WebSphere Transformation Extender

Database Interface Designer

Version 8.1

Note
Before using this information, be sure to read the general information in "Notices" on page 95.

October 2006

This edition of this document applies to WebSphere Transformation Extender, 8.1 and to all subsequent releases and modifications until otherwise indicated in new editions.

To send us your comments about this document, e-mail DTX_doc_feedback@us.ibm.com. We look forward to hearing from you.

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

Contents

Chapter 1. Database Interface Designer overview

The Database Interface Designer is used to import metadata about queries, tables, and stored procedures for data stored in relational databases. It is also used to identify characteristics of those objects to meet mapping and execution requirements such as update keys and database triggers.

The Database Interface Designer is used to:

- specify the databases to use for a source or target
- define query statements
- automatically generate type trees for queries or tables

After defining database queries or tables in the Database Interface Designer, define your map in the Map Designer where, in map cards, you can specify an input source as either a query or a stored procedure and an output target as either a table or a stored procedure.

Database connectivity is supported under the control of Relational Database Management Systems (RDBMS).

The adapters provide the option of using a driver to connect to the platform of your choice so that you can automatically create type trees for database queries and tables. The adapters also provide a test environment on the PC for maps using data stored in a database.

You can also install adapters on additional systems to provide remote database connectivity, such as with ODBC, Oracle, Sybase, or DB2, on operating systems such as HP-UX, Sun Solaris, AIX, and so on.

Non-Windows adapters do not require the Database Interface Designer if you plan to only use **mtsmaker** without a database/query file (MDQ) to generate each type tree. For information about using **mtsmaker**, see the Resource Adapters documentation.

Basic steps for using database data

The following is a high-level description of the steps required to use database data.

To use the Database Interface Designer to import database definitions

1. Create a database/query file (MDQ).
2. Define a database.
3. If the database is to be used as an input, define a query for that database.
4. Generate the type tree for the query, table, view, stored procedure, or message from which or to which you want to map.
5. If the database is to be used as an output, set keys and designate columns for performing SQL updates.
6. Save the MDQ file.

To use the Map Designer to configure database sources, targets, or operands in a rule

1. From the input or output card(s) in the executable map, select **Database** as the value for either the **GET Source** or **PUT Target** setting.
2. Select the MDQ file that contains the database-specific source or target information.
3. Perform the following for an input and/or an output card:
   - In an input card, select a query from the MDQ file.
   - In an output card, specify a table name or stored procedure.

Chapter 2. Database Interface Designer basics

This chapter introduces the **Database Interface Designer** window and provides information about working with the graphical user interface.

Starting the Database Interface Designer

During installation, an entry for the Database Interface Designer is added to the WebSphere Transformation Extender program folder (listed under the Design Studio).

When the Database Interface Designer runs, the Startup dialog appears. In this dialog, you can select how the Database Interface Designer program should open. The following options are available:

- **Open an existing database/query file**

  Browse for the MDQ file you want.
- **Create a new database/query file**

  Create a new MDQ file.
- **Open a recently used database/query file**

  Select one or more files from the displayed file list. You can also double-click on a file from this list to open it.

The Startup dialog can be disabled by enabling the **Do not show this at startup** check box; however, you can always access this dialog from the **Database Interface Designer** window **Help** menu by selecting the **Startup Window** menu option.

Database Interface Designer user interface

The main **Database Interface Designer** window provides a graphical user interface in which to create and maintain MDQ files. These files contain database definitions that include information such as database name, connection information, queries, stored procedures, and so forth.

When you create a database/query file in the Database Interface Designer, a file named **Database_Query File** (which is the default file name) followed by an assigned, sequential number appears in the Navigator. This indicates that you can begin defining databases, queries, and so on. Save this database/query file (a file name with an MDQ extension), providing an appropriate name and location.

The Navigator

The Navigator graphically presents all of your opened MDQ files and the databases that they contain. It also provides a graphical representation of the queries, stored procedures, message queues, and tables or views that have type trees generated. Also displayed are tables and views with update keys defined, as well as variables that are defined in each MDQ file.

Changing the appearance of the navigator

The Navigator can be shown or hidden. It also can be presented as either a docked window or a floating window. These choices are available from the context menu of the Navigator. To access the context menu, right-click on the top border of the Navigator.

To show or hide the Navigator:

1. To show the Navigator, from the **View** menu, select **Navigator**.

   A check mark appears next to Navigator in the **View** menu, indicating that the Navigator is displayed.
2. To hide the Navigator, from the context menu of the Navigator, select **Hide**.

   or

   Repeat Step 1 above to disable the Navigator selection from the **View** menu.

To float the Navigator in the main window:

1. Starting with the Navigator being docked, right-click on the top border of the Navigator.

   The context menu appears.
2. Click **Float In Main Window**.

   The Navigator is now a separate window, floating in the main window.

To dock the Navigator:

This procedure assumes that the Navigator is floating in the main window.

1. In the Navigator, right-click anywhere around the border of the window, except in the title bar.

   The context menu appears.
2. If the **Allow Docking** option is not available, disable the check mark from the **Float In Main Window** option.

