the row that contains columns to reorder.
3. Click **Move Row Up** or **Move Row Down** to change the order.
4. Save the relationship.

Creating a generic relationship

Use the Relationship Editor to create a generic Optim relationship.

Some databases contain sets of tables that are identical except for the creator ID. Rather than define a relationship for each set of tables, you can define a generic relationship that applies for all sets of tables that have the same base name, regardless of the creator ID. You can modify the base tables in a generic relationship.

To create a generic Optim relationship:

1. In the Repository Explorer, expand the **Optim Relationships** node and double-click the relationship to edit. The Relationship Editor opens.
2. Click **Generic**.
3. Save the relationship.

Working with Optim primary keys

A primary key is the column or columns that contain values that uniquely identify each row in a table. A database table must have a primary key for Optim to insert, update, restore, or delete data from a database table. Optim uses primary keys that are defined to the database. However, you can also define Optim primary keys to supplement the primary keys in the database.

A primary key is needed:

- In any table that is visited more than once in a process, for example, a child table that has two or more parent tables referenced in the access definition.
- To enable the point and shoot feature for a start table.

Note: If a primary key is not defined and is required to perform a specific task, an error message appears.

Types of Optim primary keys

You can define two types of primary keys:

- An **explicit** primary key applies to a single table.
- A **generic** primary key applies to any tables that have the same base name, column names, and attribute specifications, but different creator IDs.

There is no difference in function or appearance between generic and explicit primary keys. However, if a table has keys of both types, the explicit primary key is used.

Naming conventions

The fully qualified name of a primary key is the same as the fully qualified name of the database table for which it is defined. This name consists of: *alias.creatorid.tablename*.

alias Alias that identifies the database where the table resides (1 to 12 characters).

creatorid
Creator ID assigned to the table (1 to 64 characters).

tablename
Base table name (1 to 64 characters).

Note:

- The combined total length of columns for a primary key is limited to 3584 bytes.

Creating a primary key

Use the New Primary Key wizard to create a new primary key for a table.

Before creating a primary key, a data store alias must exist for the table.

To create a primary key:

1. In the Repository Explorer, right-click **Optim Primary Keys** and click **New Primary Key**. The New Primary Key wizard opens.
2. Complete the steps of the wizard.
   You must select a table.

The new primary key is displayed in the Primary Key editor.

You must use the Primary Key Editor to select primary key columns for the table.

Editing a primary key

Use the Primary Key Editor to edit a primary key for a table. You can select key columns and define a generic primary key.

Selecting primary key columns

Use the Primary Key Editor to select primary key columns for a table. You can also set the order of the key columns.

To select primary key columns for a table:

1. In the Repository Explorer, expand the **Optim Primary Keys** node and double-click the relationship to edit. The Primary Key Editor opens.
2. In the **Available columns** list, select the key columns.
3. Click >> to move the columns to the **Key columns** list.

   To set the order of the key columns:

   a. In the **Key columns** list, select a column to move.
   b. Click **Move Up** to move a column up in the order, or **Move Down** to move a column down in the order.
4. Save the primary key.

Creating a generic primary key

Use the Primary Key Editor to create a generic Optim primary key.

Some databases contain sets of tables that are identical except for the creator ID. Rather than define an explicit primary key for each table, define a generic primary key for all tables that have the same base name, regardless of the creator ID.

To create a generic Optim primary key:

1. In the Repository Explorer, expand the **Optim Primary Keys** node and double-click the relationship to edit. The Primary Key Editor opens.
2. Click **Generic**.
3. Save the primary key.

Working with table maps

Use a table map to define specifications for correlating source and destination tables of compatible data. You can map tables that have different names, modify table names, exclude tables from a process, or include column maps for greater control over the data.

Use a table map to:

- Direct the placement of data in a convert, insert, or load service.
- Exclude one or more tables from a convert, insert, or load service.
- Include a column map.

Depending on the process for which you are using a table map, the two sets of tables are referred to as source tables and target tables:

- **Source tables** are extracted tables that contain data to be used in a convert, insert, or load process.
- **Target tables** are the tables into which data is converted, inserted, or loaded.

Note: Matched tables can have different creator IDs or names.

Table maps are stored in the Optim repository or embedded in a data management service.

As an option, you can use a column map for any pair of tables in a table map. Column maps identify and match columns in a pair of tables. A column map must be used when column names or attributes are dissimilar or data transformations are needed.

Naming conventions

The fully qualified name of a table map is in the form: *identifier.name*.

identifier
Identifier assigned to the table map (1 to 8 characters).

name Name assigned to the table map (1 to 12 characters).

It is helpful to use a logical set of naming conventions to identify the use for each and to organize definitions for easy access.

Creating a table map

Use the New Table Map wizard to create a new table map.

Before creating a table map, a file data store alias must exist for the source table.

To create a table map:

1. Expand a folder in the Repository Explorer to contain the table map, right-click **Table Maps** and click **New Table Map**. The New Table Map wizard opens.
2. Complete the steps of the wizard.
   You must select a file data store alias for the source table.

The new table map is displayed in the Table Map Editor.

You must use the editor to define target data stores and schemas before saving the table map.

Editing a table map

Use the Table Map Editor to edit target data and add a column map.

Defining default target data in a table map

Use the Table Map Editor to define default target data stores and schemas in a table map.

Before defining default table map target data, a data store alias must exist for the target data.

To define default target data in a table map:

1. Expand a folder in the Repository Explorer that contains the table map, expand the **Table Maps** node, and double-click the table map. The Table Map Editor opens.
2. Select the **Table map** tab.
3. In the **Data store alias and schema map** area, select the row that contains the source data store.
4. To define a target data store alias, click the **Target Data Store Alias** cell and select the target data store alias from the list.
5. To define a target schema, click the **Target Schema** cell and select the target schema from the list.
6. Save the table map.

Editing target data in a table map

Use the Table Map Editor to edit target data stores, schemas, and tables for source tables in a table map.

Before editing table map target data, a default target data store alias must exist for the target data.

To edit target data in a table map:

1. Expand a folder in the Repository Explorer that contains the table map, expand the **Table Maps** node, and double-click the table map. The Table Map Editor opens.
2. Select the **Table map** tab.
3. In the **Table map** area, select the row that contains the source table.

4. To define a target data store alias, click the **Target Data Store** cell and enter the name of a target data store alias.
5. To define a target schema, click the **Target Schema** cell and select the target schema from the list.
6. To define a target table, click the **Target Table** cell and select the target table from the list.
7. Save the table map.

Adding a column map to a table map

Use the Table Map Editor to add a column map to a pair of mapped tables.

To add a column map to a table map:

1. Expand a folder in the Repository Explorer that contains the table map, expand the **Table Maps** node, and double-click the table map. The Table Map Editor opens.
2. Select the **Table map** tab.
3. In the **Table map** area, select the row that contains the tables for the column map.
4. Click **Add Column Map**. The New Column Map window opens with a list of column maps that contain the selected tables.
5. Select a column map or select **Create a new column map** to create a column map based on the selected tables. Click **OK**.
   a. If you choose to create a new column map, the Column Map Editor opens and is populated with the columns from the selected tables.
   b. Edit and save the new column map.
6. Save the table map.

Working with column maps

A column map provides specifications needed to match or exclude columns from processing in a data management service. Convert, insert, and load services must reference a table map, which may reference one or more column maps. You can use a column map to define data transformations with functions or column map procedures.

A column map must be used when column names or attributes are dissimilar, when data transformations are needed, or when excluding one or more columns from processing. A column map referenced in a convert, insert, or load service can modify data, age dates, or convert currency. You can define data transformations using functions or column map procedures.

When you create a new column map, you must choose a file data store alias as the source for the columns you want to map. Similarly, you must specify a data store alias for the target data.

Column maps stored in the Optim repository are available for reuse or sharing with other users. A local column map is stored as part of a data management service and is otherwise not available for other services. If the associated table map is local to a service, both the table map and column map are available only to the specific service.

Naming conventions

The fully qualified name of a column map has two parts: *identifier.name*.

identifier
: Identifier assigned to the column map (1 to 8 characters).

name
: Name assigned to the column map (1 to 12 characters).

When you create column maps, it is helpful to use a logical set of naming conventions to identify and organize definitions for easy access.

Compatibility rules for column maps

The following classes of data and associated data types are supported. These data classes are important for data compatibility when you specify column values in relationships and column maps.

Character

DBMS	Data Types
DB2	CHAR, VARCHAR, CLOB
Oracle	CHAR, VARCHAR2, LONG, CLOB, NCLOB, NCHAR, NVARCHAR
Sybase ASE	CHAR, VARCHAR, TXT
SQL Server	CHAR, VARCHAR, TXT
Informix	CHAR, VARCHAR, TXT

Note: Single-byte character columns are not compatible with multi-byte or Unicode character columns.

Numeric

DBMS	Data Types
DB2	INTEGER, SMALLINT, DECIMAL, FLOAT, DOUBLE
Oracle	NUMBER, FLOAT
Sybase ASE	TINYINT, INT, SMALLINT, DECIMAL, FLOAT, REAL, MONEY, SMALL MONEY
SQL Server	TINYINT, INT, SMALLINT, DECIMAL, FLOAT, REAL, MONEY, SMALL MONEY
Informix	INTEGER, SMALLINT, DECIMAL, FLOAT, REAL, DOUBLE PRECISION, SMALLFLOAT, SERIAL, MONEY, NUMERIC

Binary

DBMS	Data Types
DB2	CHAR (for Bit Data), VARCHAR (for Bit Data), BLOB
Oracle	RAW, LONG RAW
Sybase ASE	BINARY, VARBINARY, IMAGE
SQL Server	BINARY, VARBINARY, IMAGE
Informix	BYTE

Boolean

DBMS	Data Types
Sybase ASE	BOOLEAN (TRUE or FALSE)

Datetime

DBMS	Data Types
DB2	TIMESTAMP
Oracle	DATE, TIMESTAMP, TIMESTAMP WITH LOCAL TIME ZONE, TIMESTAMP WITH TIME ZONE
Sybase ASE	DATETIME, SMALL DATE TIME
SQL Server	DATETIME, SMALL DATE TIME
Informix	DATE, DATETIME

Date

DBMS	Data Types
DB2	DATE
Oracle	DATE
Informix	DATE

Time

DBMS	Data Types
DB2	TIME

Interval

DBMS	Data Types
Oracle	YEAR/MONTH INTERVAL, DAY/SECOND INTERVAL
Informix	YEAR/MONTH INTERVAL, DAY/TIME INTERVAL

Creating a column map

Use the New Column Map wizard to create a new column map.

Before creating a column map, a file data store alias must exist for the source and target data.

To create a column map:

1. Expand a folder in the Repository Explorer to contain the column map, right-click **Column Maps** and click **New Column Map**. The New Column Map wizard opens.
2. Complete the steps of the wizard.

   You must select a file data store alias and table for the source data. You must also select a data store alias and table for the target data.

The new column map is displayed in the Column Map Editor.

Editing a column map

Use the Column Map Editor to edit a column map and apply data masking functions.

Applying a data masking function

Use the Column Map Editor to apply and edit a function for a column.

To apply a data masking function:

1. Expand a folder in the Repository Explorer that contains the column map, expand the **Column Maps** node, and double-click the column map. The Column Map Editor opens.
2. Select the column for the policy.
3. Click **Apply Function**. The Apply Function window opens.
4. Select the function to apply. Click **OK**. The function name appears in the associated column and the function editor opens in the Column Map Editor.
5. Depending on the function, do one of the following steps:

   Select the function expression tab and edit the function expression.

   Select an option tab and select options for the function.

6. Save the column map.

Mapping a source column

Use the Column Map Editor to map a source column to a target. Source and target columns with the same name and compatible data types are automatically mapped.

To map a source column:

1. Expand a folder in the Repository Explorer that contains the column map, expand the **Column Maps** node, and double-click the column map. The Column Map Editor opens.
2. Click a source column and select a column name from the list.
3. Save the column map.

Working with column map procedures

A column map procedure is a procedure that is used to mask or transform the data in a column when you run a service. As the name indicates, you must add column map procedures to a column map. You can write column map procedures by using the Lua scripting language.

Creating a column map procedure:

Use the Lua script editor to create a column map procedure.

To create a column map procedure:

1. Expand a folder in the Repository Explorer that contains the column map, expand the **Column Maps** node, and double-click the column map. The Column Map Editor opens.
2. Select the column for the column map procedure.
3. Click **Add Procedure**. The Lua script editor opens.
4. Create and save the procedure. The column associated with the procedure displays **Procedure**.
5. Save the column map.

Editing a column map procedure:

Use the Lua script editor to edit a column map procedure.

To edit a column map procedure:

1. Expand a folder in the Repository Explorer that contains the column map, expand the **Column Maps** node, and double-click the column map. The Column Map Editor opens.
2. Select the column associated with the column map procedure.
3. Click **Edit Procedure**. The Lua script editor opens.
4. Edit and save the procedure.
5. Save the column map.

Entering values for parameters in a column map procedure:

Use the Column Map Editor to enter values for parameters in a column map procedure.

To enter values for parameters in a column map procedure:

1. Expand a folder in the Repository Explorer that contains the column map, expand the **Column Maps** node, and double-click the column map. The Column Map Editor opens.
2. Select the column that contains the procedure. The **Procedure** editor opens in the Column Map Editor.
3. In the **Procedure Parameters** field, enter column separated values for the parameters. For example: `abc, def`.
4. Save the column map.

Chapter 5. Designing data management services

Use a data management service to extract, convert, load, or insert data. A service uses access definitions to define the data to extract. A service also uses table maps and column maps to map source and target data. You can mask data by applying a data privacy policy to an entity processed by a service. You can also transform process requests in an Optim directory into data management services. Use Optim Manager in embedded mode to test services.

Working with extract services

Use an extract service to copy a set of related rows from one or more tables and save the rows to a file data store.

The extract service specifies the set of parameters needed to extract data and object definitions from source tables and the file data store in which to store the extracted information.

The extract service always extracts definitions for tables and columns. These definitions are used to create the destination tables, if necessary. You can also choose to extract other object definitions, including primary keys, relationships, and indexes.

Naming conventions

The fully qualified name of an extract service consists of: *identifier.name*.

identifier
: Identifier that serves as the prefix for the extract service name (1 to 8 characters).

name
: Name assigned to the extract service (1 to 12 characters).

When you create extract services, it is helpful to use a logical set of naming conventions to identify the use for each and to organize definitions for easy access.

Creating an extract service

Use the New Service wizard to create an extract service.

An extract service requires an access definition. During the creation of a service, you can select or create an access definition.

To create an extract service:

1. Expand a folder in the Repository Explorer to contain the extract service, right-click **Services**, and click **New Service**. The New Service wizard opens.
2. Complete the steps of the wizard.

   You must select or create an access definition. You must also enter a name for the target file data store.

The new extract service is displayed in the Extract Service Editor.

Editing an extract service

Use the Extract Service Editor to specify extract service processing options, including the data objects to extract, conversion, file compression, and variable overrides.

Specifying extract processing options

Use the **Service Properties** tab in the Extract Service Editor to specify extract service processing options.

You can specify the following extract service processing options:

- Manage the number of database connections. Increasing database connections improves performance when processing large quantities of data by allowing multiple threads to extract rows concurrently.
- Limit the number of extracted rows.
- Allow file attachments.
- Include statistical information in the extract service report.

To specify extract service processing options:

1. Expand a folder in the Repository Explorer that contains the extract service, expand the **Services** node, and double-click the extract service to edit. The Extract Service Editor opens.
2. Select the **Service Properties** tab.
3. Edit the service processing options.
4. Save the extract service.

Specifying objects and grouping options

Use the **Data and Objects** tab in the Extract Service Editor to specify data objects to extract.

You can specify the following options:

- Indicate if the service should extract data only, objects only or both.
- Select the objects to extract.

To specify data objects to extract:

1. Expand a folder in the Repository Explorer that contains the extract service, expand the **Services** node, and double-click the extract service to edit. The Extract Service Editor opens.
2. Select the **Data and Objects** tab.
3. Edit the service processing options.
4. Save the extract service.

Specifying conversion options

Use the **Conversion** tab in the Extract Service Editor to specify conversion options. You can convert data that is extracted in the service.

You can specify the following options:

- Indicate if the service should convert extracted data.
- Indicated the maximum number discarded rows, which cannot be converted, to allow before stopping the service.

To specify conversion options:

1. Expand a folder in the Repository Explorer that contains the extract service, expand the **Services** node, and double-click the extract service to edit. The Extract Service Editor opens.
2. Select the **Conversion** tab.
3. Edit the service processing options.
4. Save the extract service.

Specifying file compression options

Use the **File Compression Options** tab in the Extract Service Editor to specify options for compressing the file data store or extracted tables.

You can specify the following options:

- Indicate if the file or specific tables are compressed.
- For each table, you can choose to use a compression threshold or compress as much as possible. The threshold value is the minimum amount of reduction in size that you expect to achieve by compressing the table. Enter a value in the range 1 - 99 to set a threshold value for that table.

The following compression methods are available for a file data store:

Inline Compression

Data is compressed as it is extracted and before it is written to the file data store. Inline compression has lower I/O, when compared with post compression, but uses database resources for the duration of the extract service.

Inline Compression requires less storage resources during the extract process when compared to post compression.

Post Compression

Data is extracted and written to an uncompressed file data store. In a second step, Optim reads the uncompressed file data store, and writes a compressed version of the file data store. The benefit of post compression is that database connections are closed earlier with post compression than with inline compression. However, with post compression the total elapsed time is increased because the uncompressed file data store must be closed, read, and then a new compressed version created.

Sites having concerns about database resource contention may find post compression useful as it shortens the time database resources are needed. However, post compression increases the elapsed time and storage requirements for processing the extract service. Although increased storage is necessary during the compression operation, that temporary storage required will be released when the compression is completed.

To specify file compression options:

1. Expand a folder in the Repository Explorer that contains the extract service, expand the **Services** node, and double-click the extract service to edit. The Extract Service Editor opens.
2. Select the **File Compression Options** tab.
3. Edit the service processing options.
4. Save the extract service.

Specifying variable options

Use the **Variables** tab in the Extract Service Editor to override variable default values.

To specify variable options:

1. Expand a folder in the Repository Explorer that contains the extract service, expand the **Services** node, and double-click the extract service to edit. The Extract Service Editor opens.
2. Select the **Variables** tab.
3. Enter values to override default values.
4. Save the extract service.

Changing an access definition for an extract service

Use the Change Access Definition wizard to change an access definition associated with an extract service.

To change an access definition associated with an extract service:

1. Expand a folder in the Repository Explorer that contains the extract service, expand the **Services** node, and double-click the extract service to edit. The Extract Service Editor opens.
2. Click **Change**. The Change Access Definition wizard opens.

3. Complete the steps of the wizard.
   Select an access definition or choose to create a local access definition.
4. Save the extract service.

Working with convert services

Use a convert service to transform data in a file data store. You can transform data to assure data privacy or to systematically transform data to meet your application testing requirements.

Specify a table map to match tables in the source file data store to tables in the target file data store or to exclude tables from the convert service.

Use column maps in the table map to specify which data to convert and how it is to be converted.

Naming conventions

Note: The fully qualified name of a convert service consists of the following: *identifier.name*.

identifier
Identifier that serves as the prefix for the convert service name (1 to 8 characters).

name Name assigned to the convert service (1 to 12 characters).

When you create convert service, it is helpful to use a logical set of naming conventions to identify the use for each and to organize them for easy access.

Creating a convert service

Use the New Service wizard to create a convert service.

A convert service requires a table map. During the creation of a service, you can select or create a table map.

To create a convert service:

1. Expand a folder in the Repository Explorer to contain the service, right-click **Services**, and click **New Service**. The New Service wizard opens.
2. Complete the steps of the wizard.
   You must select or create a table map. You can enter a name for the target file data store or choose to overwrite the source file data store.

The new convert service is displayed in the Convert Service Editor.

Editing a convert service

Use the Convert Service Editor to specify convert service processing options.

Specifying convert process options

Use the **Processing Options** tab in the Convert Service Editor to specify convert service processing options.

You can specify the following convert service processing options:

- Set a discard row limit.
- Compress the target file.
- Allow file attachments.

To specify convert service processing options:

1. Expand a folder in the Repository Explorer that contains the convert service, expand the **Services** node, and double-click the convert service to edit. The Convert Service Editor opens.
2. Select the **Processing Options** tab.
3. Edit the service processing options.
4. Save the convert service.

Changing a table map for a convert service

Use the Change Table Map wizard to change a table map associated with a convert service.

To change a table map associated with a convert service:

1. Expand a folder in the Repository Explorer that contains the convert service, expand the **Services** node, and double-click the convert service to edit. The Convert Service Editor opens.
2. Click **Change**. The Change Table Map wizard opens.
3. Complete the steps of the wizard.

   Select a table map or choose to create a local table map.
4. Save the convert service.

Working with insert services

Use an insert service to insert data stored in a file data store into a destination database.

Use table maps and column maps to map data from the source to the target . You must use a table map to specify the destination for the tables in the file data store. You may also use column maps for one or more destination tables. Column maps enable you to specify the source data for each column and, optionally, to transform the data before it is inserted.

If tables in the file data store do not exist at the target, use the Create utility to create them.

Naming conventions

The fully qualified name of an insert service consists of: *identifier.name*.

identifier
: Identifier that serves as the prefix for the insert service name (1 to 8 characters).

name
: Name assigned to the insert service (1 to 12 characters).

When you create insert services, it is helpful to use a logical set of naming conventions to identify the use for each and to organize them for easy access.

Creating an insert service

Use the New Service wizard to create an insert service.