   The Navigator is docked.

   or

   If the **Allow Docking** option is available, select it and go to the next step.
3. After selecting **Allow Docking**, you can toggle between a docked window and a floating window by double-clicking the top border (below the title bar) of the Navigator. Double-click the title bar so that the Navigator is again docked.

Menu commands and tools

Actions can be performed in the **Database Interface Designer** window using menu commands, tools, and shortcut keys. Not all menu commands have corresponding tools, and not all tools are represented in the command menus. When working with database/query files, you can activate commands in several different ways:

- Select functionality using the menu bar.
- Right-click on any entry in the Navigator to display its context menu.
- Click the tools on the toolbar.
- Double-click objects in the Navigator.

Note: Database Interface Designer commands are available using the methods listed above. Most procedural information in this guide uses the menu access as a default method. Use the access method most convenient for you.

Menu

The **Database Interface Designer** window menu provides the common menu structure for commands generally used with Windows programs, as well as for commands to accomplish specific Database Interface Designer tasks.

Toolbar

The toolbar is a part of the **Database Interface Designer** window, providing quick access to various tools that invoke Database Interface Designer actions while working with database/query files. The left side of the toolbar provides the tools generally available in Windows applications. The remaining tools are specific to the Database Interface Designer. The applicability and behavior of each tool depends upon the Database Interface Designer object selected and its current state.

File menu

The **File** menu provides the commands that are generally available in Windows applications.

Using the keyboard, press `Alt F` to view the **File** menu.

Command	Shortcut Key	Description
New	**Ctrl+N**	Creates a new MDQ file
Open	**Ctrl+O**	Opens an existing MDQ file
Close		Closes the selected MDQ file
Save	**Ctrl+S**	Saves the selected MDQ file
Save As		Saves the selected MDQ file with a different name
Source Control → Create Project	**Alt+F, L, C**	Allows you to begin the process of creating a project using the third-party source control application you have selected.
Source Control → Open Project	**Alt+F, L, O**	Displays the Project window in which you can begin the process of selecting an existing project in which to work using a third-party source control dialog.
Source Control → Get Latest Version	**Alt+F, L, G**	Displays the Get Latest Version dialog in which you can select related files that you can get.
Source Control → Check Out	**Alt+F, L, E**	Displays the Check out file(s) dialog in which you can specify the files to be checked out.
Source Control → Check In	**Alt+F, L, I**	Displays the Check in file(s) dialog in which you can either only check in or both check in and check out again the selected files.

Command	Shortcut Key	Description
Source Control → Undo Check Out	**Alt+F, L, U**	Displays the Undo Check Out dialog in which you can reverse a check out of one or more files without either the check out or the undo of the check out showing up in the archive history.
Source Control → Add to Source Control	**Alt+F, L, A**	Displays the Add to source control dialog in which you select one or more files to add to your source control project archives.
Source Control → Remove from Source Control	**Alt+F, L, R**	Displays the Remove from source code control dialog in which you can specify the file(s) to be removed from the specified source control project archives.
Source Control → Show Workfiles	**Alt+F, L, F**	Displays the Project dialog in which you can view all of the workfiles in this project archive.
Source Control → Add Workfiles	**Alt+F, L, W**	Displays the Add Workfiles dialog in which you can specify the workfiles to be added to the project archives.
Source Control → Show History	**Alt+F, L, H**	Displays a third-party, source control application-specific dialog that allows you to define what you want to see in a history report generated by that third-party application.
Source Control → Show Properties	**Alt+F, L, P**	Displays a third-party, source control application-specific dialog that allows you to view labels and other source control-determined information on a revision-by-revision basis as recorded in that third-party source control application.
Source Control → Refresh Status	**Alt+F, L, R**	Synchronizes the project information for files displayed in your designer with the archival information in your third-party source control.
Database Query File Differences	**Alt+F, D**	Displays the Select First File dialog in which you can begin the process of specifying the two database/query (MDQ) files to compare.
Trace	**Alt+F, T**	Enables trace functionality to create a Database Interface Designer trace file for the selected MDQ file
Print	**Ctrl+P**	Displays the Print dialog in which to specify and print definitions for the selected database(s)
Print Setup	Alt+F, R	Displays the Print Setup dialog in which to configure printer options
Recent File		Opens the selected MDQ file from this list
Exit	Alt+F, X	Quits the application and prompts to save changes

Edit menu

The **Edit** menu provides the editing commands generally available in Windows applications. It also contains the Find command that can be very useful when viewing trace files in a trace window.

Using the keyboard, press `Alt E` to view the **Edit** menu.

Command	Shortcut Key	Description
Undo	Ctrl+Z	Reverses the last-performed action
Cut	Ctrl+X	Deletes the selection and places a copy on the clipboard
Copy	Ctrl+C	Copies the selection to the clipboard
Paste	Ctrl+V	Inserts the contents of the clipboard to the cursor location
Find	Ctrl+F	Displays the Find dialog in which to locate specified information in the active trace window
Replace	Ctrl+H	Displays the Replace dialog in which to specify the text upon which to search and the text that is replacing it

View menu

The **View** menu provides selections to control the appearance of your Database Interface Designer environment.

Using the keyboard, press `Alt V` to view the **View** menu.

Command	Shortcut Key	Description
Toolbar(s)	**Alt+V, T**	Displays the Customize dialog for toolbars in which to select the toolbars to be displayed and to adjust the tools displayed on specific toolbars
Status Bar	**Alt+V, S**	Shows or hides the status bar located at the bottom of the **Database Interface Designer** window
Navigator	**Alt+V, N**	Shows or hides the Navigator
Trace File	**Alt+V, F**	Displays the Database Interface Designer trace file for the selected MDQ file in a separate trace window

Database menu

The **Database** menu provides commands to initiate actions for a selected database.

Using the keyboard, press `Alt D` to view the **Database** menu.

Command	Shortcut key	Description
New	**Ctrl+D** or **Insert**	Displays the Database Definition window so that you can add a database to the associated MDQ file **Note:** You must first select the **Databases** object in the Navigator to be able to do this.

Command	Shortcut key	Description
Edit	none	Displays the Database Definition window that allows you to modify the selected database **Note:** There is no shortcut key. But you can double-click the database object name to perform the same function as the tool.
Delete	**Delete**	Deletes the selected database
Generate Tree From → Table	**Alt+D, G, T**	Displays the Generate Type Tree from Tables dialog that allows you to generate a type tree from a table or view in the selected database **Note:** You can also double-click the **Tables** object in the Navigator to perform the same function as the tool.
Generate Tree From → Procedure	**Alt+D, G, P**	Displays the Generate Type Tree from Stored Procedures dialog for the selected database or stored procedure in which you can specify the generation of a type tree from a stored procedure **Note:** You can also double-click the **Stored Procedures** object in the Navigator to perform the same function as the tool.
Generate Tree From → Queue	**Alt+D, G, Q**	Displays the Generate Type Tree from Queues dialog in which you can specify the generation of a type tree from a message queue
Set Update Keys	**Alt+D, S**	Displays the Set Table Update Key Columns dialog in which you can specify update keys and the key columns to update in the selected database

Query menu

The **Query** menu provides commands to initiate actions for a selected query.

Using the keyboard, press `Alt Q` to view the **Query** menu.

Command	Shortcut Key	Description
New	**Ctrl+Q** or **Insert**	Displays the New Query dialog in which to add a new query **Note:** You must first select the **Queries** object in the Navigator to be able to do this.
Edit	**Alt+Q, E**	Displays the Edit/View Query dialog in which to modify the selected query **Note:** You can also double-click the database query name to perform the same function as the tool.
Delete	**Delete**	Deletes the selected query
Generate Tree	**Alt+Q, G**	Displays the Generate Type Tree from Query dialog that allows you to generate a type tree from the selected query

Command	Shortcut Key	Description
Define Variables	**Alt+Q, V**	Displays the Define Variables dialog in which to specify pseudo variable values **Note:** You can also double-click the **Variables** object in the Navigator to perform the same function as the tool.
Define Trigger	**Alt+Q, T**	Displays the Trigger Specification dialog in which to specify the input events that must occur to launch a map using the selected query as a data source

Tools menu

The **Tools** menu provides options to customize your Database Interface Designer environment.

Using the keyboard, press `Alt L` to view the **Tools** menu.

Command	Shortcut Key	Description
Shortcuts	**Alt+L, S**	Displays the Shortcut Keys dialog in which to assign shortcut keys or key combinations to specific Database Interface Designer operations
Options	**Alt+L, O**	Displays the Options dialog in which to configure Database Interface Designer options for backups, the Navigator, trace windows, tables/views, and confirmations

Window menu (Alt+W)

The **Window** menu contains commands to provide control of open trace windows.

Using the keyboard, press `Alt W` to view the **Window** menu.

Command	Shortcut Key	Description
Close All	none	Closes all open trace windows
Cascade	none	Arranges all open trace windows so that they overlap in a descending pattern
Tile Horizontally	none	Arranges all open trace windows as horizontal, non-overlapping tiles
Tile Vertically	none	Arranges all open trace windows as vertical, non-overlapping tiles
Arrange Icons	none	Arranges all minimized trace windows in an orderly fashion at the bottom of the **Database Interface Designer** window
List of recently used files	none	Selecting a specific trace file from this list makes it the active window. A check mark appears next to the trace file that is in the active trace window.

Help menu

The **Help** menu offers choices to display information about the Database Interface Designer.

Using the keyboard, press `Alt H` to view the **Help** menu.

Command	Shortcut Key	Description
Contents	none	Displays the contents of the Help system which also lists the Help topics
Startup Window	none	Displays the Startup dialog in which to select whether to create a new or open an existing MDQ file
About Database Interface Designer	none	Displays application-specific information

Configuring the environment

Much of the Database Interface Designer environment can be configured to accommodate your preferred work environment. For example, you can:

- Specify various user interface options (font, line appearance, dialog display)
- Select the tools to display
- Change the look of the tools on the toolbar
- Assign shortcut keys

Tools > Options

From the **Tools** menu, select **Options**. The Options dialog appears. Select choices representing various aspects of the Database Interface Designer environment and configure them as desired.

General options

In the list of options, select **General** to specify values concerning the backing up and saving of your MDQ files. The fields in this dialog are as follows:

Field Description

Auto-save files every *n* minutes (where *n* represents a number)
This spin button indicates the time interval at which to automatically save opened MDQ files. The default value is 0.