An insert service requires a table map. During the creation of a service, you can select or create a table map.

To create an insert service:

1. Expand a folder in the Repository Explorer to contain the service, right-click **Services**, and click **New Service**. The New Service wizard opens.
2. Complete the steps of the wizard.

   You must select or create a table map.

The new insert service is displayed in the Insert Service Editor.

Editing an insert service

Use the Insert Service Editor to specify insert service processing options. You can specify options for inserting and deleting rows and also handling triggers and constraints.

Specifying insert processing options

Use the **Processing Options** tab in the Insert Service Editor to specify insert service processing options.

You can specify the following processing options:

- Select the type of processing to be performed. Specify parameters to lock tables, set a commit frequency and set a discard row limit.
- Delete rows from all or specified tables. Delete is useful for restoring data during testing. If a row cannot be deleted for any reason, all deleted rows up to the last commit are restored, and insert processing stops.
- Disable database triggers.
- Disable database constraints.

Row processing includes the following options:

Insert Inserts new rows into the tables.

- If the primary key of a row in the source data *does not match* the primary key of a row in the destination table, the row is inserted.
- If the primary key of a row in the source data *matches* the primary key of a row in the destination table, the row is bypassed and marked as discarded.

Mixed Updates, inserts, or updates/inserts according to your selection for each table on the Specify by Table window. To use the Specify by Table window, click **Specify by Table** and select a process option for each table.

- If you select **Mixed** and do not specify selections on the Specify by Table window, or you set all tables to the same selection, the process option changes to indicate the process used for all tables.

Note: You cannot select **Mixed** if **All rows will be deleted** is selected in the **Row delete options** area.

Update

Updates rows in the tables. Tables must have a primary key.

- If the primary key of a row in the source data *matches* the primary key of a row in the destination table, the row is updated.
- If the primary key of a row in the source data *does not match* the primary key of a row in the destination table, the row is reported as failed.

Note: You cannot select **Update** if **All rows will be deleted** is selected in the **Row delete options** area.

Update/Insert

Updates and inserts rows in tables. Tables must have a primary key.

- If the primary key of a row in the source data *does not match* the primary key of a row in the destination table, the row is inserted.
- If the primary key of a row in the source data *matches* the primary key of a row in the destination table, the row is updated.

Note: You cannot select **Update/Insert** if **All rows will be deleted** is selected in the **Row delete options** area.

To specify insert processing options:

1. Expand a folder in the Repository Explorer that contains the insert service, expand the **Services** node, and double-click the insert service to edit. The Insert Service Editor opens.
2. Select the **Processing Options** tab.
3. Edit the service processing options.
4. Save the insert service.

Changing a table map for a insert service

Use the Change Table Map wizard to change a table map associated with a insert service.

To change a table map associated with a insert service:

1. Expand a folder in the Repository Explorer that contains the insert service, expand the **Services** node, and double-click the insert service to edit. The Insert Service Editor opens.
2. Click **Change**. The Change Table Map wizard opens.
3. Complete the steps of the wizard.

   Select a table map or choose to create a local table map.
4. Save the insert service.

Working with load services

Use a load service to transform the contents of a file data store into a format that is appropriate for a particular DBMS loader and then, if specified, start the corresponding database load utility.

The load service generates a data file in the correct format for each table in the file data store and an SQL file or a BAT file (batch execution), depending on the DBMS, that contains the syntax necessary to start the database loader.

A load service contains the parameters used to prepare data for a DBMS loader and the instructions required to process the load. Specify a table map in the load service to map the destination for the data to load. Use optional column maps in the load service to transform data before loading.

Data to be loaded must be contained in a file data store.

Load versus insert

Optim can move data into a database by using a load or an insert service. Consider the following when deciding which method to use:

- The volume of data and the speed of using the database load utility may offset the advantages of the insert service.
- The data may contain referential integrity (RI) cycles that exceed the capability of the insert service to insert all the data successfully.
- The database load utility requires exclusive control of the database and prevents user access during the load service. The database is available to other users while the insert service is performed.
- The database load utility either inserts new data or replaces existing data. The insert service allows for update/insert processing in one step.

Output file names

A load service generates the following types of files to support the load process of the database utility:

Data files

Data files contain the data you want to load, prepared in the format appropriate for the DBMS you are using. Optim generates a data file for each table in the file data store. Data files are

named the same as the file data store, but contain sequentially numbered file name extensions. For example, a file data store named *demo.xf* that contains three tables will generate three data files named: *demo.001*, *demo.002*, and *demo.003*.

Message files
Message files contain information that the database load utility generates during the load service. Typically, there is one message file for the entire load service. The message file is named the same as the file data store, but contains the extension *.msg*. For example, if the file data store is named *demo.xf*, the message file is named *demo.msg*.

SQL file
For DB2, an SQL file is generated with one statement for each destination table that contains the loader syntax to manually execute the loader. The SQL file is named the same as the file data store, but has the extension *.sql*.

BAT file
For Oracle, Sybase ASE, SQL Server, and Informix, a BAT file is generated that contains the syntax to manually execute the loader for each table. A BAT file is generated for each data store alias specified in the table map. Each BAT file resides in the directory with the corresponding converted file data store. If you chose to manually execute the loader, the BAT file must be edited (in Notepad, for example) to replace a string of eight question marks with specific password information (except Informix).

Additionally, a format file is generated for each data file. A format file has the same name as the corresponding data file, except that the file name extension is different. If there are less than 500 tables to load, the format file name extension is 500 greater numerically than the data file name extension. (For example, if there are three data files named *demo.001*, *demo.002*, and *demo.003*, the corresponding format files are named *demo.501*, *demo.502* and *demo.503*, respectively.) If there are more than 500 tables, a more complex file extension generation algorithm is employed.

Note: If your file server does not allow long file names of greater than 8 characters and the file data store has a long name, the load service will fail. The best solution is to avoid using long file names for file data stores. If needed, you can copy and rename a file before you use it for a load service.

Naming conventions

The fully qualified name of a load service consists of: *identifier.name*.

identifier
Identifier that serves as the prefix for the service name (1 to 8 characters).

name Name assigned to the service (1 to 12 characters).

When you create load services, it is helpful to use a logical set of naming conventions to identify the use for each and to organize them for easy access.

Creating a load service

Use the New Service wizard to create a load service.

A load service requires a table map. During the creation of a service, you can select or create a table map.

To create a load service:

1. Expand a folder in the Repository Explorer to contain the extract service, right-click **Services**, and click **New Service**. The New Service wizard opens.
2. Complete the steps of the wizard.
   You must select or create a table map.

The new load service is displayed in the Load Service Editor.

Editing a load service

Use the Load Service Editor to specify load processing options. You can specify options for the load process and for the DBMS loader associated with each the target data store.

Specifying load processing options

Use the **Processing Options** tab in the Load Service Editor to specify load service processing options.

You can specify the following options:

- Select how to run multiple DBMS loaders when more than one data store alias is used: parallel (to run different DBMS loaders at the same time) or in sequence (to run different DBMS loaders one after another).
- Stop a DBMS loader if an error occurs. If multiple DBMS loaders are running in sequence, processing stops for the DBMS loader with an error and all subsequent DBMS loaders.
- Disable database triggers.
- Disable database constraints.

To specify load service processing options:

1. Expand a folder in the Repository Explorer that contains the load service, expand the **Services** node, and double-click the load service to edit. The Load Service Editor opens.
2. Select the **Processing Options** tab.
3. Edit the service processing options.
4. Save the load service.

Specifying DBMS loader options

Use the **Load Options** tab in the Load Service Editor to specify options for each DBMS loader.

You can specify the following options:

- A processing mode.
- Options that are specific to each target data store alias.

Depending on the DBMS of the selected target file data store, the following processing modes are available:

Insert Inserts rows from the source file data store into empty target tables. If target tables contain data, the loader returns an error.

Replace
Clears and replaces all of the existing rows in the target tables with the rows from the source file data store. (**Replace** might be significantly more resource-intensive than **Truncate** since no logging is performed.)

Append
Inserts the rows from the source file data store into the target tables. If the primary key values match, duplicate rows are discarded or inserted into the exception table (if specified).

Truncate
Truncate is the same as **Replace** but the database does not log the rows being deleted, and Truncate requires that RI constraints are disabled.

To specify DBMS loader options:

1. Expand a folder in the Repository Explorer that contains the load service, expand the **Services** node, and double-click the load service to edit. The Load Service Editor opens.
2. Select the **Load Options** tab.
3. Select the target data store alias.

4. Edit the service processing options.
5. Save the load service.

Changing a table map for a load service

Use the Change Table Map wizard to change a table map associated with a load service.

To change a table map associated with a load service:

1. Expand a folder in the Repository Explorer that contains the load service, expand the **Services** node, and double-click the load service to edit. The Load Service Editor opens.
2. Click **Change**. The Change Table Map wizard opens.
3. Complete the steps of the wizard.

   Select a table map or choose to create a local table map.
4. Save the load service.

Testing a data management service

Use Optim Manager in embedded mode to test a data management service.

To test a data management service:

1. Expand a folder in the Repository Explorer that contains the extract service and expand the **Services** node.
2. Right-click the service and click **Run Service**. Optim Manager opens and the Run Service window is displayed.
3. Click **Run**. To monitor the progress of the service, use the **Service Monitoring** tab.

Chapter 6. Masking data

You can mask data such as national ID numbers, credit card numbers, dates, numeric values, and personal information. Use a column map to enter a data masking function or create a column map procedure with a LUA script. Use a convert service to transform the data.

Applying a data masking function

Use the Column Map Editor to apply and edit a function for a column.

To apply a data masking function:

1. Expand a folder in the Repository Explorer that contains the column map, expand the **Column Maps** node, and double-click the column map. The Column Map Editor opens.
2. Select the column for the policy.
3. Click **Apply Function**. The Apply Function window opens.
4. Select the function to apply. Click **OK**. The function name appears in the associated column and the function editor opens in the Column Map Editor.
5. Depending on the function, do one of the following steps:

   Select the function expression tab and edit the function expression.

   Select an option tab and select options for the function.
6. Save the column map.

Data masking functions

Data masking functions provide various methods to transform or mask sensitive data.

Lookup Functions

Use the lookup functions to select values from a lookup table that are used to populate the target table. Use the Lookup and Hash Lookup functions to select values based on the source value. Alternatively, use the Random Lookup function to select values from a lookup table without regard to the source value.

Lookup Function

The Lookup Function obtains the value for a destination column from a lookup table, according to the value in a source column. There are two forms of the Lookup Function, single column and multiple column.

The single column form inserts a value into a single destination column. The multiple column form inserts values from multiple lookup table columns into corresponding destination columns.

You can enter the multiple column Lookup Function for any source column that will be replaced by a lookup table value, but you must edit the column map to remove the names of remaining source columns that will also be replaced.

The *ignore* parameter allows you to ignore the lookup table and use a source value when a row in a specified source column contains a specified value (NULL, SPACES (for CHAR columns), or zero-length VARCHAR).

You can use the *preserve* parameter to ignore the lookup table and use a source value when a row in a specified source column contains a specified value (NULL, SPACES (for CHAR columns), or zero-length VARCHAR). *preserve* can also be used to ignore the lookup table if a source column does not contain a value.

If a match is not found in the lookup table, a conversion error is reported.

The syntax is:

```
LOOKUP ( [sourcesearchcol, | SRCSEARCH=(sourcecol1,...,sourcecoln)]
           [dest=(col1, coln) , ]
           lktablename ( {LookupTableSearchcol
           | LKPSEARCH=(LookupTableSearchCol1,...,LookupTableSearchColn},
           {value | values=(col1, coln) }
 [,cache | ,nocache ] )
           [,ignore=(colname ( spaces, null, zero_len ), )
           | PRESERVE=( [ NOT_FOUND, ] colname (spaces, null, zero_len),... ) ] )
```

sourcesearchcol
: For single column search, name of the source table column that contains the search value (optional). If not specified, the name of the destination column is used.

SRCSEARCH=
: For multiple column search, names of the source table columns containing the search values. Column names must be enclosed in parentheses and separated by commas.

dest= Names of the destination table columns in which values from the lookup table are inserted. (Required for multiple column lookup.)

col1, coln
: Destination table column names. The order of the column names must correspond to the lookup table columns in the ***values=*** parameter.

lktablename
: Name of the lookup table. You may specify the lookup table name as *dbalias.creatorid.tablename*, *creatorid.tablename*, or *tablename*. If you do not fully qualify the table name, the qualifiers for the destination table are used.

LookupTableSearchcol
: For single column lookup, name of the column in the lookup table that contains a value to match against the search value from the source column.

value Name of the column in the lookup table that contains the translated search value to be inserted at the destination. (Required for single column lookup.)

values=
: Names of the lookup table columns that contain values to be inserted at the destination. (Required for multiple column lookup.)

col1, coln
: Lookup table column names. The order of the column names must correspond to the destination table columns in the *dest=* parameter.

cache | nocache
: Specify *cache* (default) to maintain a table of found lookup values in memory or *nocache* to discard found values. Using *cache* is faster when retrieving a value many times, but requires extra memory.

ignore=
: List of source columns with values that are inserted at the destination instead of the lookup value when the column has a row with a stated value (null, spaces, zero or zero-length varchar).

col The source column name.

For single column lookup, enter one column name only.

For multiple column lookup, the order of the column names must correspond to the destination table columns in the *dest*= parameter. The number of columns must equal the columns in the *dest*= parameter, and at least one column must include values. To not specify values for a column, do not enter a value. For example, *coln()*.

null Ignore the lookup table if the source column row has a null value.

SPACES
Ignore the lookup table if the source column row has a SPACES value. For CHAR columns only.

ZERO_LEN
Ignore the lookup table if the source column row has a zero-length VARCHAR value.

preserve=
List of source columns with values that are inserted at the destination instead of the lookup value when the source column contains a stated value (NOT_FOUND, null, spaces, or zero-length varchar).

NOT_FOUND
Ignore the lookup table if no match is found for the source column row.

Note:

preserve= and *ignore*= are mutually exclusive. *ignore*= will be deprecated in a future release.

The *col*, *null*, *spaces*, and *zero_len* operands have the same effect when used with either *preserve*= or *ignore*=.

Single column example

Use the Lookup Function to translate the source value in a lookup table to a corresponding value in another table.

For example, assume the source column, STATE, contains state abbreviations (for example, **NJ**) and the destination column is to contain the complete state name (in this example, **New Jersey**). A lookup table named STATE_LOOKUP contains a column (CODE) for state abbreviations or codes and a column (NAME) for the corresponding names.

To obtain the value for the destination column using the STATE_LOOKUP table, specify:

```
LOOKUP(STATE,STATE_LOOKUP(CODE,NAME))
```

The Lookup Function searches for a value in the CODE column of the STATE_LOOKUP table that matches the value (**NJ**) in the source table STATE column. When a match is found, the function inserts the corresponding value from the NAME column (**New Jersey**) in the destination column.

Multiple column example

Use the Lookup Function to insert values from columns in a lookup table row into columns in a destination table row, based on a value in a source column.

For example, based on a source column (SOC_SEC) that contains social security numbers, you can replace values in destination columns (FIRST_NAME and LAST_NAME) with first and last names from a lookup table. A table named NAME_LOOKUP contains a column (SSN) with the social security numbers from the source table as well as columns (FIRST_MASK and LAST_MASK) to mask corresponding names in the destination.

To replace names in the destination table based on a social security number, specify:

```
LOOKUP(SOC_SEC,DEST=(FIRST_NAME,LAST_NAME),
NAME_LOOKUP(SSN,VALUES=(FIRST_MASK, LAST_MASK)))
```

The Lookup Function searches for a value in the SSN column of the NAME_LOOKUP table that matches the value in the source table SOC_SEC column. When a match is found, the function inserts the corresponding values from the lookup table FIRST_MASK and LAST_MASK columns into the corresponding destination columns.

Ignore example

Use the following statement to extend the single column example, where you want to use the source *NULL* and *SPACES* values instead of lookup table values:

```
LOOKUP(STATE,STATE_LOOKUP(CODE,NAME),
IGNORE=(STATE(NULL,SPACES)))
```

NoCache example

Use the following statement to extend the single column example, where you do not want to maintain a table of found lookup values in memory:

```
LOOKUP(STATE,STATE_LOOKUP(CODE,NAME),NOCACHE)
```

Hash Lookup Function

The Hash Lookup Function obtains the value for a destination column from a lookup table, according to a hashed value derived from a source column. The Hash Lookup Function allows you to consistently mask data when you use the same source and lookup tables in any environment.

The source column that is hashed does not need to be a column that will be replaced by lookup table values.

The Hash Lookup Function is case-sensitive. For example, the source values John and JOHN will be hashed to different values. You can use the TRIM parameter to convert the source value to uppercase before it is hashed.

There are two forms of the Hash Lookup Function, single column and multiple column. The single column form inserts a value into a single destination column. The multiple column form inserts values from multiple lookup table columns into corresponding destination columns, based on a single hash value from a source column.

You can enter the multiple column Hash Lookup Function for any source column that will be replaced by lookup table values, but you must edit the column map to remove the names of remaining source columns that will also be replaced.

The lookup table must include a key column that contains sequential number values without any gaps, and the remaining columns contain replacement values. The key column must be a numeric data type. The lookup table is typically indexed. The function hashes a source column to derive sequential numbers from 1 to the maximum value in the key column of the lookup table. The hashed value from the source table is matched with the sequential numbers in the lookup table, and values from the corresponding lookup table row are inserted at the destination.

If the source column used to derive the hashed value contains certain values (NULL, spaces (for CHAR columns), zero-length VARCHAR), the value is not hashed and the following reserved values are used as keys to the lookup table:

Source Value	Lookup Table Key
NULL	-1
spaces (CHAR or VARCHAR)	-2
zero-length VARCHAR	-3

The lookup table should include a row for each of these numbers, allowing you to insert a lookup value for each of these source values. If one of these source values is found and a corresponding number is not in the lookup table, a conversion error is reported.

The *ignore* parameter allows you to ignore the lookup table and use a source value when a row in a specified source column contains a specified value (NULL, SPACES (for CHAR columns), or zero-length VARCHAR).

You can use the *preserve* parameter to ignore the lookup table and use a source value when a row in a specified source column contains a specified value (NULL, SPACES (for CHAR columns), or zero-length VARCHAR). *preserve* can also be used to ignore the lookup table if a source column does not contain a value.

The *trim* parameter allows you to specify characters that will be trimmed from the source value before it is hashed. For example, if you choose to trim commas from a source value, the values Smith, John, and Smith John will each be hashed to the same value. You can also use this parameter to convert the source value to uppercase before it is hashed.

If the source value is converted to uppercase, the trim characters are also converted to uppercase.

You can use the *seed* parameter to vary the calculation performed by the hashing algorithm. The hashed value from the source column and the *seed* value are matched with a sequential number from the lookup table to obtain the replacement value for the destination column.

The syntax is:

```
HASH_LOOKUP( [sourcecol,] [trim=([char1char2 ] [\u]),]
 dest=(col1, coln), lktablename (search,
  { value | values=(col1, coln) } ) [ ,cache | ,nocache ]
 [,ignore=(col (spaces, null, zero_len ), )
| PRESERVE=( [ NOT_FOUND, ] colname (spaces, null, zero_len), ) ] )[,seed=n])
```

sourcecol
: Name of the source table column from which hashed values are derived (optional). If not specified, the name of the destination column is used.

trim=
: List of characters to be trimmed from the source value before it is hashed as well as an option to convert the source value to uppercase before it is hashed. If the resulting source value is NULL or all spaces after characters have been trimmed, the source value will not be hashed and will be assigned the appropriate reserved value (-1 or -2).

  char1char2...
  : Characters to be trimmed from the source value before it is hashed. The list is case-sensitive. You can specify a space or comma as a character. After the initial occurrence of a character, any additional occurrences in the list are ignored.

    To specify a backslash "\" or a right parentheses ")", you must precede the character with a backslash escape character. For example, to specify a right parentheses, enter: *trim=(\))*.

    You can only use the escape character with a backslash, a right parentheses, or as part of the uppercase indicator.

\u Indicates the source value is to be converted to uppercase before it is hashed. The characters to be trimmed are also converted to uppercase.

dest= Names of the destination table columns in which values from the lookup table are inserted. (Required for multiple column lookup.)

col1,coln
Destination table column names. The order of the column names must correspond to the lookup table columns in the values= parameter.

lktablename
Name of the lookup table. You may specify the lookup table name as **dbalias.creatorid.tablename**, **creatorid.tablename**, or **tablename**. If you do not fully qualify the table name, the qualifiers for the destination table are used.

search Name of the column in the lookup table that contains sequential values to match against the hash values from the source column.

value Name of the column in the lookup table that contains the translated search value to be inserted at the destination. (Required for single column lookup.)

values=
Names of the columns in the lookup table that contain values to be inserted at the destination. (Required for multiple column lookup.)

col1,coln
Lookup table column names. The order of the column names must correspond to the destination table columns in the *dest=* parameter.

cache | nocache
Specify *cache* (default) to maintain a table of found lookup values in memory or *nocache* to discard found values. Using *cache* is faster when retrieving a value many times, but requires extra memory.

ignore=
List of source columns with values that are inserted at the destination instead of the lookup value when the column has a row with a stated value (NULL, SPACES, ZERO, or zero-length VARCHAR).

col The source column name.

For single column lookup, enter one column name only.

For multiple column lookup, the order of the column names must correspond to the destination table columns in the *dest=* parameter. The number of columns must equal the columns in the *dest=* parameter, and at least one column must include values. To not specify values for a column, do not enter a value. For example, *coln()*.