Show Banner
This check box specifies whether to display the banner, which appears between the title bar and the menu bar of the **Database Interface Designer** window. The default setting is enabled.

Backup on save
This check box specifies whether to create a backup copy of each MDQ file when the file is saved. The default setting is enabled.

Navigator options

In the list of options, select **Navigator** to specify how to display objects in the Navigator. The Navigator options are described in the following table.

Field/Button	Description
Font	Select this button to display the Font dialog in which to select the font and font size to be used for text. The default selection in the **Sample** list is Arial (10pt).

Field/Button	Description
Lines	This group box displays the options that control the appearance of lines in the Navigator. Select one of the following options.
None	No lines are displayed.
Solid	Solid lines are displayed.
Dotted	Dotted lines are displayed. This is the default setting.
Show tool tips	This check box allows you to display the name of each object when the cursor is held over it and the Navigator is sized too small for the name to be completely displayed. The default setting is enabled.

Trace window options

In the list of options, select **Trace Window** to specify the font used to display text in the Trace window.

Tables/views option

In the options list, select **Tables/Views** to determine the objects to be displayed that are associated with the database. The fields in this dialog are as follows:

Field Description

List tables
This check box determines whether tables are displayed in the database list. The default value is enabled.

List views
This check box determines whether views are displayed in the database list. The default value is enabled.

List synonyms
This check box determines whether database synonyms are displayed in the database list. The default value is enabled.

Confirmations options

In the list of options, select **Confirmations** to specify the actions for which you want a confirmation dialog displayed before completion of those actions.

Field/Button
Description

Database operations
This group box displays the options that control whether confirmation dialogs are displayed with regard to database operations. Select any or all of these options.

Deleting database(s)
This check box determines whether a confirmation dialog appears when deleting a database. The default value is enabled.

Copying database(s)
This check box determines whether a confirmation dialog appears when copying a database. The default value is enabled.

Moving database(s)
This check box determines whether a confirmation dialog appears when moving a database. The default value is enabled.

Table operations
This group box displays the options that control whether confirmation dialogs are displayed with regard to table operations. Select any or all of these options.

Deleting table(s)
This check box determines whether a confirmation dialog appears when deleting a table. The default value is enabled.

Copying table(s)
This check box determines whether a confirmation dialog appears when copying a table. The default value is enabled.

Moving table(s)
This check box determines whether a confirmation dialog appears when moving a table. The default value is enabled.

Query operations
This check box allows you to display the name of each object when the cursor is held over it and the Navigator is sized too small for the name to be completely displayed. The default setting is enabled.

Deleting query(s)
This check box determines whether a confirmation dialog appears when deleting a query. The default value is enabled.

Copying query(s)
This check box determines whether a confirmation dialog appears when copying a query. The default value is enabled.

Moving query(s)
This check box determines whether a confirmation dialog appears when moving a query. The default value is enabled.

Shortcut keys

You can assign your own shortcut keys for any existing or new menu items. Using the Shortcut Keys dialog, you can:

- Assign shortcut keys.
- Remove shortcut key assignments.
- Restore the shortcut key assignments present at installation.

For information about these procedures, see the Design Studio Introduction documentation.

Chapter 3. Database/query files

This chapter discusses how to use the Database Interface Designer when working with MDQ files to perform the following types of tasks:

- Create MDQ files and learn about their XML format. (For more information about the XML format, see ″Understanding the MDQ XML format″ .)
- Define the various objects included in an MDQ file. (For more information about such objects as queries and variables, see ″Defining a query″ and ″Defining Variables in SQL Statements″ .)
- Generate type trees. (See ″Generating type trees″ .)
- Print reports and enable trace functionality. (See ″Printing reports″ and ″Database Interface Designer trace files″ .)

Creating database/query files

An MDQ file contains the definitions for one or more databases, as well as queries, stored procedures, and other specifications, that may contribute to the execution of a map. Use the commands on the **File** menu in the Database Interface Designer to create and save a database/query file with an **.mdq** file extension. This file name and path appears in the title bar of the **Database Interface Designer** window when it is the selected file in Navigator, indicating that it is the active MDQ file.

After starting the Database Interface Designer, the Navigator lists one or more MDQ files depending upon whether your selection was to create a new MDQ file or to open one or more existing files.

When an MDQ file is created, it appears in the Navigator next to the appropriate icon. The **Database_QueryFile** file name is automatically assigned, along with a sequential number.

To save an MDQ file or rename it:

1. From the **File** menu, select **Save As**.
   The Save As dialog appears.
2. Enter the new file name and select the path.
3. Click **OK**.

Defining a database

When an MDQ file appears in the Navigator, you can add new database definitions to it or you can modify the name of an existing database. This is done using the Database Definition window. Each **Database Definition** window contains some settings that are common across all platforms and others that are platform-specific. To view more information about each platform-specific setting, refer to either the context-sensitive help available from the dialog itself or to each platform-specific reference guide.

For the purposes of outlining a basic procedure to define a database, the ODBC adapter for the Windows platform will be used. An example follows.

see the following table for a listing of the settings in this window and their descriptions.

Setting		Description
Database Name		This is the name of the database being defined. This name appears in the Navigator.
Adapter		
	Type	This is a list of adapters that can host the database you are defining. Select one from the list. The default value is **ODBC**. This selection will affect the list of supported platforms displayed in the **Platform** list.
	Platform	This is a list of platforms upon which the adapter (that you selected in the **Type** list) is supported.
Access user tables/procedures only		This setting determines level of user access and, to some extent, the presentation of the information being presented. The default value is **Yes**. If **No** is selected, all of the names (of tables, views, and so on) in the database are accessible and are presented, including the respective prefix of each associated user ID. If **Yes** is selected, only the names (of tables, views, and so on) in the databases associated with the current user ID are accessed and presented. Because these are all associated with the same user, no prefix is added.
Surround table/column names with		This setting indicates the character you want to use to enclose table and column names. For example, many databases require that names be enclosed by single or double quotation marks if they have spaces in their names, or when the names are SQL reserved words. When you specify table names in output cards or column names in update keys, these names will be enclosed by the character defined in this setting to provide database compatibility. There is no default value.
Data Source		
	Database Interface Designer	Both this and the **Runtime** settings are platform-specific settings. On other platforms, you may have different settings. Again, see the platform-specific adapter reference guide or the context-sensitive help for more setting-specific information. This is the name of the data source used to access database information for design-time (pre-production or testing) purposes.
	Runtime	This is the name of the data source used to access database information for run-time (map execution) purposes.

Setting		Description
Security		
	User ID	This is the user ID used to access the database. If you do not know this information, contact your database administrator. There is no default value.
	Password	This is the password used to access the database. If you do not know this information, contact your database administrator. There is no default value.

To define a database:

1. From the Navigator, select **Databases** under the MDQ file.
2. From the **Database** menu, select **New.**

   or

   Right-click the **Databases** object in the Navigator and select **New**.

   The Database Definition window appears.
3. Enter information for the remaining settings as desired.
4. Click **OK**.

   The new database name appears in the Navigator next to the database icon.

Defining a query

After you have opened an MDQ file in the **Database Interface Designer** window and a database has been defined, you can specify queries. To use a query as a source, define the query by assigning it a name and entering either the SQL SELECT statement or the stored procedure invocation statement.

To define a query:

1. In the Navigator, select **Queries** under the database icon for which you want to add a query.
2. From the **Query** menu, select **New**.

   The New Query dialog appears.
3. In the **Name** field, enter a name for your query. Use this name to reference this query using the Map Designer or you can use it in a data source override execution command.
4. In the **Query** text box, enter an SQL statement that defines how data should be retrieved from the database.

   If your SELECT statement includes table names with spaces, you must enclose them with either single or double quotation marks.
5. Click **OK**.

   The new query appears in the Navigator next to the query icon.

Editing or deleting a database or query

The following procedures describe how to edit and delete a database or a query, respectively.

To edit a database or query

1. In the Navigator, select the name of the database or query to be edited.
2. From the **Database** menu (or **Query** menu as appropriate), select **Edit**.

   or

   In the Navigator, right-click the database or query icon and select **Edit**.

   The Edit/View Query dialog appears.
3. Make any necessary changes and click **OK**.

To delete a database or query

1. In the Navigator, highlight the database(s) or one or more queries you want to delete.
2. From the **Database** menu (or **Query** menu as appropriate), select **Delete**.

   or

   In the Navigator, right-click the database or query icon(s) and select **Delete**.

   A confirmation dialog appears.
3. To confirm the deletion, click **Yes**.

Understanding the MDQ XML format

The database/query (MDQ) file is a configuration file that is used by WebSphere Transformation Extender when accessing databases using database adapters. Some of the information contained within an MDQ file could include the following:

- database logical names (to distinguish among settings for the databases defined in this file)
- database adapter types
- database connection parameters
- user names and passwords
- information about tables
- queries and stored procedures accessed by the adapters
- primary keys for the tables (used for the database updates)
- event trigger information for the Launcher

XML Schema

The MDQ files are well-formed, valid XML documents. MDQ files can be created using the Database Interface Designer (DID) or any other utility capable of creating XML documents. The XML Schema file that is used by WebSphere Transformation Extender to validate MDQ files is **mdq.xsd**.

Note: Do not modify the content of the **mdq.xsd** file. Unexpected results may occur if you do.

If *install_dir* is the directory into which the WebSphere Transformation Extender product is installed, the **mdq.xsd** file is copied to the following platform-specific directories as part of that installation process:

Platform
: **Location**

Windows
: ***install_dir***

UNIX ***install_dir*/src**

Saving database/query files

Note: If you do not use the Database Interface Designer to create MDQ files, verify that the generated XML files are well-formed and valid for the provided **mdq.xsd** XML Schema.

The following topics contain information related to saving MDQ files from the Database Interface Designer (regardless of whether they were originally created in the Database Interface Designer or using another utility):

- ″XML prolog text″
- ″XML data from XML-generated MDQ files″
- ″Attribute formatting″
- ″Password encryption″
- ″Backup copies″

XML prolog text

The XML prolog will always be generated as shown below:

```
<?xml version="1.0" encoding="UTF-8"?>
```

This means that the MDQ files saved from the Database Interface Designer use the UTF-8 encoding scheme.