NULL Ignore the lookup table if the source column row has a NULL value.

SPACES
Ignore the lookup table if the source column row has a SPACES value. For CHAR columns only.

ZERO_LEN
Ignore the lookup table if the source column row has a zero-length VARCHAR value.

preserve=
List of source columns with values that are inserted at the destination instead of the lookup value when the column contains a stated value (NOT_FOUND, null, spaces, or zero-length varchar).

NOT_FOUND
Ignore the lookup table if no match is found for the source column row.

Note:

preserve= and *ignore*= are mutually exclusive. *ignore*= will be deprecated in a future release.

The*col*, *null*, *spaces*, and *zero_len* operands have the same effect when used with either*preserve*= or *ignore*=.

seed= Use *seed*= to vary the hashing algorithm calculation. Values from 1 to 2,000,000,000 can be used. If you use a value of 0, the *seed*= parameter is ignored.

Single column example

Use the Hash Lookup Function to insert values from a column in a lookup table into a destination table column, based on a value hashed from a source column.

For example, assume the source column, FIRST_NAME, contains first names and the destination column will include replacement first names from the lookup table. A lookup table, NAME_LOOKUP, contains a column (FIRST) with first names and a column (SEQ) containing sequential values.

To obtain values for the destination column using the NAME_LOOKUP table, specify:

```
HASH_LOOKUP(FIRST_NAME,NAME_LOOKUP(SEQ, FIRST))
```

The Hash Lookup Function matches the hash values from the source column with values in the SEQ column of the NAME_LOOKUP table. When a match is found, the function inserts the corresponding value from the FIRST column into the destination column.

Multiple column example

Use the Hash Lookup Function to insert values from columns in a lookup table row into columns in a destination table row, based on a value hashed from a source column.

For example, based on values hashed from a source column (FIRST_NAME) that contains first names, you can replace values in destination columns (FIRST and LAST) with first and last names from a lookup table. A lookup table named NAME_LOOKUP contains a column (SEQ) with sequential values as well as columns (FIRST_MASK and LAST_MASK) to mask values in the destination.

To replace names in the destination table based on values hashed from a source column, specify:

```
HASH_LOOKUP(FIRST_NAME,DEST=(FIRST,LAST), NAME_LOOKUP(SEQ,VALUES=(FIRST_MASK, LAST_MASK)))
```

The Hash Lookup Function matches the hash values from the source FIRST_NAME column with values in the SEQ column of the NAME_LOOKUP table. When a match is found, the function inserts the corresponding values from the lookup table FIRST_MASK and LAST_MASK columns into the corresponding destination columns.

Ignore example

Use the following statement to extend the single column example, where you want to use the source NULL and SPACES values instead of lookup table values:

```
HASH_LOOKUP(FIRST_NAME,NAME_LOOKUP(SEQ, FIRST),IGNORE=(FIRST_NAME(NULL,SPACES)))
```

NoCache example

Use the following statement to extend the single column example, where you do not want to maintain a table of found lookup values in memory:

```
HASH_LOOKUP(FIRST_NAME,NAME_LOOKUP(SEQ, FIRST),NOCACHE)
```

Trim example

Use the following statement to extend the single column example, where you want to trim spaces and commas from the source value as well as convert the source value to uppercase before it is hashed:

```
HASH_LOOKUP(FIRST_NAME, TRIM=( ,\u),NAME_LOOKUP(SEQ,FIRST))
```

Random Lookup Function

The Random Lookup Function selects a value at random from a specified lookup table to insert in a destination column. The function generates a random number between 1 and the limit or number of rows in the lookup table to use as a subscript into the table. The column value or values from the row that correspond to the subscript are inserted in the destination column.

There are two forms of the Random Lookup Function, single column and multiple column. The single column form inserts a value into a single destination column. The multiple column form inserts values from multiple lookup table columns into corresponding destination columns.

You can enter the multiple column Random Lookup Function for any source column that will be replaced by a lookup table value, but you must edit the column map to remove the names of remaining source columns that will also be replaced.

The *ignore* parameter allows you to ignore the lookup table and use a source value when a row in a specified source column contains a specified value (NULL, SPACES (for CHAR columns), or zero-length VARCHAR).

You can use the *preserve* parameter to ignore the lookup table and use a source value when a row in a specified source column contains a specified value (NULL, SPACES (for CHAR columns), or zero-length VARCHAR).

The syntax is:

```
RAND_LOOKUP(lktablename, { columnname | dest=(col1,coln) ,values=(col1,coln) }
 [,limit ] [,ignore=(col(spaces, null, zero_len), ) | PRESERVE=( colname (spaces, null, zero_len), ) ] )
```

lktablename
: Name of the lookup table. You may specify the lookup table name as ***dbalias.creatorid.tablename***, ***creatorid.tablename***, or ***tablename***. If the table name is not fully qualified, destination table qualifiers are used.

columnname
: Name of the column in the lookup table that contains the values to be randomly selected for insertion at the destination. (Required for single column lookup.)

dest=
: Names of the destination table columns in which values from the lookup table are inserted. (Required for multiple column lookup.)

  col1,coln
  : Destination table column names. The order of the column names must correspond to the lookup table columns in the *values*= parameter.

values=
: Names of the columns in the lookup table that contain values to be inserted at the destination. (Required for multiple column lookup.)

  col1,coln
  : Lookup table column names. The order of the column names must correspond to the destination table columns in the *dest*= parameter.

limit Optional limit on number of rows from the lookup table used to select column values. Specify an integer, up to a maximum value of 2,000,000,000. If no limit is specified, all rows are used.

Note: A table of column values is generated in memory. The size of this table may be limited by system resources.

ignore=
List of source columns with values that are inserted at the destination instead of the lookup value when the column has a row with a stated value (NULL, SPACES, or zero-length VARCHAR).

col The source column name.

For single column lookup, enter one column name only.

For multiple column lookup, the order of the column names must correspond to the destination table columns in the *dest=* parameter. The number of columns must equal the columns in the *dest=* parameter, and at least one column must include values. To not specify values for a column, do not enter a value. For example, *coln()*.

null Ignore the lookup table if the source column row has a NULL value.

spaces Ignore the lookup table if the source column row has a SPACES value. For CHAR columns only.

zero_len
Ignore the lookup table if the source column row has a zero-length VARCHAR value.

preserve=
List of source columns with values that are inserted at the destination instead of the lookup value when the column contains a stated value (NOT_FOUND, null, spaces, or zero-length varchar).

NOT_FOUND
Ignore the lookup table if no match is found for the source column row.

Note:

preserve= and *ignore=* are mutually exclusive. *ignore=* will be deprecated in a future release.

The *col*, *null*, *spaces*, and *zero_len* operands have the same effect when used with either *preserve=* or *ignore=*.

Single column example

To select a value at random from the STATE column in the first 50 rows of a table named STATE_LOOKUP and insert it in the destination column, specify:

```
RAND_LOOKUP(STATE_LOOKUP,STATE,50)
```

Multiple column example

To select values from the CITY, STATE, and ZIPCODE columns in a random row of a table named STATE_LOOKUP and insert them in the corresponding destination columns, specify:

```
RAND_LOOKUP(STATE_LOOKUP,
DEST=(CITY,STATE,ZIPCODE),
VALUES=(CITY,STATE,ZIP))
```

Ignore example

Use the following statement to extend the single column example, where the source column is named STATES and you want to use the source NULL and SPACES values instead of lookup table values:

```
RAND_LOOKUP(STATE_LOOKUP,STATE,50, IGNORE=(STATES(NULL,SPACES)))
```

Shuffle Function

The Shuffle Function replaces a source value with another value from the column that is then inserted in a destination column. The source row and the row that contains the replacement value will never be the same, but depending on your data, source and replacement values can be identical.

You can indicate the number of times the function will refetch a replacement value until a value that does not match the source value is found (a "retry"), or you can allow a replacement value to match the source. Each Shuffle Function operates independently of other Shuffle Functions used in a column map.

There are two forms of the Shuffle Function, single column and multiple column. The single column form inserts a replacement value into a single destination column. The multiple column form inserts replacement values from multiple columns in a row into corresponding destination columns. A column cannot be included in more than one Shuffle Function in a column map. If the retry feature is used with a multiple column shuffle, the function will refetch another replacement row if any value in the source row column matches the value in a corresponding replacement row column. (The multiple column form cannot be used in a Propagate Function.)

To create a multiple column Shuffle Function, enter the function for a source column that will be replaced with shuffled values, and edit the column map to remove the names of any other source columns with values that will also be replaced.

The *ignore* parameter prevents the function from replacing a source row or using a replacement row if either contains a specified value (NULL, SPACES (for CHAR columns), or zero-length VARCHAR). If no retries are allowed, the *ignore* parameter will not apply to the replacement row.

The syntax is:

```
SHUFFLE [ ( dest=(col1,coln) ) ] |
[ ( dest=(col1,coln) , retry[=number] ) ] |
[ ( dest=(col1,coln) [ , retry[=number] ] , ignore=( col1 ( [spaces] | [spaces,null]
| [spaces,null,zero_len] | [null] | [null,zero_len] | [zero_len] ) , coln (....) ) ) ] |
[ ( retry[=number] ) ] |
[ ( retry[=number] , ignore=( col ( [spaces] | [spaces,null] | [spaces,null,zero_len]
| [null] | [null,zero_len] | [zero_len] ) ) ) ] |
[ ( ignore=( col ( [spaces] | [spaces,null] | [spaces,null,zero_len]
| [null] | [null,zero_len] | [zero_len] ) ) ) ]
```

where:

dest= Names of the destination table columns in which replacement values are inserted. (Required for multiple column shuffle.)

col1, *coln*, **...**
Destination table column names.

retry Number of times to refetch a replacement value to find a value that does not match the source row. Enter zero to allow a replacement value to match the source.

Note: Using a high retry value with columns that contain many duplicate values will increase the processing time. For these columns, it may be best to use a retry value of zero.

=number
Enter a value in the range 0-1000. Enter 0 to allow a replacement value to match the source.

ignore=
List of columns for which the function will not replace a source value or not use a replacement value if either is a specified value (NULL, SPACES (for CHAR columns), or zero-length

VARCHAR). If a replacement value is ignored, the function will refetch another replacement value. If no retries are allowed, the *ignore*= parameter will not apply to replacement values.

col The source column name.

For single column shuffle, enter one column name only.

For multiple column shuffle, the order of the column names must correspond to the destination table columns in the *dest*= parameter. The number of columns must equal the columns in the *dest*= parameter, and at least one column must include values. To not specify values for a column, do not enter a value. For example, *coln()*.

null Do not replace the source value or use a replacement value if either is a NULL value.

spaces Do not replace the source value or use a replacement value if either is a SPACES value. For CHAR columns only.

zero_len
Do not replace the source value or use a replacement value if either is a zero-length VARCHAR value.

Single column default example

The following example inserts shuffled values in a single column.

```
SHUFFLE
```

Single column retry example

The following example inserts shuffled values in a single column and refetches a replacement value that does not match the source up to 12 times.

```
SHUFFLE(RETRY=12)
```

Multiple column example

The following example inserts shuffled values in the STATE and ZIP columns and refetches a replacement value that does not match the source up to 12 times.

```
SHUFFLE(DEST=(STATE,ZIP),RETRY=12)
```

Ignore example

The following example inserts shuffled values in the STATE and ZIP columns and refetches a replacement value that does not match the source up to 12 times. The example also does not replace a source value or use a replacement value for the STATE column if a source or replacement row contains a NULL or SPACES value, but does not ignore any source or replacement rows for the ZIP column.

```
SHUFFLE(DEST=(STATE,ZIP),RETRY=12,
IGNORE=(STATE(NULL,SPACES),ZIP()))
```

TRANS SSN Function

Use the TRANS SSN function to generate a valid and unique U.S. Social Security Number (SSN). By default, TRANS SSN algorithmically generates a consistently altered destination SSN based on the source SSN. TRANS SSN can also generate a random SSN when the source data does not have an SSN value or when there is no need for transforming the source SSN in a consistent manner.

An SSN is made of 3 subfields. The first 3 digits (area) represent an area generally determined by the state in which the SSN is issued. The next 2 digits (group) define a group number corresponding to the area number. The last 4 digits (serial) are a sequential serial number. Regardless of the type of processing, default or random, TRANS SSN will generate an SSN with a group number appropriate to the area number.

The default processing method generates an SSN that includes the source area number as well as altered group and serial numbers based on the source SSN.

The random processing method generates an SSN that can include the source area number and uses a group number most recently issued by the Social Security Administration for the destination area number. Serial numbers begin with 0001 and are incremented by 1 for each additional SSN generated for the area number. When the serial number exceeds 9999, the serial number will be reset to 0001 and the group number preceding the number most recently issued for the area number will be used.

The syntax of TRANS SSN is:

TRANS SSN [('[=*flags*] [*sourcecol* **[preserve=invalid]')]**

flags You can specify one or more case-insensitive processing option flags.

- *n* Generate a random SSN that is not based on a source value.
- *m* Use the maximum group of all SSN area values, including values from 773 through 899, and excluding invalid area numbers.
- *r* Generate a SSN with a random area number corresponding to the same state as input SSN.
- *v* Validate the source group number by comparing it with numbers used by the Social Security Administration.
- \- The destination SSN should include dashes separating the fields (for example, 123-45-6789). Requires a character-type destination column at least 11 characters long.

sourcecol
The source column name. If a source column name is not specified, the destination column name will be used. If a source column name is not specified and the destination column name does not match a column name in the source table, an error will occur during processing.

preserve=invalid
If the source column contains an invalid SSN, do not replace it with a generated value. The source column value will be used in the destination column.

Data types allowed

The following source and destination data types are permitted:

CHAR
The length of data in the column must be from 9 to 256 characters.

DECIMAL
The precision of the column must be 9 - 20 and the scale 0.

INTEGER
No restrictions.

VARCHAR
The length of data in the column must be from 9 to 254 characters.

If a source or destination column does not adhere to these restrictions, an error will occur during processing.

Destination processing rules

The following rules apply to the destination SSN value, according to the destination data type or value:

CHAR
If the source value is 0, spaces, or a zero-length VARCHAR, the destination value will be set to spaces.

If a source value is 11 characters or more and includes embedded dashes (-), or if the '-' flag is specified, the destination value will include dashes if the destination column length is 11 characters or more.

DECIMAL, INTEGER
If the source value is 0, spaces, or a zero-length VARCHAR, the destination value will be 0.

VARCHAR
If the source value is 0, spaces, or a zero-length VARCHAR, the destination length will be 0.

If a source value is 11 characters or more and includes embedded dashes (-), or if the '-' flag is specified, the destination value will include dashes if the destination column length is 11 characters or more.

NULL If the source value is NULL, the destination value will be NULL.

Skipped rows

The following conditions may cause a source row to be skipped and not written to the destination:

- The source value is NULL, and the destination column does not allow a NULL value.
- The source column is CHAR or VARCHAR, and the source value is less than 9 characters, contains a non-numeric character (other than dashes between the 3 subfields), or is too large.
- The source area number has not been used by the Social Security Administration.
- The source group number has not been used with the area number by the Social Security Administration (only if the 'v' flag has been specified).
- The source serial number is 0000, or the SSN is a reserved value not issued by the Social Security Administration (for example, 078-05-1120).
- The source value cannot be converted to a format TRANS SSN supports.

Error messages

The following error messages may be issued:

SSN01
Parm on Col ccccc ("ppp") is invalid

Explanation
The column contains a TRANS function with a processing option flag that is not valid.

User Action
Ensure that the TRANS function on the column specified uses a valid processing option flag (n, r, v, -).

SSN02
Col ccccc not on source

Explanation
The column that was entered as a *sourcecol* parameter or the destination column name (if the *sourcecol* parameter was omitted) was not found on the source table.

User Action
Check the source table and resolve any discrepancies or missing columns.

SSN03

Source Col ccccc-aaa invalid

Explanation

The format of the source column is not supported because the attribute indicated is not valid.

User action

Check the source column and ensure the values for type, length, precision, and scale are appropriate.

SSN04

Dest Col ccccc-aaa invalid

Explanation

The format of the destination column is not supported because the indicated attribute is not valid.

User Action

Check the destination column to ensure the values for type, length, precision, and scale are appropriate.

SSN05

Get col ccccc data-rc=nnn

Explanation

An unexpected internal error has occurred while getting the value from the source column.

User Action

Check the values of the source and destination columns and ensure the values for type, length, precision, and scale are appropriate. If the problem persists, contact IBM Software Support.

SSN08

Put col ccccc data-rc=nnn

Explanation

An unexpected internal error has occurred while setting the value on the destination column.

User Action

Check the values of the source and destination columns and ensure the values for type, length, precision, and scale are appropriate. If the problem persists, contact IBM Software Support.

If any other errors occur, contact Technical Support.

Example 1

The following example uses a source column name that matches the destination column and generates a random SSN that is not based on the source value:

```
TRANS SSN ('=n')
```

Example 2

The following example uses a source column name (NATIONAL_ID) that differs from the destination column and generates an SSN using the default processing method and including dashes:

```
TRANS SSN ('=- NATIONAL_ID')
```

TRANS CCN Function

Use the TRANS CCN function to generate a valid and unique credit card number (CCN). By default, TRANS CCN algorithmically generates a consistently altered CCN based on the source CCN. TRANS CCN can also generate a random value when the source data does not have a CCN value or when there is no need for transforming the source CCN in a consistent manner.

A CCN, as defined by ISO 7812, consists of a 6-digit issuer identifier followed by a variable length account number and a single check digit as the final number. The check digit verifies the accuracy of the CCN and is generated by passing the issuer identifier and account numbers through the Luhn algorithm. The maximum length of a CCN is 19 digits.

The default processing method generates a CCN by including the first 4 digits of the issuer identifier from the source CCN and altering the remaining 2 digits of the issuer identifier number and the account number based on the source CCN. A valid check digit is also assigned.

The random processing method generates a CCN that can include the first 4 digits of the source issuer identifier number or an issuer identifier number assigned to American Express, Discover, MasterCard, or VISA. A valid check digit is also assigned. If the first four digits of a source issuer identifier number are included, the first account number based on those digits will begin with 1, and for each additional CCN that uses those digits, the account number will be incremented by 1.

The syntax of TRANS CCN is:

TRANS CCN [('[=*flag*] [*sourcecol*] [preserve=invalid]')]

flag Specify an option flag to generate a random CCN.

- *n* Generate a random CCN that is not based on a source value and includes an issuer identifier number assigned to American Express, Discover, MasterCard, or VISA.
- *r* Generate a random CCN that includes the first 4 digits of the source issuer identifier number.

sourcecol

The source column name. If a source column name is not specified, the destination column name is used.

If a source column name is not specified and the destination column name does not match a column name in the source table, an error will occur during processing.

preserve=invalid

If the source column contains an invalid CCN, do not replace it with a generated value. The source column value will be used in the destination column.

Data types allowed

The following source and destination data types are permitted:

CHAR

The column length must be from 13 to 256 characters.

VARCHAR

The column length must be from 13 to 254 characters.

DECIMAL

The precision of the column must be from 13 to 254 and the scale 0.

If a source or destination column does not adhere to these restrictions, an error message occurs.

Destination processing rules

The following rules apply to the destination CCN value, according to the destination data type or value:

CHAR
If the source value is spaces or a zero-length VARCHAR, the destination value will be set to spaces.

VARCHAR
If the source value is spaces or a zero-length VARCHAR, the destination length will be 0.

DECIMAL
If the source value is 0, the destination value will be 0.

NULL If the source value is NULL, the destination value will be NULL.

Skipped rows

The following conditions may cause a source row to be skipped and not written to the destination:

- The source value is NULL, and the destination column does not allow a NULL value.
- The source value is less than 13 characters, contains a non-numeric character, is too large, or has an incorrect check digit.
- The source value length is not valid for the credit card issuer.
- The source value cannot be converted to a format TRANS CCN supports.

Error messages

The following error messages may be issued:

CCN01
Parm on Col ccccc ("ppp") is invalid

Explanation
The indicated column contains a TRANS function with a processing option flag that is not valid.

User Action
Ensure that the TRANS function on the column specified uses a valid processing option flag (n, r, 6).

CCN02
Col ccccc not on source

Explanation
The column that was entered as a *sourcecol* parameter or the destination column name (if the *sourcecol* parameter was omitted) was not found on the source table.

User Action
Check the source table and resolve any discrepancies or missing columns.

CCN03
Source Col ccccc-aaa invalid

Explanation
The format of the source column is not supported because the attribute indicated is not valid.

User Action
Check the source column and ensure the values for type, length, precision, and scale are appropriate.

CCN04

Dest Col ccccc-aaa invalid

Explanation

The format of the destination column is not supported because the indicated attribute is not valid.

User Action

Check the destination column to ensure the values for type, length, precision, and scale are appropriate.

CCN05

Get col ccccc data-rc=nnn

Explanation

An unexpected internal error has occurred while getting the value from the source column.

User Action

Check the values of the source and destination columns and ensure the values for type, length, precision, and scale are appropriate. If the problem persists, contact IBM Software Support.

CCN08

Put col ccccc data-rc=nnn

Explanation

An unexpected internal error has occurred while setting the value on the destination column.

User Action

Check the values of the source and destination columns and ensure the values for type, length, precision, and scale are appropriate. If the problem persists, contact IBM Software Support.

If any other errors occur, contact Technical Support.

Example 1

The following example uses a source column name (CREDITCARD) that differs from the destination column and generates a random CCN not based on the source value:

```
TRANS CCN ('=n CREDITCARD')
```

Example 2

The following example uses a source column name (CREDITCARD) that differs from the destination column and generates a CCN using the default processing method:

```
TRANS CCN ('CREDITCARD')
```

TRANS EML Function

Use the TRANS EML function to generate an email address. An email address consists of two parts, a user name followed by a domain name, separated by '@'. For example, user@domain.com.