XML data from XML-generated MDQ files

If you have created this MDQ file outside of the DID, the following information is ignored by the DID when it reads the MDQ file and the information is removed from the MDQ file when it is saved from the DID:

- XML comments
- namespace declarations
- CDATA sections and processing instructions

  The content from the CDATA sections is preserved. However, it is saved using the XML entity references.

Attribute formatting

Attribute values are always enclosed in double quotes.

Password encryption

The XML schema provides two different password elements:

- `PASSWORD.literal`
- `PASSWORD.encrypted`

In an MDQ file that has not been saved in the Database Interface Designer, only the `PASSWORD.literal` value can be specified, which is not encrypted. When the file is saved in the Database Interface Designer, the password will be encrypted (using the `PASSWORD.encrypted` element).

Backup copies

If the MDQ file was created (and saved) in an earlier version of the DID and when that file is edited and saved from the DID, this newly saved MDQ file is in the current version. In addition, a backup copy of the MDQ file is created for the older version using the ODQ filename extension (instead of MDQ).

Processing MDQ files

The MDQ files can be processed either at runtime or design-time, depending upon how (and where) the files are specified.

Runtime execution

If the `-MDQ` adapter command is used along with the `DBLOOKUP`, `DBQUERY`, `GET` or `PUT` map functions or if it is specified as an adapter command in the database cards, the MDQ file is parsed at runtime.

Design-time execution

If the MDQ file name is specified in the `DatabaseQueryFile` setting of a map card, the MDQ file is parsed when the map is compiled, which is during design time for the map, not at runtime.

Comparing database/query files

Database/query (MDQ) files contain definitions of one or more databases, as well as queries, stored procedures, tables, queues, and variables. The file differentiation process identifies MDQ file differences first at the database-level, and then at the query-, stored procedure-, table-, and variable-levels. See ″Creating a database/query file comparison″ .

After you have performed a comparison between two MDQ files, color-coding is used to highlight what is unique between the two:

- Differences are highlighted in red.
- New information is highlighted in blue.

For more information, refer to ″Viewing database/query file differences″ .

Criteria for analyzing differences

The criteria for analyzing differences for the different entities is as follows:

- MDQ file
  - Any of the database definitions are different.
  - Any of the variables are different.
  - Any of the objects exist in one, but not the other, MDQ file.
- database definition
  - Any of the database definition settings are different.
  - Any of the syntax of the queries, column overrides of the stored procedures, column overrides or update keys of the tables, or any of the values of the variables are different.
- query
  - The syntax of the queries is different.
  - Column overrides of the queries are different.
- stored procedure
  - Column overrides for the stored procedures are different.
- table
  - Column overrides for the tables are different.
  - Update keys for the tables are different.
- variable
  - Values for the variables are different.

Creating a database/query file comparison

Use this procedure to compare database/query file differences between two selected MDQ files. After you have created the comparison, you can view the results.

To create an MDQ file comparison:

1. From the **File** menu, select **Database Query File Differences**.

   The Select First File dialog appears.
2. Navigate your file system and select the first MDQ file to be compared.
3. Click **Open**.

   The Select Second File dialog appears.
4. Navigate your file system and select the second MDQ source file to be compared.
5. Click **Open**.

   The progress of the comparison is shown briefly in the Database Source File Differences Analysis dialog. When the analysis has completed, the Database Differences window appears.

Viewing database/query file differences

Note: The Database Differences window should already appear. If you do not see it, you must create the file comparison as described in ″Creating a database/query file comparison″ .

Differences are indicated using color-coded text:

- Differences are **red**.
- Additions or new information items are **blue**.

To view MDQ file differences:

1. Place current focus in theDatabase Differences window.
2. Resize any of the four panes as needed.
3. Select any database, query, stored procedure, table, or variable to view the differences.

   The settings display in the two lower windowpanes, depending upon what you select.
4. Press **F8** or **F7** to view the next or previous difference, respectively.

   In the Database Differences window, you can view all of the settings for the selected element in the compared MDQ file, even though differences may not exist.

Database/query file difference results

When database/query source files are compared, they are considered different if any of the database definition settings are different, any of the variables are different, or any of the objects exist in one MDQ file and not the other. Text color is used to provide a visual distinction. see the table below as a guide:

Condition	Upper Panes	Lower Panes
Same name and content	black text	black text on tab, only one pane at a time
Same name, different content	red text	red text on tab, both panes appear, unique text in red

Condition	Upper Panes	Lower Panes
Different name and content	blue text	red text on tab, only one pane at a time, unique text in **green** (for database settings only, rest have unique text in blue)

Database definition setting differences

If any database definition settings of either of the two compared MDQ files are different, both database names appear in red letters in the Database Differences window.

The **Database Properties** tab in each lower windowpane also appears in red letters. (If there were no differences, this tab would appear in black letters.)

Query differences

If there are differences between the two compared MDQ files with regard to query syntax or column overrides, the name of the query with differences appears in red letters in the Database Differences window. If the query in each MDQ file is uniquely named, each query name appears in blue letters.

If each MDQ file has a query with the same name, but containing unique information, the **Query** and/or **Column Overrides** tabs in each lower windowpane appear in red letters. (If there were no differences, these tabs would appear in black letters and you would only see one display in the bottom pane at a time, depending upon where your cursor is in the upper pane.) When the tabs are red, you can go to each display and see the differences indicated in red letters.