TRANS EML generates an email address with a user name based on either destination data or a literal concatenated with a sequential number. The domain name can be based on an email address in the source data, a literal, or randomly selected from a list of large email service providers. The email address can also be converted to upper or lower case.

TRANS EML can generate a user name based on the values in one or two destination table columns (usually containing the name of a user). Processing options allow you to use only the first character of the value in the first column (for example, the initial letter of a first name) and separate the values from both columns using either a period or an underscore.

If the user name is based on a single destination column value or a literal, the name will be concatenated with a sequential number. If a user name is based on values in two destination table columns and a separating period or underscore is not used, the values are concatenated. If a parameter is not provided for the user name, the name will be formed by the literal “email” concatenated with a sequential number. Sequential numbers for user names are suffixes that begin with 1 and are incremented by 1.

The syntax of TRANS EML is:

```
TRANS EML [( '[=flags] , [{sourcecol | "domain" | , }
[{name1col[name2col] | "userpfx"}] ] [preserve=invalid]' )]
```

flags You can specify one or more case-insensitive processing option flags.

- *n* Generate a random domain name from a list of large email service providers.
- *.* Separate the *name1col* and *name2col* values with a period.
- *_* Separate the *name1col* and *name2col* values with an underscore.
- *i* Use only the first character of the *name1col* value.
- *l* Convert the email address to lower case.
- *u* Convert the email address to uppercase.

sourcecol
The source column name with email addresses used to provide the domain name.

If neither the ‘n' flag nor the *domain* parameter are defined, the domain name in the source column is used. (If *sourcecol* is not defined, the source column name is based on the destination column name.)

If a source column name is not specified and the destination column name does not match a column name in the source table, an error will occur during processing.

domain A literal, up to 31 characters, that forms the domain name.

, A comma is required if neither a *sourcecol* nor a *domain* parameter is defined and you define either a literal or column name(s) for the domain name.

name1col
A destination table column name with values used to form the first (or only) part of the user name.

name2col
A destination table column name with values used to form the second part of the user name.

userpfx A literal, up to 31 characters, that is concatenated with a sequential number to form the user name.

preserve=invalid
If the source column contains an invalid email address, do not replace it with a generated value. The source column value will be used in the destination column.

Data types allowed

The following source and destination data types are permitted:

CHAR
The column length must be from 3 to 4096 characters.

VARCHAR
The column length must be from 3 to 4094 characters.

If a source or destination column does not adhere to these restrictions, an error message will be issued.

Destination processing rules

The following rules apply to the destination email value, according to the destination data type or value:

CHAR
If the source value is spaces or a zero-length VARCHAR, the destination value will be set to spaces.

VARCHAR
If the source value is spaces or a zero-length VARCHAR, the destination length will be 0.

NULL If the source value is NULL, the destination value will be NULL.

Skipped rows

The following conditions may cause a source row to be skipped and not written to the destination:

- The source value is NULL, and the destination column does not allow a NULL value.
- The source value is a VARCHAR less than 3 characters long.
- The source email value does not contain a '@'.
- The source value cannot be converted to a format TRANS EML supports.

Error messages

The following error messages may be issued:

EML01
Parm on Col ccccc ("ppp") is invalid

Explanation
The indicated column contains a TRANS function with a processing option flag that is not valid.

User Action
Ensure that the TRANS function on the column specified uses a valid processing option flag (n, . , _ , i, l, u).

EML02
Col ccccc not on source

Explanation
The column that was entered as a *sourcecol* parameter or the destination column name (if the *sourcecol* parameter was omitted) was not found on the source table.

User Action
Check the source table and resolve any discrepancies or missing columns.

EML03
Source Col ccccc-aaa invalid

Explanation
The format of the source column is not supported because the attribute indicated is not valid.

User Action

Check the source column and ensure the values for type, length, precision, and scale are appropriate.

EML04

Dest Col ccccc-aaa invalid

Explanation

The format of the destination column is not supported because the indicated attribute is not valid.

User Action

Check the destination column to ensure the values for type, length, precision, and scale are appropriate.

EML05

Get col ccccc data-rc=nnn

Explanation

An unexpected internal error has occurred while getting the value from the source column.

User Action

Check the values of the source and destination columns and ensure the values for type, length, precision, and scale are appropriate. If the problem persists, contact IBM Software Support.

EML08

Put col ccccc data-rc=nnn

Explanation

An unexpected internal error has occurred while setting the value on the destination column.

User Action

Check the values of the source and destination columns and ensure the values for type, length, precision, and scale are appropriate. If the problem persists, contact IBM Software Support.

EML09

Domain literal sssss too long

Explanation

The string specified as the domain name literal (domain) exceeds the maximum limit of 31 characters.

User Action

Specify a domain name consisting of 31 characters or fewer.

EML10

User literal sssss too long

Explanation

The string specified as the user name literal (*userpfx*) exceeds the maximum limit of 31 characters.

User Action

Specify a user name consisting of 31 characters or fewer.

EML11

Name1 Col ccccc not on dest

Explanation
To perform the indicated TRANS function, a *name1col* column must be specified on the destination table.

User Action
Verify that the specified *name1col* column matches the *name1col* column indicated in the TRANS function.

The *name1col* column name was not found on the destination table.

EML12
Name1 Col ccccc-aaa invalid

Explanation
To perform the indicated TRANS function, the *name1col* column specified must be in a valid format.

User Action
Verify that the *name1col* column availability, type, and length are appropriate.

EML13
Name2 Col ccccc not on dest

Explanation
To perform the indicated TRANS function, a *name2col* column must be specified on the destination table.

User Action
Verify that the specified *name2col* column matches the *name2col* column indicated in the TRANS function.

The *name2col* column name was not found on the destination table.

EML14
Name2 Col ccccc-aaa invalid

Explanation
To perform the indicated TRANS function, the *name2col* column specified must be in a valid format.

User Action
Verify that the *name2col* column availability, type, and length are appropriate.

If any other errors occur, contact Technical Support.

Example 1

The following example uses a literal (optim.com) to form the domain name and two destination table columns (NAME_FIRST and NAME_LAST) to form a user name that includes an underscore:

```
TRANS EML ('=_ "optim.com" NAME_FIRST NAME_LAST')
```

Example 2

The following example uses a domain name from the source column and a literal (OptimUser) to form a user name that will be suffixed with a sequential number:

```
TRANS EML (', "OptimUser" ')
```

TRANS COL Function

The TRANS COL function can mask data that has no inherent format or a format that is not widely known. TRANS COL maintains the format and character type of the source data at the destination.

If the source data is upper case, alphabetic characters, TRANS COL generates upper case, alphabetic characters at the destination. This function masks alphabetic and numeric characters, but other characters in the source data are copied to the destination without being changed. You can use TRANS COL to mask CHAR, VARCHAR, and non-float numeric data types. You can generate unique values, a different value for each occurrence of the same source, and you can generate values with a length different from the source.

The syntax of TRANS COL is:

```
TRANS COL ( '{ unique | hash }[ source=colname ]
[ copy=( (start,len [, "lit" ] )...) ]
[ seed= {"lit" | var (variable )| RANDOM} ]
[ length=n | max ] [ preserve=( [ null ] [ spaces ] [ zero_len ] ) ]
[ TRIM=(char1 [ charn,...] [\u] [\r] ) ] [num]' )
```

unique Generate a unique destination value. The length of the destination value will be the same as the source value length.

hash Generate a destination value by hashing the source value. When hash is used, different source values can produce the same destination values each time the process is run.

Note:
For the same source value, it is possible to obtain the same destination value when either the unique parameter or the hash parameter is used. Use hash with the seed parameter to produce different destination values each time the process is run.

source=**colname**
Use this parameter to specify the name of the source column if the destination column is different from the source column name. The value you specify will be converted to uppercase; to prevent the value from conversion to uppercase, enclose the value in double quotation marks.

copy= One or more pairs of substrings to be copied to the destination without being masked. If you supply a literal string, the source characters in the specified positions are replaced. The *copy=* parameter is valid only for a character data type column.

seed= Value used to alter the behavior of the masking algorithms. Specify a literal string, reference to an environment variable, or RANDOM.

"lit" To specify a literal string, enclose the string in double quotation marks.

var **(variable)**
Specify an environment variable enclosed in parentheses. The variable name and its value cannot include double quotation marks.

RANDOM
Generate a random seed value from the current system date and time.

length={n | max }
Generate a destination value with a length different from the source value length. Use *length=max* to generate a destination value that will fill the column completely. Specifying a length shorter than the source value causes the source value to be truncated when it is written to the destination. The value you select for *n* cannot exceed the defined length of the destination column. The *length=* parameter is valid only with *hash=*.

preserve=
List one or more source values that should not be replaced at the destination.

null If the source column has a null value do not replace the value at the destination.

spaces If the source column has a value of spaces do not replace the value at the destination. For CHAR columns only.

zero_len
If the source column has a zero-length VARCHAR value do not replace the value at the destination.

TRIM=(char1 [charn,...])
The specified source column character or characters are not masked and not written to the destination. For example, if you specify *TRIM=(x,y,z,1,2,3)*, if any of the characters x, y, z, 1, 2, 3 appear anywhere in the source column, they will not be masked or written to the destination.

[\u] Use this parameter to convert the character(s) to upper case before masking. If a character has no upper case representation, it remains unchanged. For example, specifying *TRIM=(x,y \u)* does not mask the characters x and y if they appear anywhere in the source column, and changes any other source column characters to upper case before masking them.

[\r] Use this operand to remove trailing spaces. For example, *TRIM=(x,y \u \r)* does not mask the characters x and y if they appear anywhere in the source column, changes any other source column characters to upper case before masking them, and removes any trailing spaces before masking.

num Use this parameter to cause the transformation of integers in a character data type column to be identical to that of a numeric data type column. The num parameter is valid only on numeric values in a character data type column. When used in this way, foreign key integrity is maintained across differing data types. If you use this parameter, do not specify *copy=* or *length=*.

Examples:

- TRANS COL UNIQUE for source value **CDE-7834**, could produce a destination value of: **ZWQ-4598**
- TRANS COL UNIQUE SEED=RANDOM for source value **CDE-7834-2008** could produce a destination value of: **SWX-3162-8451**
- TRANS COL UNIQUE for source value **Smith, John** could produce a destination value of: **Fnxwq, Lrzp**
- TRANS COL UNIQUE for source value **SMITH JOHN** could produce a destination value of: **FNXWQ LRZP**
- TRANS COL UNIQUE (COPY=((1,3)(10,4))) for source value **CDE-7834-2008** could produce a destination value of: **CDE-4032-2008**
- TRANS COL HASH (LENGTH=13) for source value **CDE-7834** could produce a destination value of: **ZWQ-4598RN7A0**
- TRANS COL UNIQUE PRESERVE=(spaces) for source value **XYZ 477 6835** could produce a destination value of: **LMN 623 0972**
- TRANS COL HASH TRIM=(e \u) for source value **InfoSphere** could produce a destination value of: **RBIWACRL**

TRANS NID

Use the TRANS NID function to mask national ID numbers.

You can mask national ID numbers with either a repeatable method that preserves part of the source value or a random method that does not preserve any part of the source value.

You can also specify the type of separators used in the output values (dashes, periods, spaces, or no separators).

The TRANS NID function uses the following syntax:

```
[TRANS] NID ('SWI=country_code, [FMT=(output_format)], [MTD={mask|random}],
          [SRC=column_name], [VAL={Y|N}], [PRE=INV]')
```

SWITCH **or** *SWI*

A two-character value that indicates the type of national ID to mask. This parameter is required. Only one switch value is permitted.

The following two-character values are valid:

Canada: Canadian Social Insurance Number (SIN)
CA

France: French National Institute for Statistics and Economic Studies Number (INSEE)
FR

Italy: Italian Fiscal Code Number (CF)
IT

Spain: Spanish Fiscal Identification Number (NIF)
ES

United Kingdom: United Kingdom National Insurance Number (NINO)
UK

United States of America: United States Social Security Number (SSN)
US

FMT **or** *FORMAT*

Determines the output format and which parts of the source value to mask. The syntax for this parameter is determined by the national ID being masked:

- Canadian Social Insurance Number (SIN)
- French National Institute for Statistics and Economic Studies Number (INSEE)
- Italian Fiscal Code Number (CF)
- Spanish Fiscal Identification Number (NIF)/Foreign Identification Number (NIE)
- United Kingdom National Insurance Number (NINO)
- United States Social Security Number (SSN)

If this parameter is omitted and *MTD=mask*, the source format is used and the default fields will be masked.

If this parameter is omitted and *MTD=random*, the output values will not include separators.

If the destination column is not large enough to contain an output format that uses separators, separators will not be included.

MTD **or** *METHOD*

Determines which masking method to use, repeatable or random. If this parameter is omitted, repeatable masking is performed (MTD=mask).

mask The source values are masked in a repeatable manner. The output values are based on the source values. Default.

random
The output values are generated by a random masking algorithm. The output values are not based on the source values.

Note:

- *MTD=random* is not compatible with the following parameters: *VAL=Y* and *PRE=INV*.

- If *MTD=random* and the *FMT* parameter specifies that part of the source value should be copied to the output value, the output value will not include source values; however, any separators specified in the *FMT* parameter will be included in the output value.

SRC **or** *SOURCE*
The name of the column that contains the source values. Use this parameter only if the names of the source and destination columns do not match.

VAL **or** *VALIDATE*
Determines if country-specific validation is performed on the source values. If this parameter is omitted, validation is not performed (*VAL=N*).

Y Validate the source values.

Note: The option cannot be used if *MTD=random*.

N Do not validate the source values. Default.

PRE **or** *PRESERVE*
Determines if invalid source values are copied to the destination column. If this parameter is omitted, invalid source values are not copied to the destination column and rows that contain these values are skipped. This parameter includes only one option, INV or INVALID.

INV **or** *INVALID*
Copy invalid source values to the destination column.

Note: The option cannot be used if *MTD=random*.

Examples

The following syntax will mask United States Social Security Numbers (SSN) with a repeatable method. The function will copy the first three digits of the source SSN and include a dash separator for the output value. Validation will be performed on the output value.

```
NID('SWITCH=US, FMT=(US=3C-2X-4X), MTD=MASK, VAL=Y')
```

The following syntax will mask French National Institute for Statistics and Economic Studies numbers (INSEE) with a random method. The function will use the default output format, which will not include separators.

```
NID('SWITCH=FR, MTD=RANDOM')
```

The following syntax will mask Spanish Fiscal Identification Numbers (NIF) with a repeatable method. The function will preserve invalid source values and use the default output format.

```
NID('SWITCH=ES, MTD=MASK, PRE=INV')
```

Canadian Social Insurance Number masking

You can use the TRANS NID function to mask Canadian Social Insurance Numbers (SIN).

An SIN is a nine-digit number that consists of a one-digit region code number followed by an eight-digit serial number. The first three digits are called the header. The last digit of the serial number is a check digit.

The TRANS NID function generates a masked SIN with a check digit that is calculated based on the preceding masked eight digits of the output value.

Output formats *(FMT=)*

The following output formats are available for an SIN.

C indicates values that are copied. *X* indicates values that are masked. For example, *3C4X* indicates that the first three characters are copied and the next four characters are masked.

Fields to be masked	Format without separator	Format with dash separator	Format with space separator	Format with period separator
Serial number without header digits (*MTD=mask* default)	CA=3C6X	CA=3C-3X-3X	CA=3C 3X 3X	CA=3C.3X.3X
Serial number and header digits	CA=9X	CA=3X-3X-3X	CA=3X 3X 3X	CA=3X.3X.3X

Data types

The following data types are permitted for source and destination columns. Restrictions for each data type are noted.

CHAR and NCHAR
The length of the column must be at least 9 characters.

DECIMAL
The precision of the column must be between 9 and 20 and the scale must be 0.

INTEGER
No restrictions.

VARCHAR and NVARCHAR
The length of the column must be at least 9 characters.

Validation *(VAL=Y)*

If the *(VAL=Y)* parameter is used, a source row will be skipped if any of the following apply:

- The first digit is eight.
- There are three consecutive zeros at positions 1-3, 4-6, or 7-9.
- The check digit is invalid.

Special processing

The following checks will be made during processing:

- If the source value is 0 (INTEGER or DECIMAL data type), spaces, or either a zero-length VARCHAR or NVARCHAR, and
  - the destination column is an INTEGER or DECIMAL data type, the output value will be 0.
  - the destination column is a CHAR or NCHAR data type, the output value will be spaces.
  - the destination column is a VARCHAR or NVARCHAR data type, the destination length will be 0.
- If a destination column length is 11 characters or more and separators were specified for the output value, the separators will be included if the destination column is a CHAR, NCHAR, VARCHAR, or NVARCHAR data type.
- If the source value is NULL, the output value will be NULL.

Skipped rows

A source row will be skipped and not written to the destination table if the following occurs:

Null value
The source value is NULL but the destination column does not allow a NULL value.

Invalid SIN

- The source column is a CHAR, NCHAR, VARCHAR, or NVARCHAR but the source value is less than 9 characters (not including separators).
- The source column is a CHAR, NCHAR, VARCHAR, or NVARCHAR but the source value is more than 9 characters (not including separators).
- The source value includes a non-numeric character.

French National Institute for Statistics and Economic Studies Number masking

You can use the TRANS NID function to mask French National Institute for Statistics and Economic Studies numbers (INSEE).

An INSEE number is a 15-digit number with the following format: *SYYMMDDCCCOOOKK*.

S Sex and citizenship information.

YY Last two digits of the year of birth.

MM Month of birth.

DD Department of origin.

CCC Commune of origin.

OOO Order number.

KK Control key or check digits.

The TRANS NID function generates a masked INSEE according to the following rules:

- If the department field is masked, the commune field is also masked with a compatible value.
- The order field is always masked.
- The check digit field is calculated based on the preceding masked 13 digits of the output value.

Output formats *(FMT=)*

The following output formats are available for an INSEE.

All formats mask the order and check digit fields. If the department field is masked, the commune field will also be masked with a compatible value.

C indicates values that are copied. *X* indicates values that are masked. For example, *3C4X* indicates that the first three characters are copied and the next four characters are masked.

Fields to be masked (in addition to Order and Check Digit)	Format without separator	Format with dash separator	Format with space separator
Sex, Year, Month, Commune (*MTD=mask* default)	FR=5X2C8X	FR=5X2C6X-2X	FR=5X2C6X 2X
Sex	FR=1X9C5X	FR=1X9C3X-2X	FR=1X9C3X 2X
Sex, Year	FR=3X7C5X	FR=3X7C3X-2X	FR=3X7C3X 2X
Sex, Month	FR=1X2C2X5C5X	FR=1X2C2X5C3X-2X	FR=1X2C2X5C3X 2X
Sex, Commune	FR=1X6C8X	FR=1X6C6X-2X	FR=1X6C6X 2X
Sex, Department	FR=1X4C8X	FR=1X4C6X-2X	FR=1X4C6X 2X
Sex, Year, Month	FR=5X5C5X	FR=5X5C3X-2X	FR=5X5C3X 2X
Sex, Year, Commune	FR=3X4C8X	FR=3X4C6X-2X	FR=3X4C6X 2X
Sex, Year, Department, Commune	FR=3X2C10X	FR=3X2C8X-2X	FR=3X2C8X 2X

Fields to be masked (in addition to Order and Check Digit)	Format without separator	Format with dash separator	Format with space separator
Sex, Month, Commune	FR=1X2C2X2C8X	FR=1X2C2X2C6X-2X	FR=1X2C2X2C6X 2X
Sex, Month, Department, Commune	FR=1X2C12X	FR=1X2C10X-2X	FR=1X2C10X 2X
Sex, Year, Month, Department, Commune	FR=15X	FR=13X-2X 13X	FR=13X 2X
Year	FR=1C2X7C5X	FR=1C2X7C3X-2X	FR=1C2X7C3X 2X
Year, Month	FR=1C4X5C5X	FR=1C4X5C3X-2X	FR=1C4X5C3X 2X
Year, Commune	FR=1C2X4C8X	FR=1C2X4C6X-2X	FR=1C2X4C6X 2X
Year, Department	FR=1C2X2C10X	FR=1C2X2C8X-2X	FR=1C2X2C8X 2X
Year, Month, Commune	FR=1C4X2C8X	FR=1C4X2C6X-2X	FR=1C4X2C6X 2X
Year, Month, Department	FR=1C14X	FR=1C12X-2X	FR=1C12X 2X
Month	FR=3C2X5C5X	FR=3C2X5C3X-2X	FR=3C2X5C3X 2X
Month, Commune	FR=3C2X2C8X	FR=3C2X2C6X-2X	FR=3C2X2C6X 2X
Month, Department	FR=3C12X	FR=3C10X-2X	FR=3C10X 2X
Commune	FR=7C8X	FR=7C6X-2X	FR=7C6X 2X
Department	FR=5C10X	FR=5C8X-2X	FR=5C8X 2X

Data types

The following data types are permitted for source and destination columns. Restrictions for each data type are noted.

CHAR and NCHAR
The length of the column must be at least 15 characters.

VARCHAR and NVARCHAR
The length of the column must be at least 15 characters.

Validation *(VAL=Y)*

If the *(VAL=Y)* parameter is used, a source row will be skipped if any of the following apply:

- The source commune field value is invalid.
- The source check digit field is invalid.