Click the **Query** and **Column Overrides** tabs in each pane to view the differences.

Variable differences

If the variables of the two compared MDQ files have the same name, but different settings, the variable name will appear in red letters in the upper pane and both variables will appear in the bottom panes with the tab letters also in red letters. In the bottom panes, the unique value(s) will appear in red letters. If the variable names of either of the two compared MDQ files are different, the name of each variable appears in blue letters in the Database Differences window. If the variables are exactly the same, the name of the variable in both the upper and lower pane and the **Variables** tab would all appear in black letters.

Click the **Variables** tab in each pane to view the differences.

Unique database information

If the databases of the two compared MDQ files have the same name, but different settings, this tab will appear in red letters. Also, both databases will appear in the bottom panes of the Database Differences window. If the database names of either of the two compared MDQ files are different, the name of each database appears in blue letters in the Database Differences window. If the databases are exactly the same, the name of the database in both the upper pane and the **Database Properties** tab would all appear in black letters. In both of these latter scenarios (with the different-named or duplicate databases), you can view only one tab at a time by clicking on the Database icon in the upper (Navigator-like) portion of the window for the specific MDQ file. The resulting display in this tab is a reproduction of the settings of the Database Definition dialog. To see the respective database differences, you must click alternately on each database icon in each upper half of the Database Differences window.

Stored procedure differences

If the stored procedures of the two compared MDQ files have the same name, but different values, the stored procedure name will appear in red letters in the upper pane and both stored procedures will appear in the bottom panes with the **Column Overrides** tab letters also in red letters. In the bottom panes, the unique value(s) will appear in red letters. If the stored procedure names of either of the two compared MDQ files are different, the name of each stored procedure appears in blue letters in the Database Differences window. If the stored procedures are exactly the same, the name of the stored procedure in the upper pane and the **Column Overrides** tab and columns would all appear in black letters.

The columns on this tab (**Column Name, Column Length, Presentation**, and **Interpret As**) correspond to fields and list names in the Column Datatype Specification dialog.

Table differences

If the tables of the two compared MDQ files have the same name, but different values for column overrides or update key definitions, the table name will appear in red letters in the upper pane and both tables will appear in the bottom panes with the **Column Overrides** and/or **Update Keys** tab letters also in red letters. In the bottom panes, the unique value(s) will appear in red letters. If the table names of either of the two compared MDQ files are different, the name of each table appears in blue letters in the Database Differences window. If the tables are exactly the same, the name of the table in the upper pane and the **Column Overrides** and Update Keys tabs and related information would all appear in black letters.

Click the **Column Overrides** and **Update Keys** tabs in each pane to view the differences.

The columns on the **Column Overrides** tab (**Column Name, Column Length, Presentation**, and **Interpret As**) correspond to fields and list names in the Column Datatype Specification dialog.

Defining variables in SQL statements

Elements of SQL statements can be executed as map sources that are determined at runtime. Use the Database Interface Designer to define a statement variable with a pseudo value in an SQL statement and then pass the actual value on the command line at runtime. This technique is beneficial when using the `RUN` function because it allows one map to modify the SQL statement of another map or to build, potentially, an entire SQL statement.

When you generate type trees using the Database Interface Designer, a substitution value must be entered for each variable to ensure that the syntax of the SQL statement is valid. The Database Interface Designer provides a facility for specifying a value for these variables; however, the value you enter for a variable in the Database Interface Designer does not have to be the same value passed at runtime. (Any value can be passed.)

Defining a query with variables

In the Database Interface Designer Query dialog, variables can be specified in SQL statements as literals enclosed in pound sign (#) characters. For example, you might enter a statement that defines a variable named **ID**:

```
select * from BigTable where Identifier = #ID#
```

Because the value of a variable may be a text string, you can also create larger elements of the statement variable. For example:

```
select * from BigTable where #WhereClause#
```

When you define variables in a query, the Database Interface Designer automatically detects the presence of the variables in the statement and lists each variable in the Navigator, along with a variable icon.

Use the Define Variables dialog to enter the variable values. Note that you cannot generate a type tree for the query until you have specified a value for *each* variable it contains. Also, if the variable you are defining in the Define Variables dialog is a text string, you must enclose the value in single quotation marks.

The pseudo values specified using the Define Variables dialog are used in the Database Interface Designer only when accessing the database to generate the type tree. They are not used when executing a map.

To specify values in the Define Variables dialog

1. In the Navigator, select the variable to which you want to add one or more values.
2. From the **Query** menu, select **Define Variables**.

   The Define Variables dialog appears, listing all the variables in the MDQ file.
3. In the **Value** field, enter a value for each variable and click **OK**.

   The pseudo values are stored in the MDQ file and can only be changed in the Define Variables dialog. Also, if the variable you are defining is a text string, you must enclose the value in single quotation marks.

To delete a variable

When a variable is removed from a query statement that was previously defined in the Database Interface Designer, the variable is automatically deleted from the Navigator and no longer displays.

Specifying the values at runtime

Even though you have specified pseudo values for the defined variables using the Database Interface Designer, they are only substitution values that are not referenced at runtime. Values *must* be provided for the variables at runtime; otherwise, the SQL statement will be syntactically invalid.