Special processing

The following checks will be made during processing:

- If the source value is spaces or either a zero-length VARCHAR or NVARCHAR, and
  - the destination column is a CHAR or NCHAR data type, the output value will be spaces.
  - the destination column is a VARCHAR or NVARCHAR data type, the destination length will be 0.
- If the source value is NULL, the output value will be NULL.
- If a destination column length is 16 characters or more and a separator was specified for the output value, the separator will be included.

Skipped rows

A source row will be skipped and not written to the destination table if the source value includes the following errors:

Null value
The source value is NULL but the destination column does not allow a NULL value.

Invalid INSEE

- The size of the input INSEE value is more or less than valid.
- The input INSEE value contains invalid separators or separators in the wrong position.
- The sex field is not one of the following values: 1, 2, 7, or 8.

Italian Fiscal Code number masking

You can use the TRANS NID function to mask Italian Fiscal Codes (CF).

A CF is a 16-character alphanumeric value with the following format: *FFF-NNN-YYMDD-RRRRC*.

FFF Encoded surname.

NNN Encoded given name.

YY Year of birth.

M Month of birth.

DD Date of birth.

RRRR Region code.

C Control character.

The TRANS NID function generates a masked CF according to the following rules:

- Any consonant that appears in the given name or surname fields is masked as a consonant and any vowel is masked as a vowel. If an X appears after a vowel, it must be copied to the output value.
- The control character field is calculated based on the preceding masked 15 digits of the output value.

Output formats *(FMT=)*

The following output formats are available for a CF.

C indicates values that are copied. *X* indicates values that are masked. For example, *3C4X* indicates that the first three characters are copied and the next four characters are masked.

Fields to be masked	Format without separator	Format with dash separator	Format with space separator
Date of birth, Region (*MTD=mask* default)	IT=6C10X	IT=3C-3C-5X-5X	IT=3C 3C 5X 5X
Surname, Given name, Region	IT=6X5C5X	IT=3X-3X-5C-5X	IT=3X 3X 5C 5X
Surname, Given name, Date of birth	IT=11X4C1X	IT=3X-3X-5X-4C1X	IT=3X 3X 5X 4C1X
Surname, Given name	IT=6X9C1X	IT=3X-3X-5C-4C1X	IT=3X 3X 5C 4C1X
Date of birth	IT=6C5X4C1X	IT=3C-3C-5X-4C1X	IT=3C 3C 5X 4C1X
Region	IT=11C5X	IT=3C-3C-5C-5X	IT=3C 3C 5C 5X
Surname, Given name, Date of birth, Region	IT=16X	IT=3X-3X-5X-5X	IT=3X 3X 5X 5X

Data types

The following data types are permitted for source and destination columns. Restrictions for each data type are noted.

CHAR and NCHAR
The length of the column must be at least 16 characters.

VARCHAR and NVARCHAR
The length of the column must be at least 16 characters.

Validation *(VAL=Y)*

If the *VAL=Y* parameter is used, a source row will be skipped if it contains an invalid control character.

Special processing

The following checks will be made during processing:

- If the source value is spaces or either a zero-length VARCHAR or NVARCHAR, and
  - the destination column is a CHAR or NCHAR data type, the output value will be spaces.
  - the destination column is a VARCHAR or NVARCHAR data type, the destination length will be 0.
- If the source value is NULL, the output value will be NULL.
- If a destination column length is 19 characters or more and separators were specified for the output value, the separators will be included.

Skipped rows

A source row will be skipped and not written to the destination table if the following occurs:

Null value
The source value is NULL but the destination column does not allow a NULL value.

Invalid CF
The source column has an invalid source value or the source value is too large.

The source value is less than 16 characters.

Spanish Fiscal Identification Number and Foreign Identification Number masking

You can use the TRANS NID function to mask Spanish Fiscal Identification Numbers (NIF) and Foreign Identification Numbers (NIE).

An NIF is an eight character value in the following format *NNNNNNN-A*, where the first seven characters are a serial number and the final character is an alphabetic suffix. The suffix is a check digit.

Foreign Spanish nationals use a Foreign Identification Number (NIE), which is a nine character value that uses the same format as an NIF, but is preceded by an X. An NIE uses the following format: *X-NNNNNNN-A*.

The TRANS NID function generates a masked NIF or NIE with a check digit that is calculated based on the preceding masked 7 digits of the output value.

Output formats *(FMT=)*

The following output formats are available for an NIF and NIE.

For each format, all characters are masked. NIF and NIE numbers use the same format options. An NIE source value will always include an X prefix in the output value.

Fields to be masked	Format without separator	Format with dash separator	Format with space separator
Serial, Suffix (*MTD=mask* default)	ES=8X	ES=7X-1X	ES=7X 1X

Data types

The following data types are permitted for source and destination columns. Restrictions for each data type are noted.

CHAR and NCHAR
The length of the column must be at least 8 characters.

VARCHAR and NVARCHAR
The length of the column must be at least 8 characters.

Validation *(VAL=Y)*

If the *VAL=Y* parameter is used, a source row will be skipped if it contains an invalid suffix.

Special processing

The following checks will be made during processing:

- If the source value is spaces or either a zero-length VARCHAR or NVARCHAR, and
  - the destination column is a CHAR or NCHAR data type, the output value will be spaces.
  - the destination column is a VARCHAR or NVARCHAR data type, the destination length will be 0.
- If the source value is NULL, the output value will be NULL.
- If a destination column length is 11 characters or more and a separator was specified for the output value, the separator will be included.

Skipped rows

A source row will be skipped and not written to the destination table if the following occurs:

Invalid length
The input value does not contain separators and is less than 8 characters (NIF) or is less than 9 characters (NIE).

The input value contains separators and is less than 9 characters (NIF) or is less than 11 characters (NIE).

Invalid pattern
The length of the source value is valid, but the value does not match the pattern of an NIF or NIE.

Separators
Different separators were found in positions 2 and 10 in an NIE source value.

The source value contains an invalid separator.

Null value
The source value is NULL but the destination column does not allow a NULL value.

United Kingdom National Insurance Number masking

You can use the TRANS NID function to mask United Kingdom National Insurance Numbers (NINO).

A NINO consists of three parts: two letters (the prefix), six digits (the number), and one optional letter (the suffix).

Output formats *(FMT=)*

The following output formats are available for a NINO.

A NINO can be masked without a separator or with a separator in either a three- or five-part format.

C indicates values that are copied. *X* indicates values that are masked. For example, *3C4X* indicates that the first three characters are copied and the next four characters are masked.

To create a NINO without a separator, use the following parameters:

Fields to be masked	**Format without separator**
Prefix, Number	UK=8X1C
Number (*MTD=mask* default)	UK=2C6X1C
Prefix, Number, Suffix (*MTD=random* only)	UK=9X

To create a NINO with either a three- or five-part format, use the following parameters:

Fields to be masked	**Format with dash separator**	**Format with space separator**	**Format with period separator**
Prefix, Number (three-part)	UK=2X-6X-1C	UK=2X 6X 1C	UK=2X.6X.1C
Prefix, Number (five-part)	UK=2X-2X-2X-2X-1C	UK=2X 2X 2X 2X 1C	UK=2X.2X.2X.2X.1C
Number (three-part)	UK=2C-6X-1C	UK=2C 6X 1C	UK=2C.6X.1C
Number (five-part)	UK=2C-2X-2X-2X-1C	UK=2C 2X 2X 2X 1C	UK=2C.2X.2X.2X.1C
Prefix, Number, Suffix (three-part) (*MTD=random* only)	UK=2X-6X-1X	UK=2X 6X 1X	UK=2X.6X.1X
Prefix, Number, Suffix (five-part) (*MTD=random* only)	UK=2X-2X-2X-2X-1X	UK=2X 2X 2X 2X 1X	UK=2X.2X.2X.2X.1X

Data types

The following data types are permitted for source and destination columns. Restrictions for each data type are noted.

CHAR and NCHAR
The length of the column must be at least 9 characters.

VARCHAR and NVARCHAR
The length of the column must be at least 9 characters.

Validation *(VAL=Y)*

The *VAL=Y* parameter is not valid for NINO masking and will be ignored.

Special processing

The following checks will be made during processing:

- If the source value is spaces or either a zero-length VARCHAR or NVARCHAR, and

  - the destination column is a CHAR or NCHAR data type, the output value will be spaces.
  - the destination column is a VARCHAR or NVARCHAR data type, the destination length will be 0.
- If the source value is NULL, the output value will be NULL.
- If the destination column cannot accommodate a separator, the output value will not include a separator.

Skipped rows

A source row will be skipped and not written to the destination table if the following occurs:

Null value
The source value is NULL but the destination column does not allow a NULL value.

Invalid NINO
- The size of the source value is less or greater than the size of a valid NINO.
- The source value includes separators in the wrong positions.
- The source value includes an invalid separator.
- The source value includes an invalid prefix.
- The source value includes a suffix other than A, B, C, or D.
- The source value includes a number field that is not between 000001 and 999999.

United States Social Security Number masking

You can use the TRANS NID function to mask United States Social Security Numbers (SSN).

An SSN consists of 3 subfields with the following format: *AAAGGSSSS*.

AAA Area number. The area is generally determined by the state in which the SSN is issued.

GG Group number. A group number is assigned based on the area number.

SSSS Serial number.

The TRANS NID function generates a masked SSN according to the following rules:
- A group number that is appropriate for the area number is generated. The group number will be the most recent group used by the Social Security Administration for the area.
- Serial numbers begin with 0001 and are incremented by 1 for each additional SSN generated for the area number. When the serial number exceeds 9999, the serial number will be reset to 0001 and the group number preceding the number most recently issued for the area number will be used.
- When *MTD=mask*, the output value will include an area number corresponding to the same state as the source area number.

Output formats *(FMT=)*

The following output formats are available for an SSN.

C indicates values that are copied. *X* indicates values that are masked. For example, *3C4X* indicates that the first three characters are copied and the next four characters are masked.

Fields to be masked	Format without separator	Format with dash separator	Format with space separator	Format with period separator
Group, Serial number (*MTD=mask* default)	US=3C6X	US=3C-2X-4X	US=3C 2X 4X	US=3C.2X.4X
Area, Group, Serial number	US=9X	US=3X-2X-4X	US=3X 2X 4X	US=3X.2X.4X

Data types

The following data types are permitted for source and destination columns. Restrictions for each data type are noted.

CHAR and NCHAR
The length of the column must be at least 9 characters.

DECIMAL
The precision of the column must be between 9 and 20 and the scale must be 0.

INTEGER
No restrictions.

VARCHAR and NVARCHAR
The length of the column must be at least 9 characters.

Validation *(VAL=Y)*

If the *(VAL=Y)* parameter is used, a source row will be skipped if any of the following apply:

- The source area number exceeds the maximum value.
- The source area number has not been used by the Social Security Administration.
- The source group number has not been used for the source area number.

Special processing

The following checks will be made during processing:

- If the source value is 0 (INTEGER or DECIMAL data type), spaces, or either a zero-length VARCHAR or NVARCHAR, and
  - the destination column is an INTEGER or DECIMAL data type, the output value will be 0.
  - the destination column is a CHAR or NCHAR data type, the output value will be spaces.
  - the destination column is a VARCHAR or NVARCHAR data type, the destination length will be 0.
- If a destination column length is 11 characters or more and separators were specified for the output value, the separators will be included.
- If the source value is NULL, the output value will be NULL.

Skipped rows

A source row will be skipped and not written to the destination table if the following occurs:

Null value
The source value is NULL but the destination column does not allow a NULL value.

Invalid SSN

- The source column is a CHAR, NCHAR, VARCHAR, or NVARCHAR, but the source value is more or less than 9 characters (not including separators).
- The source value includes a non-numeric character.
- The source value is 0 or a reserved value such as 078-05-1120 and 457-55-5462.

Substring Function

The Substring Function returns a substring of the contents of the named column.

`SUBSTR(columnname, start, [length])`

columnname
Name of a character or binary column.

start The position of the first character in the string.

length The number of characters to use.

- If the locale uses a comma as the decimal separator, you must leave a space after each comma that separates numeric parameters (for example, after the comma between *start* and *length*.
- *start* and *length* are integers greater than or equal to 1.
- *start* plus *length* cannot exceed the total data length plus 1.
- *column-name* and *start* value are required. If you specify only one integer, it is used as the *start* value. The substring begins at *start* and includes the remainder of the column value.

Example

If the PHONE_NUMBER column is defined as CHAR(10), you can use the Substring Function to map the area code. To obtain a substring of the first three positions of the phone number (area code) for the destination column, specify:

```
SUBSTR(PHONE_NUMBER, 1, 3)
```

Random Function

The Random Function returns a number selected at random within the range indicated by the low and high values.

```
RAND(low, high)
```

low Lowest possible random value.

high Highest possible random value.

- Use the Random Function with character or numeric data.
- If the locale uses a comma as the decimal separator, you must leave a space after the comma.
- *low* and *high* are integers within the range **-2,147,483,648** to **2,147,483,647**.
- *low* and *high* are further limited by the data type and length for the destination column.
- *low* must be less than *high*.
- When you use the Random Function in a concatenated expression, a variable length string is returned.

Example

You can use the Random Function to mask or change sales data for a test database. Assume the YTD_SALES column is defined as DECIMAL(7,2). The maximum number of digits preceding the decimal is 5; the possible range for this column is -99999 to 99999. To create test data within a range from 1000 (low) to 89999 (high), specify:

```
RAND(1000, 89999)
```

In this example, the function returns random sales values within the range you specified from 1000.00 to 89999.99.

Sequential Function

The Sequential Function returns a number that is incremented sequentially. The syntax is:

```
SEQ(start, step)
```

start Start value.

step Incremental value.

- Use the Sequential Function with character and numeric data.

- If the locale uses a comma as the decimal separator, you must leave a space after the comma.
- *start* and *step* are integers within the range of **-2,147,483,648** and **2,147,483,647**.
- *start* and *step* are further limited by the data type and length of the destination column.
- If the calculated value exceeds the length of the destination column, the function automatically resets to the *start* value.
- When you use the Sequential Function in a concatenated expression, a variable length string is returned.

Example 1

You can use the Sequential Function to change customer data for a test database. Assume that the CUST_ID column is defined as CHAR(5). To increment by 50, starting at 1, specify:

```
SEQ(1, 50)
```

In this example, the function returns CUST_ID values starting at '00001' and increments by 50 to generate '00051', '00101', etc. When the result exceeds '99951', the function resets to the *start* value of 1.

Example 2

You can use the Sequential Function in a column map to mask sales data for a test database. Assume that the YTD_SALES column is defined as DECIMAL(7,2). To increment by 100 starting at 1000, specify:

```
SEQ(1000, 100)
```

In this example, the function returns YTD_SALES values starting at 1000 and increments by 100 to generate 1100, 1200, etc. When the result exceeds 99999, the function resets to the *start* value of 1000.

Example 3

Assume that the SALESMAN_ID column is defined as CHAR(6). To insert values beginning with 'NJ,' followed by a number starting at 50 and incremented by 10, use the function in a concatenated expression:

```
'NJ'||SEQ(50, 10)
```

In this example, the function returns SALESMAN_ID values starting at 'NJ50 ' and increments by 10 to generate 'NJ60 ', 'NJ70 ', etc. When the result exceeds 'NJ9990', the function resets to the *start* value.

Identity or Serial Function

The Identity and Serial Functions direct the DBMS to supply a sequential value (integer) for a destination column. The syntax for these functions is:

```
IDENTITY( )

SERIAL ( )
```

- Use the Identity Function for Identity columns in DB2, Sybase ASE, and SQL Server databases.
- Use the Serial Function for Serial columns in Informix databases.
- Both functions are valid for Insert (update/insert) and Load Processing, but are not valid for Convert Processing.
- If rows are updated in an Insert Process (update/insert), the destination column targeted by the Identity Function or Serial Function retains the original value. In addition, if the destination column is part of the primary key, the column value remains unchanged when the row is updated.

- You can use the Identity Function or Serial Function with the Propagate Function for Insert Processing; however, you cannot propagate Identity or Serial columns in a Load Process.

Oracle Sequence Function

The Oracle Sequence Function assigns a value to the destination column by using an Oracle Sequence.

```
schema.seqname.NEXTVAL [INCL_UPD]
```

schema Qualifier for the Oracle Sequence name.

seqname
Name of the Oracle Sequence that assigns sequential values.

NEXTVAL
Keyword that inserts the next Oracle value into the destination column.

INCL_UPD
Optional keyword that updates a sequence value assigned to a column when rows are updated during an insert process. If not specified (default), the column value remains unchanged when the row is updated.

- You can use the Oracle Sequence Function to assign unique sequential values for rows to be inserted into an Oracle database.
- The Oracle Sequence Function is valid when used in a column map for insert or load processing, but is not valid for convert processing.
- If rows are updated in an insert (update/insert) process and the destination column is part of the primary key, the column value remains unchanged when the row in the destination table is updated. To use Oracle Sequence when performing an update/insert, include *INCL_UPD* with the function.
- During a load process, the process uses the Oracle Sequence Function to assign a new value to each destination row. The Load calls the DBMS to obtain these values. If you choose not to run the Oracle Loader, these sequence values are never used.

Example 1

To assign a sequential value to increment customer numbers, where the name of the Oracle Sequence is **schema.numeven**, specify:

```
schema.numeven.NEXTVAL
```

Example 2

To expand the first example and update an existing sequence value, specify:

```
schema.numeven.NEXTVAL(INCL_UPD)
```

Propagate Primary or Foreign Key Value Function

The Propagate Function assigns a value to a primary key or foreign key column and propagates that value to all related tables.

The syntax is:

```
PROP( { value [, columnname| ] EXIT exitname |
PROC { LOCAL | identifier.name } } )
```

value Value to assign to the column. Specify any valid column map source value (for example, a column name, string literal, expression, or function). The value must be appropriate for the column.

columnname
Name of the source column that contains the value that is the subject of the function. The resulting value is inserted into the destination column of the mapped table and the appropriate destination column in the participating related tables.

The column name is required *only* if no source column matches the destination column in both name and data type. If not specified, the name of the destination column is used.

exitname
A column map exit name.

identifier.name
A column map procedure name.

- If the locale uses a comma as the decimal separator, you must leave a space after each comma that separates numerical parameters.
- The Propagate Function is valid in a column map for insert (but not update or update/insert), load, or convert processing.
- When you use the Propagate Function, at least one related table must be included in the process. You can use Propagate multiple times for the same process.
- You can use the Propagate Function for either a primary key column or its corresponding foreign key column, but not both.
- If multiple columns define a relationship, you can use the Propagate Function for one or more of those columns. However, in an Optim extended relationship, you can specify the Propagate Function only on column-to-column relations.
- You can use the Identity Function or the Serial Function within the Propagate Function for insert processing; however, you cannot propagate the Identity Function in a load process.
- The parameters specified in the Propagate Function are not validated until run time. If there are conflicts, the process does not run.
- Insert can have propagate cycles. However, load and convert processing may not result in propagate cycles. Cycles are detected when the process is validated at run time. If a load or convert request generates a propagate cycle, the process does not run.
- Optim remembers the source values and the values assigned to corresponding destination columns. Therefore, you can propagate to destination columns where the source is an expression. When the evaluated expression matches a source value, Optim assigns the corresponding destination value. When the evaluated expression does not match any source values, a conversion error occurs.

Before executing an insert, load, or convert process, you can review the column map to verify how the Propagate Function is used in the process.

Example 1

You can generate a random number, assign it to the default destination column, and propagate the number in the destination columns of related tables. To generate a value between 10000 and 99999, insert it into the mapped destination column and propagate it to the destination columns of related tables, specify:

```
PROP(RAND(10000, 99999))
```

Example 2

You can perform the same function as in Example 1 when the source and destination column names do not match. To include the name of the source column (CUST_NUMBER) in the Propagate Function, specify:

```
PROP(RAND(10000, 99999), CUST_NUMBER)
```

Example 3

You can use Oracle Sequence to generate the value for the destination column and propagate that value in destination columns of the related tables. To propagate the Oracle Sequence named, **schema.numeven**, specify:

```
PROP(schema.numeven.NEXTVAL)
```

Concatenated expressions

Concatenation allows you to combine column values or combine a column value with another value, by using a concatenation operator (CONCAT, ||, or +). A concatenated expression can include character values or binary values, but not both:

Character Values
Concatenated character values can be character columns, string literals, substrings of values in character columns, the sequential function, or the random function.

Binary Values
Concatenated character values can be binary columns, hexadecimal literals, substrings of binary columns, the sequential function, or the random function.

- A concatenated expression cannot include a zero-length string literal (' '), a special register, or the Age Function.

Example

Assume that the CUSTOMERS table stores an address in two columns: ADDRESS1 and ADDRESS2. The SHIP_TO table stores an address in one column: ADDRESS. You can use a concatenated expression to combine address information from two columns in one table to one column in another.

To combine the address, specify one of the following examples:

CUSTOMERS Table	SHIP_TO Table
ADDRESS1 || ADDRESS2	ADDRESS
ADDRESS1 CONCAT ADDRESS2	ADDRESS
ADDRESS1 + ADDRESS2	ADDRESS

Numeric expressions

Use a numeric expression to specify a value in the source column whenever the data types for the corresponding source and destination columns are compatible.

An numeric expression consists of the following syntax:

```
operand1 operator operand2
```

Each operand must be a numeric column or a numeric constant. The operator specifies whether to add (+), subtract (-), divide (/), or multiply (*).

Example 1

To increase the value in a column UNIT_PRICE defined as DECIMAL(5,2) by 10 percent, specify the following example:

```
1.1 * UNIT_PRICE
```

Example 2

To divide the value in an INTEGER column ON_HAND_INVENTORY in half, specify the following example:

```
ON_HAND_INVENTORY / 2.
```

Literal and value functions

Use the literal and value functions to specify literals such as a string or hexadecimal and values such as NULL or a special register.

Function	Description
Column Name	An explicit column name (column names are case⌂insensitive).
NULL	NULL. The destination column must be nullable.
Numeric Constant	A numeric constant. The constant value must fit into the destination column as defined by its data type, precision, and scale.
Boolean Constant	A Boolean constant (TRUE or FALSE).
Special Register	A special register: • CURRENT DATE • CURRENT_DATE • CURRENT TIME • CURRENT_TIME • CURRENT TIMESTAMP • CURRENT_TIMESTAMP • CURRENT SQLID • CURRENT_SQLID • CURDATE( ) • CURTIME( ) • GETDATE( ) • GETTIME( ) • SYSDATE( ) • NOW( ) • WORKSTATION_ID • USER
String Literal	A string literal, enclosed in single quotes. The destination column must contain character data. Example: 'CA' or '90210'.
Hexadecimal Literal	A hexadecimal literal. Example: X‘1234567890ABCDEF' or 0X1234567890ABCDEF
Date/Time Literal	A date/time literal, enclosed in single quotes. Separate the date and time with a space. To format the date/time with a decimal fraction, place a period after the time, followed by the fraction. The date format is determined by the settings in **Regional Options** on the Control Panel of your computer. **Note:** For Oracle Timestamp with Time Zone columns, you must specify the time zone suffix last.

Age Function

Use the Age Function to age values in a source column. The source column can contain character, numeric, date, or timestamp data. A CHAR or VARCHAR column has a maximum length of 256 bytes.

The Age Function is formatted as:

`AGE(parameters)`

- Define the Age Function to include one or more aging parameters.
- Use commas or spaces to separate parameters in the Age Function.
- Parameters can be specified in any order.

Parameter	Format	Valid Values
Column Name: Specify the name of the source column if it differs from the destination column.	*SC=column-name* *SRCCOL=column-name*	Column Name
Default: Age dates based on the date adjustment value specified in a process request.	*DEF*	Uses date adjustment value specified in the process request.
None: Do not age value.	*NONE*	Value should not be aged regardless of specifications in the process request.
Incremental – Incremental Aging is based on a known time unit. Optim supports date aging in single units (for example 20 years) or multiple units (for example, 2 years, 3 months, 2 days).	+ or -] *n*Y [+ or -] *n*M [+ or -] *n*W [+ or -] *n*D (The plus [+] sign is optional.)	*n*Y -2500 to +1581 *n*M -30000 to +30000 *n*W -30000 to +30000 *n*D -99999 to +99999
Specific Year: Age dates based on a specific four⌂digit year in the desired format.	*nnnn*Y	1582 - 3999
Multiple/Rule: Age dates based on the number of times to apply a business rule. If you define the Age Function using the Multiple/Rule, you must also include the RULE parameter.	*nnnnn*R	1 - 30000

Semantic Aging

Semantic Aging is based on a set of rules that you define to manage dates that occur on holidays, weekends, and so on. You can use Semantic Aging to adjust dates so that they occur on valid business days.

Calendar –

Name of the calendar that defines the special dates to which the rules apply. If you use CALENDAR, you must also specify a RULE.
CA=*calendar-name*
CALENDAR=*calendar-name*

Rule –

Name of the rule that defines the adjustment for special dates. If DEF is specified, the default rule specified in the process request is used.
RU=*rule-name*
RULE=*rule-name*
RU=DEF
RULE=DEF

Century Pivot –

Determines the century for two-digit years. Enter a value 00 to 99.
PI=*nn*
PIVOT=*nn*

- If you specify **AGE(RU=DEF)**, the RULE specified in a process request is used. You must specify values for any other age function parameters.
- If you use RULE and do not specify a CALENDAR, then the Age Function uses the default calendar you specify in a process request.
- If you do not include CALENDAR, RULE, and PIVOT where needed in the Age Function, the default values you specify in a process request apply.
- To specify the correct century for a two-digit year, you must include the PIVOT in the Age Function.
- If you specify a PIVOT value, all two-digit years equal to or greater than the PIVOT value are placed in the 20th century (19xx). All two⌂digit years less than the PIVOT value are placed in the 21st century (20xx). The default PIVOT is **65**.

Date Formats

The source date format and the destination date format must contain a single valid date format and must be less than or equal to the length of the destination column. The format string must be delimited by single quotation marks.

Source Date Format –

Applies the source column format string to age character and numeric columns.
SF=*'format-string'*
SRCFMT=*'format-string'*

If the source column is character or numeric, you must use SRCFMT or a Source Exit Routine (SRCEXIT) to describe the contents of the column. These parameters are mutually exclusive.

Destination Date Format –

Applies the destination column format string to age character and numeric columns.
DF='format-string'
DSTFMT=*'format-string'*

If the destination column is character or numeric, you can specify DSTFMT or a Destination Exit Routine (DSTEXIT). If you do not specify a format for the destination, the date aging function uses SRCFMT by default. The destination column for an AGE function cannot be binary.

Use the following character strings to specify components of the date format:

Year	Month	Day	Time	Parts/Second
YYYY	MONTH	DDD	HH	FFFFFF
CCYY	MMM	DD	MI	FFFFF
YY	MM	D	SS	FFFF
	M			FFF
				FF
				F

- If you specify a question mark (**?**) in a format string, the Age Function maps the character value as it is. (Use the question mark to include slashes, dashes, periods, and so on, in the date format.)
- If you specify an asterisk (*****) in a format string, the Age Function maps any remaining characters in the source column to the destination column. (Use the asterisk when the column value is a date concatenated to additional characters.)

Note: You can use the Calendar Utility to define a default separator and a default output year. These defaults apply when the source and destination formats require separators or a specific year.

Example 1

To age a date column by 2 years, 6 months, 40 weeks, and 15 days, and then apply a rule, format the Age Function as:

```
AGE(+2Y,+6M,+40W,+15D,RU=NEXTPAYDAY)
```

Example 2

To age only the year portion in a date column to the year 2020, and apply a rule, format the Age Function as:

```
AGE(2020Y,RU=NEXTWORKDAY)
```

Example 3

To age a date column using MULTIPLE/RULE to increment by five occurrences of a rule called NEXTSTRTQTR, using a calendar called PSAPRULE, format the Age Function as:

```
AGE(CA=PSAPRULE,RU=NEXTSTRTQTR,5R)
```

Example 4

To age data in a character or numeric column by the following parameters:

- A named source column.
- The source format, using the first two characters for the last two digits of the year and the remaining 3 digits as the day in the Julian calendar.
- A century pivot to determine the correct century because the source is formatted with a two-digit year. The century pivot in this example is 42. All two-digit years greater than or equal to 42 are placed in the 20th century (19xx). All two-digit years less than 42 are placed in the 21st century (20xx).
- Age date by 5 years.

Format the Age Function as:

```
AGE(5Y,SC=ORDER_DATE,SF='YYDDD',PI=42)
```

Currency Function

Use the Currency Function to convert a currency value in a source column from one currency to another. The source column must be defined as numeric, but not floating point. Two conversion methods are available:

Direct Conversion

Provide conversion parameters based on values defined in a Currency Definition. Use the Currency Function to convert a monetary value in a column (replacing the prior value) or, by defining different source and destination columns, retain the original value and the converted value. You can explicitly define the source and destination currency types or you can identify a reference column to indicate the currency type.

The first calculation preference is to use the conversion rate for the source currency to the destination currency. The second calculation conversion preference is to use the conversion rate for the destination currency to the source.

Triangulation

Convert the value in a column from the source currency to the euro dollar and then convert the euro dollar to the destination currency. Both rates must be provided in the rate table: one between the euro dollar and the source currency, and one between the euro dollar and the destination currency. The specification expression is TRIANG or TR.

The Currency Function is formatted as follows:

```
CURRENCY( {ST=code | SS=(column-name,Types Table number)}
 {DT=code | DS=(column-name,Types Table number)}
 [SC=column-name] [TR] [CU=Currency Definition name]
 [TD=transaction-date-column-name] [DF='format']
 [NS=scale] )
```

The Currency Function must include at least a combination of the source currency type (ST) or source specification (SS) and the destination currency type (DT) or destination specification (DS). All other parameters are optional.

The source and destination currency types can be specified in one of two ways:

1. Use the ST/DT keywords to allow explicit specification of the currency using the three-character ISO 4217 Currency Code.
2. Use the SS/DS keywords to allow indirect specification of the currency type where a value in a named column in the row is used as a key. The key is correlated with a currency type as defined in the specified Currency Definition Type Table.

If you specify a transaction date (TD) and the transaction date column does not use the DATE format, you must also specify a date format (DF). If a specified transaction date is outside the date ranges specified in the Currency Definition Rates Table, the nearest date range is used for conversion calculations.

Note: If any required data (for example, currency types, rates) are missing, conversion errors will result at run time.

The following table describes the valid format and allowed values for the Currency Function parameters. Parameters can be specified in any order. Use commas or spaces to separate parameters in the Currency Function.

Parameter	Format
Source Column	SC=*column-name* *SRCCOL=column name*
Source Currency Type	ST=*code* *SRCTYP=code* *where code = ISO 4217 Currency Code*
Destination Currency Type	DT=*code* DSTTYP=*code* *where code = ISO 4217 Currency Code*
Source Specification	SS=*column name, Types Table number* *SRCSPC=column name, Types Table number* *where column name, Types Table number = Types Table column and number (defined in the Currency Definition) to be used to specify the source currency type*

Parameter	Format
Destination Specification	DS=*column name, Types Table number* *DSTSPC=column name, Types Table number* *where column name, Types Table number = Types Table column and number (defined in the Currency Definition) to be used to specify the destination currency type*
Triangulation (Forces conversion via the Euro dollar)	TR TRIANG
Currency Definition	CU=*Currency Definition name* *CURTBL=Currency Definition name* *where Currency Definition name = Currency Definition that contains the appropriate conversion parameters.*
Transaction Date	TD=*column name* TRNDAT=*column name* *where column name = Transaction Date column name to identify the conversion date*
Date Format	DF=*'format'* *DATFMT='format' where format = format of transaction date column, if not Date type.*
Numeric Scale	NS=*scale* *NUMSCL=scale* *where scale = scale to be applied to Oracle numeric destination columns with an undefined scale.*

Example 1

To convert from Finnish Markkas to Euro Dollars, format the Currency Function as:

```
CURRENCY(ST=FIM DT=EUR)
```

If the original value must be preserved, use the Currency Function to provide a value for a different column in the destination table.

Example 2

To convert from Finnish Markkas to Euro Dollars, and create a new column to retain the original source value (in Finnish Markkas) in a column labeled ITEM_COST, format the Currency Function as:

```
CURRENCY(ST=FIM DT=EUR SC=ITEM_COST)
```

Auto-Generated Email Name

The auto-generated email name function generates an email address with a user name based on a literal concatenated with a sequential number. The sequential numbers are suffixes that begin with 1 and are incremented by 1. The function uses the domain name from an email address in a specified source column.

Formatted Email Name

The formatted email name function generates an email address with a user name based on values obtained from one or two attributes. The policy uses the domain name from an email address in a specified source column.

Random Number Function

The random number function generates numbers selected at random within the range indicated by the low and high values.

You can use the random number function to replace character or numeric data. The low and high values must be integers within the range -2,147,483,648 to 2,147,483,647. The low value must be less than the high value.

Sequential Number Function

The sequential number function generates numbers that are incremented sequentially.

You can use the sequential number function to replace character or numeric data. You must enter a start value and a value by which the numbers are incremented. The start and incremental values must be integers within the range -2,147,483,648 to 2,147,483,647.

The generated value is limited by the data type and length of the destination column. If the generated value exceeds the length of the destination column, the function automatically resets to the start value.

Using Exit Routines

When you create a service to convert, create, insert, load, or restore data, you can specify a table map that includes one or more column maps to derive the appropriate values for destination columns.

There are several ways to specify a column value in a column map. One way is to specify an exit routine as the source; the exit sets values that could not otherwise be defined for destination columns. Another way is to use exit routines to exclude rows from processing. You can use three types of exit routines:

Standard Exit

The Standard Exit routine is called to derive the value for a destination column in a column map. This type of exit routine is useful when you want to perform data transformations that are beyond the scope of column maps. For example, an exit can change an employee department number for selected rows according to a complex algorithm, or select specific rows to be processed and discard all others.

A standard exit can get a substring segment of a source LOB column. To insert a new LOB value in a destination LOB column, an exit can create a file and pass the file name back to the column map processor.

Source Format Exit

The Source Format Exit is called to format the source column in an Age Function that would otherwise not be supported in a column map. This exit routine examines the source date in a character or integer column and converts it into a date format usable as input to the Age Function.

Destination Format Exit

The Destination Format Exit is called to format the destination column in an Age Function that would otherwise not be supported in a column map. This exit routine converts a date into one of four different destination formats. The data type of the destination column determines the format.

Exit in a column map

To use an exit routine in a column map, you must specify one of the following in the appropriate source column:

Standard Exit
EXIT *dllname[(parm[[,]parm]...)]*

Source Format Exit
AGE(SRCEXIT=*dllname*)

Destination Format Exit
AGE(DSTEXIT=*dllname*)

The process calls the column map exit routine, once for each data row processed and passes a termination call after the last row is processed. Optional parameters specified with a standard exit routine are passed to the exit, and must be string (enclosed in single quotes) or numeric literals (limit 8).

Writing Exit Routines

You can write exit routines in any programming language; however, calls to subroutines must conform to conventions used in the C programming language.

Header files

To define the parameters, structures, and return codes used in the exit routine, an exit routine must include two C program header files, PSTEXIT.H and, depending on the character format of the metadata (table names, column names, etc.), either PSTCMXIT.H or PSTCMWXT.H:

PSTEXIT.H
Specifies the data types, return codes, and structures for Optim-defined data types.

PSTCMXIT.H
For metadata in single-byte (e.g., ASCII) format.

Provides the prototypes for column map call⌂back routines and specifies the *defines* and *structure definitions* for the column map exit parameters.

PSTCMWXT.H
For metadata in UTF-16 (WCHAR) format.

Provides the prototypes for column map call⌂back routines and specifies the *defines* and *structure definitions* for the column map exit parameters.

These header files are located in the same directory with the IBM InfoSphere Optim application files.

Note: If a parameter has a single-byte form and a UTF-16 form, this chapter provides the UTF-16 form in parentheses.

Using DLLs

You must compile and link each exit routine as a separate DLL. Optim loads each DLL dynamically at run time. A DLL can contain only one occurrence of a particular type of column map exit. However, you can include one of each type of column map exit routine in the same DLL.

You must use the name of the DLL in the column map. Specify the DLL name as it is defined to the platform (that is, do not use the generic name). Write the DLL to name and export the actual functions that implement the exit routine:

Standard Exit
PSTColMapExit
(PSTColMapWExit)

Source Format Exit
PSTColMapAgeSrcExit
(PSTColMapAgeSrcWExit)

Destination Format Exit
PSTColMapAgeDstExit
(PSTColMapAgeDstWExit)

Requirements

Each column map exit routine must satisfy the following requirements:

- The exit must generate a value appropriate for the destination column and must not change any other storage areas.
- Optim uses the primary key to build internal work areas. If the exit routine assigns a value to a column that is part of the primary key, and that routine is called several times for the same source row, ensure that the exit routine generates the same output value every time.
- An exit may be called frequently; avoid unnecessary overhead. For example, the exit should initialize processing in the first call and save information for subsequent calls in the work area.

Standard Exit Routine

When you write a Standard Exit, you specify the exit function, PSTColMapExit (PSTColMapWExit), and the following parameters:

single-byte

```
PSTColMapExit
   (PST_STRUCT_CM_EXIT_PARM * pInputParms,
    PST_STRUCT_CM_EXIT_COL_LIST * pSrcColList,
     PST_STRUCT_CM_EXIT_COL_LIST * pDstColList)
```

UTF-16

```
PSTColMapWExit
   (PST_STRUCT_CM_WEXIT_PARM * pInputParms,
    PST_STRUCT_CM_WEXIT_COL_LIST * pSrcColList,
     PST_STRUCT_CM_WEXIT_COL_LIST * pDstColList)
```

Parameters

When a process calls a standard exit routine, the process passes the following parameters on every call:

pInputParms
Pointer to PST_STRUCT_CM_EXIT_PARM (PST_STRUCT_CM_WEXIT_PARM). This structure contains information about the source and destination tables, the nature of the current call, number of optional parameters specified with the exit, and pointers to the call⌂back functions, work areas, and optional parameters.

The first field in this structure is the **FuncCode** field identified by either PST_CM_FUNC_TRANSFORM (PST_CMW_FUNC_TRANSFORM) or PST_CM_FUNC_TERMINATE (PST_CMW_FUNC_TERMINATE).

The field *NumParms* contains the number of optional parameters specified with the column map exit (0 to 8).

The field *pParm* contains an array of pointers to each optional parameter specified with the column map exit.

pScrColList
Pointer to PST_STRUCT_CM_EXIT_COL_LIST (PST_STRUCT_CM_WEXIT_COL_LIST). This structure describes the source columns.

pDstColList
Pointer to PST_STRUCT_CM_EXIT_COL_LIST (PST_STRUCT_CM_WEXIT_COL_LIST). This structure describes the destination columns.

Call-back functions

Each time an Optim process calls a standard exit routine, the process passes the addresses of the following call-back functions:

pPSTGetColValue( )
Retrieves data for all source columns and most destination columns in the current data row. In general, the exit routine calls this function only once to retrieve the data for a source column. However, the exit routine can call this function several times to retrieve the data for different columns.

pPSTPutColValue( )
Specifies data for the destination column in the current data row. The exit routine determines the value for the destination column and returns the value to Optim. This function must be called unless the row is rejected or the process is aborted.

Processing

Typical processing for the Standard Exit routine is summarized in the following steps:

1. On every call from Opitm, the exit routine checks for a first time call. On the first call, the exit performs any initialization tasks and normal processing (step 2). On subsequent calls only normal processing is performed (step 2).
2. Optim does not pass column data to the standard exit routine; however, the exit routine can make calls to the **pPSTGetColValue()** call-back function to obtain data for the source columns. These values are needed to determine the value of the destination column.
3. After the exit routine generates the destination value, the exit either calls the **pPSTPutColValue( )** call-back function to store the value in the destination column or passes an appropriate return code instructing the process to skip the data row or abort.
4. After the last data row is processed, the Optim passes a termination call to the exit routine, identified by a value of PST_CM_FUNC_TERMINATE (PST_CMW_FUNC_TERMINATE) in the FuncCode field. This call prompts the exit routine to free any dynamically allocated storage. When final tasks are complete, the exit routine passes a return code to Optim.

Return codes

The following return codes apply to standard exit routines:

```
PST_CM_EXIT_SUCCESS
(PST_CMW_EXIT_SUCCESS)
```

Destination column is assigned a value or is successfully transformed.

```
PST_CM_EXIT_REJECT_ROW
(PST_CMW_EXIT_REJECT_ROW)
```

Destination column cannot be assigned a value or transformed. Discard the row.

```
PST_CM_EXIT_ABORT_PROCESS
(PST_CMW_EXIT_ABORT_PROCESS)
```

Fatal error. Terminate processing. To return an error message, place the message in the work area and set the unused space to blanks or NULL.

Source Format Exit

When you write a Source Format Exit routine for Optim, you specify the exit function, PSTColMapAgeSrcExit (PSTColMapAgeSrcWExit), and the following parameters:

single-byte

```
PSTColMapAgeSrcExit
   (PST_STRUCT_CM_AGE_SRCFMT_PARM * pInputParms,
   PST_STRUCT_CM_EXIT_COL_LIST * pSrcColList,
   PST_STRUCT_CM_EXIT_COL_LIST * pDstColList)
```

UTF-16

```
PSTColMapAgeSrcWExit
   (PST_STRUCT_CM_AGE_SRCFMT_WPARM *
pInputParms,
   PST_STRUCT_CM_WEXIT_COL_LIST * pSrcColList,
   PST_STRUCT_CM_WEXIT_COL_LIST * pDstColList)
```

Parameters

When an Optim process calls a source format exit routine, the process passes the following parameters:

pInputParms
Pointer to PST_STRUCT_CM_AGE_SRCFMT_PARM (PST_STRUCT_CM_AGE_SRCFMT_WPARM). This structure contains information about the source and destination tables, the nature of the current call, and pointers to a call-back function and work areas.

The first field in this structure is the **FuncCode** field identified by either PST_CM_SRCFMT_TRANSFORM (PST_CMW_SRCFMT_TRANSFORM) or PST_CM_SRCFMT_TERMINATE (PST_CMW_SRCFMT_TERMINATE).

pSrcColList
Pointer to PST_STRUCT_CM_EXIT_COL_LIST (PST_STRUCT_CM_WEXIT_COL_LIST). This structure describes the source columns.

pDstColList
Pointer to PST_STRUCT_CM_EXIT_COL_LIST (PST_STRUCT_CM_WEXIT_COL_LIST). This structure describes the destination columns.

Call-Back Function

Each time an Optim process calls a source format exit routine, the process passes the address of the following call-back function:

pPSTGetColValue( )
Retrieves data for all source columns and most destination columns in the current data row. In general, the exit routine does not need to call this function because the data for the source column is provided in the first parameter. However, the exit routine can call this function to retrieve the data for different columns.

Processing

Typical processing for the Source Format Exit routine is summarized in the following steps:

1. On every call from Optim, the exit routine checks for a first time call. On the first call, the exit performs any initialization tasks and normal processing (step 2). On subsequent calls only normal processing is performed (step 2).
2. The exit receives the value of the source column as specified in the Age Function defined in the column map. Optim does not apply the Age Function before calling the exit routine and stores the raw value in one of the **InputValue** union fields as indicated by the **ValueType** field in the header file.
3. If the exit needs to examine other columns to calculate the value for the destination column, the exit must call the **pPSTGetColValue( )** call-back function to obtain the value for those columns.
4. After the destination value is generated, the exit routine must format the value and place it either in the **OutputTimeStamp** field or **OutputSybDateTime** field. Optim validates this value and applies the Age

Function. The exit must pass an appropriate return code indicating the field where the data is saved or instructing the process to skip the data row or abort.

5. After the last data row is processed, Optim passes a termination call to the exit routine, identified by a value of PST_CM_SRCFMT_TERMINATE (PST_CMW_SRCFMT_TERMINATE) in the **FuncCode** field. This call prompts the exit routine to free any dynamically allocated storage. When final tasks are complete, the exit routine passes a return code to Optim.

Abort modes

There are several ways that the Source Format Exit routine can abort processing:

- Process rows with skipped dates or invalid dates. If you select either of these options in a process request, and the source and destination columns have the same attributes, the source column is copied unchanged to the destination column. If you do not select either option, the row is rejected.
- Reject the row. Reject the row regardless of the process options for skipped or invalid dates, based on specifications in the exit routine.
- Abort the whole process, based on specifications in the exit routine.

The exit routine passes a return code indicating which date format or which abort mode to use.

During a process, the exit routine may interrogate any columns from the input row and some of the columns from the destination row. However, the exit routine cannot interrogate a destination column that includes an exit routine and is defined in the column map after the current destination column. All other destination columns are available.

Return codes

The following return codes apply to source format exits:

```
PST_CM_SRCFMT_USE_TIMESTAMP
(PST_CMW_SRCFMT_USE_TIMESTAMP)
```

Destination column is assigned a value in the **OutputTimeStamp** field of the first parameter passed to the exit.

```
PST_CM_SRCFMT_USE_SYB_DATETIME
(PST_CMW_SRCFMT_USE_SYB_DATETIME)
```

Destination column is assigned a value in the **OutputSybDateTime** field of the first parameter passed to the exit.

```
PST_CM_SRCFMT_SKIP
(PST_CMW_SRCFMT_SKIP)
```

Aging is not applied. If you do not select the option to **Process rows with skipped dates**, the row is rejected. Otherwise, the data is copied to the source, as long as the source and destination are compatible. If not compatible, the row is rejected.

```
PST_CM_SRCFMT_COL_INVALID
(PST_CMW_SRCFMT_COL_INVALID)
```

Aging is not applied. If you do not select the option to Process rows with invalid dates, the row is rejected. Otherwise, the data is copied to the source, as long as the source and destination are compatible. If not compatible, the row is rejected.

```
PST_CM_SRCFMT_REJECT_ROW
(PST_CMW_SRCFMT_REJECT_ROW)
```

Source column cannot be assigned a value. Reject (discard) the row.

```
PST_CM_SRCFMT_ABORT_PROCESS
(PST_CMW_SRCFMT_ABORT_PROCESS)
```

Fatal error. Terminate. To return an error message, place the message in the work area and set the unused space to blanks or NULL.

Destination Format Exit

When you write a Destination Format Exit routine for Optim, you specify the exit function, PSTColMapAgeDstExit (PSTColMapAgeDstWExit), and the following parameters:

single-byte

```
PSTColMapAgeDstExit
   (PST_STRUCT_CM_AGE_DSTFMT_PARM * pInputParms,
   PST_STRUCT_CM_EXIT_COL_LIST * pSrcColList,
   PST_STRUCT_CM_EXIT_COL_LIST * pDstColList)
```

UTF-16

```
PSTColMapAgeDstWExit
   (PST_STRUCT_CM_AGE_DSTFMT_WPARM * pInputParms,
   PST_STRUCT_CM_WEXIT_COL_LIST * pSrcColList,
   PST_STRUCT_CM_WEXIT_COL_LIST * pDstColList)
```

Parameters

When an Optim process calls a destination format exit routine, the process passes the following parameters:

pInputParms

Pointer to PST_STRUCT_CM_AGE_DSTFMT_PARM (PST_STRUCT_CM_AGE_DSTFMT_WPARM). This structure contains information about the source and destination tables, the nature of the current call, and pointers to a call-back function and work areas.

The first field in this structure is the **FuncCode** field, identified by:

PST_CM_DSTFMT_TO_CHAR (PST_CMW_DSTFMT_TO_WCHAR), PST_CM_DSTFMT_TO_INTEGER (PST_CMW_DSTFMT_TO_INTEGER), PST_CM_DSTFMT_TO_TIMESTAMP (PST_CMW_DSTFMT_TO_TIMESTAMP), PST_CM_DSTFMT_TO_SYB_DATETIME (PST_CMW_DSTFMT_TO_SYB_DATETIME), or PST_CM_DSTFMT_TERMINATE (PST_CMW_DSTFMT_TERMINATE).

pSrcColList

Pointer to a PST_STRUCT_CM_EXIT_COL_LIST (PST_STRUCT_CM_WEXIT_COL_LIST). This structure describes the source columns.

pDstColList

Pointer to a PST_STRUCT_CM_EXIT_COL_LIST (PST_STRUCT_CM_WEXIT_COL_LIST). This structure describes the destination columns.

Call-back function

Each time an Optim process calls a destination format exit routine, the process passes the address of the following call-back function:

pPSTGetColValue( )

Retrieves data for all source columns and most destination columns in the current data row. In general, the exit routine does not need to call this function because the data for the aged source column is provided in the first parameter. However, the exit routine can call this function to retrieve the data for different columns.

Formats

The input date is in both a PST_C_TIMESTAMP and PST_C_SYB_DATETIME format. The exit is directed to transform that date into one of the following formats, based on the data type of the destination column.

PST_C_CHAR_SZ
CHAR and VARCHAR destination columns.

PST_C_INTEGER_CHAR_SZ
NUMERIC destination columns.

PST_C_TIMESTAMP
DB2 and Oracle DATE/TIME columns.

PST_C_SYB_DATETIME
Sybase ASE DATETIME columns.

Processing

Typical processing for the Destination Format Exit routine is summarized in the following steps:

1. On every call from Optim, the exit routine checks for a first time call. On the first call, the exit performs any initialization tasks and normal processing (step 2). On subsequent calls only normal processing is performed (step 2).
2. The exit receives the value of the source column as specified in the Age Function defined in the column map. Optim applies the Age Function before calling the column map exit and stores the aged value in both the **InputTimeStamp** and **InputSybaDateTime** fields in the header file.
3. If the exit needs to examine other columns to calculate the value for the destination column, the exit must call the **pPSTGetColValue( )** call-back function to obtain the value for those columns.
4. After the destination value is generated, the exit routine must format the value and place it in one of the fields in the **OutputValue** union. The **FuncCode** field indicates the field in the **OutputValue** union where the value must be placed. The exit must return an appropriate code indicating the field where the data is saved or instructing the process to skip the row or abort.
5. After the last data row is processed, Optim passes a termination call to the exit routine, identified by a value of PST_CM_DSTFMT_TERMINATE (PST_CMW_DSTFMT_TERMINATE) in the **FuncCode** field. This call prompts the exit routine to free any dynamically allocated storage. When final tasks are complete, the exit routine passes a return code to Optim.

Abort Modes

There are several ways that the Destination Format Exit routine can abort processing:

- Process rows with skipped dates or invalid dates. If you select either of these options in a process request, and the source and destination columns have the same attributes, the source column is copied unchanged to the destination column. If you do not select either option, the row is rejected.
- Reject the row. Reject the row regardless of the process options for skipped or invalid dates, based on specifications in the exit routine.
- Abort the whole process, based on specifications in the exit routine.

The exit routine passes a return code indicating whether the conversion was successful or which abort mode to use.

During a process, the exit routine may interrogate any columns from the input row and some of the columns from the destination row. However, the exit routine cannot interrogate a destination column that includes an exit routine and is defined in the column map after the current destination column. All other destination columns are available.

Return codes

The following return codes apply to destination format exits:

`PST_CM_DSTFMT_SUCCESS`
`(PST_CMW_DSTFMT_SUCCESS)`

Destination column is assigned a value, as specified in the **FuncCode** field of the first parameter passed to the exit.

`PST_CM_DSTFMT_SKIP`
`(PST_CMW_DSTFMT_SKIP)`

If you do not select the option to **Process rows with skipped dates**, the row is rejected. Otherwise, the source data is copied to the target, as long as the source and target are compatible. If not compatible, the row is rejected.

`PST_CM_DSTFMT_COL_INVALID`
`(PST_CMW_DSTFMT_COL_INVALID)`

If you do not select the option to **Process rows with invalid dates**, the row is rejected. Otherwise, the data is copied to the source, as long as the source and target are compatible. If not compatible, the row is rejected.

`PST_CM_DSTFMT_REJECT_ROW`
`(PST_CMW_DSTFMT_REJECT_ROW)`

Destination column cannot be assigned a value. Reject (discard) the row.

`PST_CM_DSTFMT_ABORT_PROCESS`
`(PST_CMW_DSTFMT_ABORT_PROCESS)`

Fatal error. Terminate. To return an error message, place the message in the work area and set the unused space to blanks or NULL.

Writing column map procedures with Lua scripting

A column map procedure is a procedure that is used to mask or transform the data in a column when you run a service. As the name indicates, you must add column map procedures to a column map. You can write column map procedures by using the Lua scripting language.

For more information about the Lua programming language, see the Lua web site at http://www.lua.org

Lua functions for column map procedures

Column map procedures support most standard Lua functions. Column map procedures also support functions that are specific to Optim.

Standard function names for user-defined functions

Use the following function names in your Lua column map procedures. Each of these functions is called automatically at the point indicated. You can also create functions with other function names and call these functions from the standard functions.

Name	Description	Required
`cm_load()`	This function is called before any tables are processed.	No
`cm_unload()`	This function is called after all tables are processed.	No

Name	Description	Required
`cm_starttable()`	This function is called at the start of processing for each table.	No
`cm_endtable()`	This function is called at the end of processing for each table.	No
`cm_transform()`	This function is called for every row processed.	Yes

Every column map procedure must contain a `cm_transform()` function. If the `cm_transform()` function does not exist, the code is evaluated as a complex expression.

Global functions

The following functions are available in all column map procedure execution contexts:

Name	Description
`numparms()`	Get the number of parameters that are passed to the column map procedure.
`parms.get(n)`	Get the value of the parameter at index *n*.
`print()`	Print messages to process report.
`rejectrow()`	Skip row and go to the next row.

Data store functions

Use the following functions to get information about the source and target data stores:

Name	Description
`source.getdbalias()`	Get the DB alias of the source data store.
`source.getcreatorid()`	Get the creator ID of the source data store.
`target.getdbalias()`	Get the DB alias of the target data store.
`target.getcreatorid()`	Get the creator ID of the target data store.

Table functions

Use the following functions to get information about the source and target tables:

Name	Description
`source.gettablename()`	Get the name of the source table.
`target.gettablename()`	Get the name of the target table.

Column functions

Use the following functions to get information about the source and target columns, transform column data, write the result to the target column, and exit:

Name	Description
source.column.getvalue()	Get a nonnumeric value from a source column. A runtime error is generated if you use the source.column.getvalue() function to get values from columns of data type PST_SQL_BINARY.
source.column.getasdouble()	Get a numeric value from a source column (column map procedures process numeric data in double-precision format). Use this function for columns of the following data types: PST_SQL_DECIMAL, PST_SQL_DOUBLE, PST_SQL_FLOAT, PST_SQL_INFX_DEC_FLOAT, and PST_SQL_ORA_NUMBER.
source.column.getlength()	Get the length of the string in a source column.
source.column.getname()	Get the name of the source column.
source.column.gettype()	Get the data type of the source column.
target.column.setvalue()	Set the value of a target column. A runtime error is generated if you use the target.column.setvalue() function to set values in columns of data type PST_SQL_BINARY.
target.column.getlength()	Get the length of the string in a target column.
target.column.getname()	Get the name of the target column.
target.column.gettype()	Get the data type of the target column.
optimmask()	Call an Optim Data Privacy Providers (ODPP) provider.
userexit()	Call a user exit.

Unsupported functions

The following categories of functions are not supported in column map procedures.

- Input and output facilities that are built into the Lua io library
- string.dump()

Limitations of column map procedures

Bear in mind the following limitations when you write a column map procedure.

Numeric data processed in double-precision format

Column map procedures process numeric data in double-precision format. You must therefore use the source.column.getasdouble() function to get numeric data from source columns.

Encoding

Column map procedures use UTF-16 encoding in their internal processing.

PST_SQL_BINARY data type not supported

A runtime error is generated if you use the source.column.getvalue() function to get values from columns of data type PST_SQL_BINARY. A runtime error is also generated if you use the target.column.setvalue() function to set values in columns of data type PST_SQL_BINARY.

Unsupported functions

The following categories of functions are not supported in column map procedures.

- Input and output facilities that are built into the Lua `io` library
- `string.dump()`

Column map procedure example: Generic procedure

The column map procedure example illustrates the structure of a column map procedure with its standard functions: `cm_load`, `cm_unload`, `cm_starttable`, `cm_endtable`, and `cm_transform`.

```
--------------------------------------------------------------------------------
--
--   IBM Optim sample column map procedure
--
--   Name:          OptimSample
--
--   Revision:      1.0
--
--   Description:   Demonstrates all capabilities of Optim/Lua column map procedures
--
--   Input:         Zero or more parameters, which will simply be echoed back to the
--                  Optim process report
--
--
--   Output:        Section in Optim process report showing information from this
--                  column map procedure.  The column itself is left unchanged.
--
--
--------------------------------------------------------------------------------


--------------------------------------------------------------------------------
--  cm_load function - Called before any tables are processed
--------------------------------------------------------------------------------
function cm_load()

  print("  *** Start of Process ***")

  colinfoshown = false

  -- Display parameters passed from Column Map

  print("  Argument Count: " .. string.format("%d", numparms()))
    for i = 1, numparms(), 1 do
      print("    Argument " .. string.format("%d", i-1) .. " " .. parms.get(i-1))
  end

end

--------------------------------------------------------------------------------
--  cm_unload function - Called after all tables are processed
--------------------------------------------------------------------------------
function cm_unload()

  print("  *** End of Process ***")

end

--------------------------------------------------------------------------------
--  cm_starttable function - Called at the start of processing for each table
--------------------------------------------------------------------------------
function cm_starttable()

  print("  \nStart of processing table")
  fullname = source.getdbalias() .. "." .. source.getcreatorid() .. "." .. source.gettablename()
```

```
  print("      Source Table: " .. fullname)
  fullname = target.getdbalias() .. "." .. target.getcreatorid() .. "." .. target.gettablename()
  print("      Target Table: " .. fullname)

end

-------------------------------------------------------------------------------
--  cm_endtable function - Called at the end of processing for each table
-------------------------------------------------------------------------------
function cm_endtable()

  print("  \nEnd of processing table")

end

-------------------------------------------------------------------------------
--  cm_transform function - Called for each row processed
-------------------------------------------------------------------------------
function cm_transform()


  if (not colinfoshown) then
     colinfoshown = true
     print("  Processing column " .. source.column.getname())
     print("     Type: " ..  source.column.gettype())
     print("     Length: " .. source.column.getlength())
  end

  -- This statement gets the value in the column for which cm_transform was called
  -- Optionally, the name of another column can be specified, for example:
  -- source.column.getvalue("COL1") will return the value in column COL1
  oldvalue = source.column.getvalue()

  -- This code sets that target column to the same value as the source column
  -- Logic to change the value would be placed here.
  -- If you wish to NOT insert this row into the target table, call the rejectrow() function
    newvalue = oldvalue
    target.column.setvalue(newvalue)
end
```

Column map procedure example: Switched lookup

The column map procedure example illustrates how the mask_parms function masks a column by using data from a lookup table.

```
-------------------------------------------------------------------------------
--
--   IBM Optim sample column map procedure
--
--   Name:          OptimSwitchedLookup
--
--   Revision:      1.0
--
--   Description:   Masks a column by using table lookup.  The lookup table to
--                  be used is determined by the value of another column.
--
--   Input:         Parameter 1 (Required):
--                  A string that indicates the type of lookup to use:
--                  HASH, RANDOM, or LOOKUP
--
--                  Parameter 2 .... n-1 (Required)
--                  An expression that indicates the lookup table to use. The format is
--                  COND(column-name=value) DATASOURCE(datasource_parameters)
--                  This parameter can be repeated multiple times.  If a row does not
--                  satisty any of the COND parameters than it will not be inserted
--                  into the target table.
--
```

```
--                  Parameter n (Optional):
--                  A string containing additional parameters to be
--                  copied into the optimmask invocation.
--                  This is in addition to the datasource_parameters value in
--                  the COND clause of Parameter 2 and the mask_parms_constant field
--                  that is declared at the start of this script.
--
--
--
--   Output:       - The masked column data as set by the target.column.setvalue function
--                 - Text directed to the Optim report by the print function
--
--   Return Codes: 0 - Successful execution
--                 1 - Reject row (Use in cm_transform; row will not be inserted to
--                     destination table)
--                 2 - Abort process
--
--                 Two helper functions are specified to specify conditions other than
--                 return code 0.
--                 There is no need to code a return statement when using these
--                 functions.
--                 error(string) - This call causes the Optim process to abort
--                                 and string is shown in the Optim report
--                                 as an error message.
--                 rejectrow()   - This call causes Optim to reject the row
--                                 currently being processed. The row is
--                                 not inserted into the destination table.
--
--
--
-------------------------------------------------------------------------------

function cm_transform()

    -- Change this field to contain parameters that should
    -- be placed into all optimmask calls
    local mask_parms_constant = 'CACHE=Y,WHENNF=PRE'

    --
    -- Validate number of parameters:
    --
    nparm = numparms()
    if (nparm  < 2) then
       process_error("Call to column map procedure OptimSwitchedLookup must have 2 or more parameters")
    end

    --
    -- Process Parameter 1 (lookup type)
    parm = parms.get(0)
    if (string.lower(parm) == "hash") then
      provalue = "HASH_LOOKUP"
    elseif (string.lower(parm) == "random") then
      provalue = "RANDOM_LOOKUP"
    elseif (string.lower(parm) == "lookup") then
      provalue = "LOOKUP"
    else
      process_error("Invalid parameter.  Expected HASH, RANDOM or LOOKUP.  Found " .. parm)
    end


    --
    -- Process COND/DATASOURCE parameters
    --
    gotcond = false
    for parmptr = 1, nparm-1 do
      parm = parms.get(parmptr)
      if (string.lower(string.sub(parm, 1, 5)) == "cond(") then
```

```
        gotcond = true
        datasource_parameters = process_cond()
        if (datasource_parameters > "") then
          break;
        end
      end
    end
    if (not gotcond) then
      process_error("No COND parameter found")
    end

    -- No COND matched this row, so reject the row
    if (datasource_parameters <= "") then
      rejectrow()
    end

    --
    -- Process optional additional optimmask parameter
    --
    lastparm = parms.get(nparm-1)
    if (string.lower(string.sub(lastparm, 1, 5)) == "cond(") then
       optimmask_additional_parms = ""
    else
       optimmask_additional_parms = lastparm
    end

    --
    -- Construct call to optimmask, make the call,
    -- and place new value into target column
    --
    mask_parms = "PRO=" .. provalue .. "," .. mask_parms_constant

    -- This use of environment variables to store userid and password
    -- for the system table is for simple illustrative purposes only.
    -- For greater security, store this information in environment variables
    -- in an encrypted format.
    userid = os.getenv("optimmaskuserid")
    if (userid) then
      mask_parms = mask_parms .. ",USER=" .. userid
    end
    password = os.getenv("optimmaskpswd")
    if (password) then
      mask_parms = mask_parms .. ",PASS=" .. password
    end

    mask_parms = mask_parms .. "," .. optimmask_additional_parms

    oldvalue = source.column.getvalue()
        newvalue = optimmask(oldvalue, mask_parms)
        target.column.setvalue(newvalue)
end



function process_cond()

  strptr = 6     -- Point to first character after "COND("

  -- Get the column name
  equalsign = string.find(parm, "=", strptr, true)
  if (not equalsign) then
    process_error("Syntax error around character " .. strptr .. " in expression: " .. parm)
  end
  colname = string.sub(parm, strptr, equalsign-1)

  -- Get the column value
  strptr = equalsign + 1
```

```
  closeparen = string.find(parm, ")", strptr, true)
  if (not closeparen) then
    process_error("COND expression is missing closing parenthesis in expression: " .. parm)
  end
  colvalue = string.sub(parm, strptr, closeparen-1)

  --Debug - Print scan results:
  --print ("Found colname=" .. colname .. " in parm: " .. parm)
  --print ("Found colvalue=" .. colvalue .. " in parm: " .. parm)

  -- If COND(column-name=value) condition not met for this row, then
  -- do no further processing on this parameter.  Return "" to indicate no hit.
  if (source.column.getvalue(colname) ~= colvalue) then
    return ""
  end

  -- Got a match on COND, so get the DATASOURCE value
  strptr = closeparen + 1
  datasourceValuePtr = string.find(parm, "DATASOURCE(", strptr, true)
  if (not datasourceValuePtr) then
    process_error("DATASOURCE clause not found in expression: " .. parm)
  end
  -- (Note: This scanning is very simple; it assumes that there
  -- is no errant text between the COND and DATASOURCE clauses)
  closeparen = string.find(parm, ")", datasourceValuePtr, true)
  if (not closeparen) then
    process_error("COND expression is missing closing parenthesis in expression: " .. parm)
  end
  return string.sub(parm, datasourceValuePtr+11, closeparen-1)

end
function process_error(msg)
  errprefix = "Error in column map procedure for column " .. source.column.getname()
  errprefix = errprefix .. " in table " .. source.getdbalias() .. "."
     .. source.getcreatorid() .. "." .. source.gettablename()
  error(errprefix .. ":\n   " .. msg)
end
```

Column map procedure example: National ID masking

The column map procedure example illustrates how the mask_parms function masks a column that contains a national ID.

```
-------------------------------------------------------------------------------
--
--   IBM Optim sample column map procedure
--
--   Name:          OptimNID
--
--   Revision:      1.0
--
--   Description:   Masks a national ID column based on the value of another
--                  column or a script argument with the country code.
--
--   Input:         Parameter 1 (Required):
--                  A string country code (eg. US for United States)
--                  --OR--
--                  COL(column-name) where column-name is the name of a column
--                  that contains the country code
--
--                  Parameter 2 (Optional):
--                  A string containing additional parameters to be copied into the
--                  optimmask invocation
--                  This is in addition to the mask_parms_constant field that is
--                  delcared at the start of this script.
--
--
```

```
--
--   Output:       - The masked column data as set by the target.column.setvalue function
--                 - Text directed to the Optim report by the display function
--
--   Return Codes: 0 - Successful execution
--                 1 - Reject row (Use in cm_transform;  row will not be inserted to
--                     destination table)
--                 2 - Abort process
--
--                 Two helper functions are specified to specify conditions other than
--                 return code 0.
--                 There is no need to code a return statement when using these
--                 functions.
--                 error(string) - This call will cause the Optim process to abort
--                                 and string will be shown in the Optim report
--                                 as an error message.
--                 rejectrow()   - This call will cause Optim to reject the row
--                                 currently being processed The row will
--                                 not be inserted into the destination table.-
--
--
-------------------------------------------------------------------------------

function cm_transform()

    -- Change this field to contain parameters that should
    -- be placed into all optimmask calls
    local mask_parms_constant = 'MTD=REPEATABLE'

    --
    -- Obtain parameters:
    --
    nparm = numparms()
    if nparm  ~= 1 and nparm ~= 2 then
       msg = "Call to column map procedure for column "
                   msg = msg ..  source.column.getname()
       msg = msg .. " must have 1 or 2 parameters"
       error(msg)
    end
    parm1 = parms.get(0)
    if (nparm == 2) then
       parm2 = parms.get(1)
    end

    --
    -- Get country code into swivalue field based on parameters
    --
    if string.lower(string.sub(parm1, 1, 4)) == 'col(' then
       closeparen = string.find(parm1, ')', 5, true)
       if (not closeparen) then
           error("No closing parenthesis found in expression: " .. parm1)
       end
       colname = string.sub(parm1, 5, closeparen-1)
       swivalue = source.column.getvalue(colname)
    else
       swivalue = parm1
    end

    --
    -- Construct call to optimmask, make the call,
               -- and place new value into target column
    --
    mask_parms = 'PRO=NID, SWI=' .. swivalue .. ', ' .. mask_parms_constant
    if (parm2) then
       mask_parms = mask_parms .. ', ' .. parm2
    end
```

```
    oldvalue = source.column.getvalue()
        newvalue = optimmask(oldvalue, mask_parms)
        target.column.setvalue(newvalue)
end
```

Chapter 7. Managing data

Use the Optim Designer utilities to edit, browse, and compare data. You can also create tables.

Browsing data

Use the Browse utility to review the contents of a file data store without having to restore it to the database.

Use the help facility in the Browse utility to obtain more information about a specific topic or function. You can open a help window by selecting **Help** > **Contents**, right-clicking an item and selecting **What's This**, or by pressing F1.

To browse data:

1. From Optim Designer, click **Tools** > **Browse** The Browse windows opens.
2. Click **File** > **Open** to open the Open window and select a data file created by a service. The files are stored in the data directory specified in the Configuration utility.
3. Click **Open**. The tables in the data file are listed in the Browse window.
4. Click **Tools** > **Create All Selected Objects**. The table opens in the Browse Data window.

Editing data

Use the Table Editor to browse and edit sets of relationally intact data in database tables. The editor handles an arbitrarily complex data model consisting of any number of tables and relationships and ensures a referentially intact set of data.

Use the help facility in the Table Editor to obtain more information about a specific topic or function. You can open a help window by selecting **Help** > **Contents**, right-clicking an item and selecting **What's This**, or by pressing F1.

To edit data:

1. From Optim Designer, click **Tools** > **Edit** The Edit Options windows opens.
2. Enter a **Table Name** or **Named Access Definition**, select **Initial Display** or **Mode** options, and click **OK**. The Table Editor window opens.
3. Edit one or more tables at the same time.

   Commit changes to the database by moving the pointer to a different row or by using menu options. Each instance of a commit counts as an undo level. An undo level is defined as a change to a row that is committed to the database.

   If an error condition results when you attempt to commit data to the database, the data is not committed, but the attempt still counts as an undo level. The editor allows you to restore the data you modify to a specific commit point.

   A fetch set is the set of rows the editor reads from a single table in the database.

   - To undo changes and restore rows to the original versions obtained in the current fetch set for all tables, select **Tools** > **Undo**.
   - To undo a change for a row, right-click the row and click **Undo**.
   - To display each successive version of the row committed in the current fetch set, and the original version of the row in the fetch set, right-click the row and click **Undo...**.

Comparing data

Use the Compare utility to compare data from one set of source tables with data from another. You can define a compare request that is stored in the Optim repository and save the results of a comparison in a compare file.

Defining a compare request

Use the Compare Request Editor to define data sources to compare and processing options.

Use the help facility in the Compare utility to obtain more information about a specific topic or function. You can open a help window by selecting **Help** > **Contents**, right-clicking an item and selecting **What's This**, or by pressing F1.

To define a compare request:

1. In the Repository Explorer, right-click **Compare** and click **New Compare**. The Compare Request Editor opens.
2. In the **General** tab, complete the following steps:
   a. Enter a compare file name in the **Compare File** field.
   b. Select **Browse Results Immediately** to view the results of the compare process after the process is competed.
   c. Select a **Comparison** mode.

      Single Table

      Source File – Source File
      Compare data in a single table in one source file with data in another source file.

      Source File – Database Table
      Compare data in a single table in a source file with data in a single table in the database.

      Database Table – Database Table
      Compare data in a single table in one database with data in a single table in another database.

      Multiple Tables

      Source File – Source File
      Compare the data in tables in one source file with data in another source file.

      Source File – Access Definition
      Compare the data in tables in a source file with the data specified in an access definition.

      Source File – All Database Tables
      Compare the data in tables in a source file with data in a database.

      Note: This selection is extremely useful when testing a database application. The source file represents the "before" image compared with the database tables, which represent the "after" image.

      Access Definition – Access Definition
      Compare the data specified in one access definition with data specified in another access definition.

      Access Definition – All Database Tables
      Compare the data specified in an access definition with data in a database.
3. In the **Source** tab, select the data sources to compare based on the comparison mode.
4. Click **File** > **Run** to begin the compare process.

5. Click **File** > **Save**to save the compare request. Enter a request name in the format IDENTIFIER.NAME.

The process will create a compare file in the data directory specified in the Configuration utility. Use the Browse utility to view the results of the comparison.

Editing a compare request

Use the Compare Request Editor to edit a compare request.

Use the help facility in the Compare utility to obtain more information about a specific topic or function. You can open a help window by selecting **Help** > **Contents**, right-clicking an item and selecting **What's This**, or by pressing F1.

To edit a compare request:

1. In the Repository Explorer, expand the **Compare** node, right-click the request to edit and click **Edit**. The Compare Request Editor opens.
2. Edit the request.
3. Click **File** > **Save**to save the compare request.

Running a compare process

You can run a compare process to generate a compare file that you can view with the Browse utility.

To run a compare request:

Run the request by using one of the following methods:
To run the compare process from the Repository Explorer, right-click a compare request and click **Run**.
To run the compare process from the Compare Request Editor, click **File** > **Run**.

The process will create a compare file in the data directory specified in the Configuration utility. Use the Browse utility to view the results of the comparison.

Creating tables

Use the Create utility to create objects based on definitions in a source data file.

Use the help facility in the Create utility to obtain more information about a specific topic or function. You can open a help window by selecting **Help** > **Contents**, right-clicking an item and selecting **What's This**, or pressing F1.

To create tables:

1. From Optim Designer, click **Tools** > **Create** The Create Options windows opens.
2. Select a source data file created by a service. The files are stored in the data directory specified in the Configuration utility.
3. Select one of the **Table Map Options**.

   If you choose a Local table map, the Local - Table Map Editor opens. Complete the following steps:

   a. In the **Qualifier** field, you must enter a target data store alias and creator ID in the format `data_store_alias.creator_id`.
   b. Modify the destination table names.
   c. Click **File** > **Update and Return**.

   If you select a named table map, complete the following step:

   a. In the **Table Map Name** field, enter or select a table map name.

4. Click **OK**. The Create window opens.
5. Edit and select the objects to create.
6. Click **Tools** > **Create All Selected Objects** to begin the create process.

Chapter 8. Managing Data with Optim Designer

Optim Designer provides a single design interface for Optim data privacy and test t data management solutions.

Extracting data with Optim Designer

This tutorial teaches you how to use Optim Designer to create an Optim extract service. In this tutorial, you will define an access definition and an extract service.

After completing this tutorial you will be able to run an extract service.

Learning objectives

When you complete the exercises, you will know how to do the following tasks:

- Create a Repository Explorer folder
- Create an access definition to define the data to extract
- Edit an access definition to defines election criteria
- Create an extract service
- Test an extract service

Time required

This module should take approximately 60 minutes to complete.

Prerequisites

This tutorial requires an Optim Designer environment that includes a connection to an Optim repository and a data store alias that contains the Optim sample data.

This tutorial can be completed in the Optim Designer environment.

Creating a Repository Explorer Folder

In this exercise, you will create a Repository Explorer folder. A Repository Explorer folder contains services, access definitions, column maps, and table maps. Use the folders to organize these objects.

To create a data access plan and a selection policy:

1. Right-click the Repository Explorer view and click **New** > **Folder**. The New Folder window opens.
2. In **Name**, type `Tutorial` and click **OK**.

You have created a Repository Explorer folder named Tutorial.

Creating an access definition

In this exercise you will create an access definition. Use access definitions to specify the tables, relationship traversal, and selection criteria for the data you want to process.

Before creating an access definition, a data store alias must exist for the database that contains the start table.

To create an access definition:

1. Expand the Tutorial folder in the Repository Explorer, right-click **Access Definitions**, and click **New Access Definition**. The New Access Definition wizard opens.
2. On the Enter Access Definition Name page, enter `SAMPLE.AD`. Click **Next**.
3. On the Select a Data Store Alias page, select the data store alias that contain the Optim sample data. Click **Next**.
4. On the Select a Start Table page, do the following steps:
   a. In the **Start Table** field, enter the following search pattern: *SCHEMA*`.OPTIM_CUSTOMERS`, where *SCHEMA* is the schema that contains the Optim sample data. For example: `SAMPLE.OPTIM_CUSTOMERS`.
   b. Click **Find Tables**. The table list displays tables that match the pattern.
   c. Select the OPTIM_CUSTOMERS table.
   d. Click **Next**.
5. On the Specify a Table Selection Method page, click **Find related tables**. Click **Next**.
6. On the Enter Table Pattern and Select Related Tables page, do the following steps:
   a. Click **Find Tables**. The table list displays tables related to OPTIM_CUSTOMERS.
   b. Click **Select All**.
   c. Click **Finish**.

The new access definition is displayed in the Access Definition Editor.

Defining selection criteria

In this exercise, you will define selection criteria in the access definition. Use selection criteria to focus on a specific set of related data by defining an SQL WHERE clause and using substitution variables with default values.

Selection criteria must conform to SQL syntax and include relational or logical operators. Logical operators and syntax vary among DBMSs. Refer to the appropriate DBMS documentation for information.

To select the desired set of data for a table, you may need a combination of `AND` and `OR` logical operators.

To define selection criteria:

1. Expand the Tutorial folder in the Repository Explorer, expand the **Access Definitions** node, and double-click the SAMPLE.AD access definition. The Access Definition Editor opens.
2. Select the **Tables** tab.
3. Select the OPTIM_CUSTOMERS table from the table list.
4. Click **Edit Selection Criteria**. The Edit Selection Criteria window opens and displays a WHERE clause for the table.
5. Enter the following criteria: `COUNTRY_CODE='US'`.

   Click **Check Syntax** to verify the syntax and identify errors.
6. Click **OK** to return to the Access Definition Editor.

   If the syntax is not valid, a prompt will open to identify to error. You cannot save the criteria if it contains errors.
7. From the main menu, click **File** > **Save** to save the access definition.

You have defined selection criteria that will only select rows from the OPTIM_CUSTOMERS table in which the value of the COUNTRY_CODE column is 'US'.

Creating an extract service

In this exercise, you will create an extract service based on the data specified in an access definition. Use an extract service to copy a set of related rows from one or more tables and save the rows to a file data store.

To create an extract service:

1. Expand the Tutorial folder in the Repository Explorer, right-click **Services**, and click **New Service**. The New Service wizard opens.
2. On the Enter a Service Name and Select a Service page, do the following steps:
   a. In the **Name** field, enter `SAMPLE.EXTRACT`.
   b. In the list of service types, select **Extract**.
   c. Click **Next**.
3. On the Select an Access Definition page, select SAMPLE.AD. Click **Next**.
4. On the Enter a Target File Data Store Name page, enter `SAMPLE_EXT` in the **Target file data store name** field.
5. Click **Finish**.

The new extract service displayed in the Extract Service Editor. You can use this service to create a file data store for a convert, insert, or load service. You can also use the file data store to create a column map or table map.

Masking Data with Optim Designer

This tutorial teaches you how to use Optim Designer to create a convert service that transform data in a file data store.

This tutorial requires the SAMPLE_EXT file data store created in the "Extracting data with Optim Designer" on page 121 tutorial.

After completing this tutorial you will be able to apply a privacy function and run a convert service.

Learning objectives

When you complete the exercises, you will know how to do the following tasks:

- Create a Repository Explorer folder
- Create an access definition to define the data to extract
- Edit an access definition to defines election criteria
- Create an extract service
- Test an extract service

Time required

This module should take approximately 60 minutes to complete.

Prerequisites

This tutorial requires an Optim Designer environment that includes a connection to an Optim repository and a data store alias that contains the Optim sample data.

This tutorial can be completed in the Optim Designer environment.

Creating a table map

In this exercise, you will create a table map. Use a table map to define specifications for correlating source and destination tables of compatible data. You can map tables that have different names, modify table names, exclude tables from a process, or include column maps for greater control over the data.

To create a table map:

1. Expand the Tutorial folder in the Repository Explorer, right-click **Table Maps** and click **New Table Map**. The New Table Map wizard opens.
2. On the Enter Table Map Name page, enter `SAMPLE.TMAP` in the **Name** field. Click **Next**.
3. On the Select a Data Store Alias page, select SAMPLE.EXT.
4. Click **Finish**.

The new table map is displayed in the Table Map Editor.

You must use the editor to define target data stores and schemas before saving the table map.

Editing a table map

In this exercise, you will use the Table Map Editor to define target data stores and schemas for source tables in a table map.

To edit target data in a table map:

1. Expand the Tutorial folder in the Repository Explorer, expand the **Table Maps** node, and double-click the SAMPLE.TMAP table map. The Table Map Editor opens.
2. Select the **Table map** tab.
3. In the **Data store alias and schema map** area, select the row that contains the source data store.
4. Click the **Target Data Store Alias** cell and select the data store alias that contains the Optim sample data.
5. Click the **Target Schema** cell and select the schema that contains the Optim sample data.
6. From the main menu, click **File** > **Save** to save the table map.

The table map uses the same source and target tables in order to mask source data and maintain the same schema and data store alias.

Creating a column map

In this exercise, you will create a column map. A column map provides specifications needed to match or exclude columns from processing in a data management service. Convert, insert, and load services must reference a table map, which may reference one or more column maps. You can use a column map to define data transformations with functions or column map procedures.

To create a column map:

1. Expand the Tutorial folder in the Repository Explorer, right-click **Column Maps** and click **New Column Map**. The New Column Map wizard opens.
2. On the Enter Column Map name page, enter `SAMPLE.CMAP` in the Name field. Click **Next**.
3. On the Select a Source File Data Store Alias page, select SAMPLE_EXT. Click **Next**.
4. On the Select a Source Table page, do the following steps:
   a. In the **Table Search Pattern** field, enter the following search pattern: *DATA_STORE_ALIAS.SCHEMA*`.OPTIM_CUSTOMERS`, where *DATA_STORE_ALIAS.SCHEMA* are the data store alias and schema that contain the Optim sample data. For example: `OPTIM.SAMPLE.OPTIM_CUSTOMERS`.
   b. Click **Find Tables**. The table list displays tables that match the pattern.

c. Select the OPTIM_CUSTOMERS table.
d. Click **Next**.

5. On the Select a Target Data Store Alias page, select the data store alias that contain the Optim sample data. Click **Next**.
6. On the Select a Target Table page, do the following steps:
   a. In the **Table Search Pattern** field, enter the following search pattern: *SCHEMA*.OPTIM_CUSTOMERS, where *SCHEMA* is the schema that contains the Optim sample data. For example: SAMPLE.OPTIM_CUSTOMERS.
   b. Click **Find Tables**. The table list displays tables that match the pattern.
   c. Select the OPTIM_CUSTOMERS table.
7. Click **Finish**.

The new column map is displayed in the Column Map Editor.

Applying a data masking function

In this exercise, you will apply a data masking function to a column in a column map. You can mask data such as national ID numbers, credit card numbers, dates, numeric values, and personal information.

To apply a data masking function.

1. Expand the Tutorial folder in the Repository Explorer, expand the **Column Maps** node, and double-click the SAMPLE.CMAP column map. The Column Map Editor opens.
2. Select the PHONE_NUMBER column.
3. Click **Apply Function**. The Apply Function window opens.
4. Select the **Shuffle** function. Click **OK**. The function name appears in the **Source Column** column and the function editor opens in the Column Map Editor.
5. In the function editor **Column map expression** field, enter SHUFFLE(RETRY=12).
6. From the main menu, click **File** > **Save** to save the column map.

You have applied the Shuffle Function to the PHONE_NUMBER column. The function will mask data by replacing values in the column with other values in the column. The function will search up to twelve times for a replacement value that does not match the source value.

Adding a column map to a table map

In this exercise you will a column map to a table map. A table map is required for a convert service. Use the associated column map to enable the convert service to perform the data masking function defined in the column map.

To add a column map to a table map:

1. Expand the Tutorial folder in the Repository Explorer, expand the **Table Maps** node, and double-click the SAMPLE.TMAP table map. The Table Map Editor opens.
2. Select the **Table map** tab.
3. In the **Table Map** area, select the OPTIM_CUSTOMERS table.
4. Click **Add Column Map**. The New Column Map window opens with a list of column maps that contain the selected tables.
5. Select the SAMPLE.CMAP column map. Click **OK**.
6. From the main menu, click **File** > **Save** to save the table map.

You have added the SAMPLE.CMAP column map (and its associated data masking function) to the SAMPLE.TMAP table map, which will enable a convert service to transform data.

Creating a convert service

In this exercise you will create a convert service to mask data.

To create a convert service:

1. Expand the Tutorial folder in the Repository Explorer, right-click **Services**, and click **New Service**. The New Service wizard opens.
2. On the Enter a Service Name and Select a Service page, do the following steps:
   a. In the **Name** field, enter `SAMPLE.CONVERT`.
   b. In the list of service types, select **Convert**.
   c. Click **Next**.
3. On the Select a Table Map page, select the SAMPLE.TMAP table map. Click **Next**.
4. On the Enter Target Properties page, enter `SAMPLE_CONV` in the **Target file data store name** field.
5. Click **Finish**.

The new convert service displayed in the Convert Service Editor. You can use this service to mask the PHONE_NUMBER column in the OPTIM_CUSTOMERS table.

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 other 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 grant 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 or send inquiries, in writing, to:

IBM World Trade Asia Corporation
Licensing 2-31 Roppongi 3-chome, Minato-ku
Tokyo 106-0032, Japan

The following paragraph does not apply to the United Kingdom or any other country 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 which has been exchanged, should contact:

IBM Corporation
Software Interoperability Coordinator, Department 49XA
3605 Highway 52 N
Rochester, MN 55901
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 information 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.

All IBM prices shown are IBM's suggested retail prices, are current and are subject to change without notice. Dealer prices may vary.

This information is for planning purposes only. The information herein is subject to change before the products described become available.

This information contains 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.

If you are viewing this information softcopy, the photographs and color illustrations may not appear.

Trademarks

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

IBM
IBM logo
DB2
AIX
Informix
InfoSphere
Optim

Netezza® is a registered trademark of IBM International Group B.V., an IBM Company.

Adobe, Acrobat, PostScript and all Adobe-based trademarks are either registered trademarks or trademarks of Adobe Systems Incorporated in the United States, other countries, or both.

Apache Derby is a trademark of The Apache Software Foundation.

Eclipse is a trademark of Eclipse Foundation, Inc.

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

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

Linux is a trademark of Linus Torvalds in the United States, other countries, or both.

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

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

Index

P

R

S

T

V

Printed in USA