The correct value for each variable is specified at run time using the Variable adapter command (`-VAR`) in the Input Source - Override execution command (`-ID`). For more information about how to specify values for these variables at runtime, see the Resource Adapters documentation.

Generating type trees

Use the Database Interface Designer to generate a type tree from a table, query, stored procedure, or message queue associated with a particular database. A type tree generated using the Database Interface Designer contains a type for the table, query, stored procedure, or message, as well as a row type and types for each column in that table, query, stored procedure, or message.

As an alternative, use **mtsmaker** on non Windows-based platforms to generate type trees when you cannot use the Database Interface Designer. For more information about using **mtsmaker**, see the Resource Adapters documentation.

From a table or view

When you generate a type tree for a table or view, information from your database and from the Database Interface Designer is used to create a type tree file (MTT). If you change your database table definition or edit the database definition in the Database Interface Designer, you *must* generate a new type tree to reflect the new information.

Generated type trees may differ when generated using different database adapters. For example, a date field may be represented differently by different databases. For this reason, use the same adapter type that will be used at run time to generate the type tree with the Database Interface Designer. Or, if you use ODBC, use the **Column Attributes Override** dialog that is described in the Resource Adapters documentation.

To define the type tree to generate from a table or view, use the Generate Type Tree from Tables dialog or the Generate Type Tree from Views dialog. The following table contains fields and descriptions.

Field	Description
Database Name	This is the name of the database from which you want to generate a type tree. (This name is automatically inserted based upon your selection in the Navigator.)
Tables/Views	This list displays all of the tables, views, or synonyms that have been defined in this database. Select one or more from this list. **Note:** If the table, view, or synonym you want is not displayed in this list, add it by right-clicking in the list, selecting **Insert**, and typing the desired name.
File name	This field displays a system-generated file name based upon the database selection in the Navigator. The default value is the database name with a type tree extension (MTT) and a default path. To change any of this information, click the browse button.
Overwrite file	This check box determines whether you want to replace the existing type tree file (MTT) upon saving it. The default value is disabled.
Override type	This check box determines whether to replace existing types with matching, newly generated types and to add any non-matching new types to the type tree file (MTT). This is useful when you want to add types for additional tables into an existing type tree. The default value is enabled.
Row group format	This list displays options to determine the row format in the generated type tree. The default value is **Delimited**.

Field		Description
	Delimited	Select this option to create rows delimited with a pipe character (|) or some other user-defined selection. This is the preferred option if you are using large column widths where the values may be smaller than the width of the columns. The pipe character is the default selection.
	Fixed	Select this option to create rows of fixed size in the generated type tree. There are certain situations in which you would want to select **Fixed**: • The query result contains a large number of rows where each row consists of small, fixed-length fields. In this situation, delimiters are unnecessary and they also consume memory. • You need to append fields in the output. This may be true if multiple fields have some meaning when concatenated together.
Delimiter		This is the value for the delimiter to be used when generating this type tree. For a list of available symbols, click the browse button. The Symbols dialog appears, allowing you to select the desired value. The default value is a pipe character (|).
Terminator		This is the value for the terminator to be used when generating this type tree. For a list of available symbols, click the browse button. The Symbols dialog appears, allowing you to select the desired value. The default value is **<CR><LF >**.
Release		This is the value for the release character to be used when generating this type tree. For a list of available symbols, click the browse button. The Symbols dialog appears, allowing you to select the desired value. The default value is an exclamation point (**!**).
Override Column Definitions		(For ODBC only) This check box indicates whether the database driver (with check box disabled) or a command line override (with check box enabled) is used to interpret columns. The default value is disabled. For more information about this field, see the adapter-specific documentation in each adapter reference guide.
Generate type trees in 1.4.0 format		(For Oracle adapter only) This check box determines whether you can generate type trees that are compatible with maps for Version 1.4.0. You might enable this functionality if you have legacy 1.4.0 maps that you want to maintain for some reason. The default value is disabled. For more information about this field, refer to adapter-specific documentation in each adapter reference guide.

Field	Description
Represent date/time columns as text items	This check box determines whether to automatically format this information in its database-specific date and time format (enabled) or as a text string (disabled). If you enable this check box (generating date and time as a text string), you may have to use either the `TEXTTODATE` or `TEXTTOTIME` function in a map rule to convert the text string to the database-specific date and time format. If you are generating new type trees, make sure this check box is disabled. The default is disabled. Because of adapter-specific differences in date and time formats, see the adapter-specific instructions in each adapter reference guide.
Treat text item as	This list displays the languages available in which to handle text items. The default value is **Western**.

To generate a type tree from a table or view:

1. In the Navigator, highlight the icon of the database containing the table from which you want to generate a type tree.
2. From the **Database** menu, select **Generate Tree From → Table**.

   or

   In the Navigator, right-click the database icon and select **Generate Tree From → Table**.

   The Generate Type Tree from Tables dialog appears, including a list of tables and views.
3. If you need to add a table because it is not displayed in the **Tables/Views** list, you must insert it using the instructions provided in the **Note** below.

   or

   If you see one or more tables in the **Tables/Views** list that you want to select, if it is one; select it. If it is multiple tables, press either **Shift** or **Ctrl** and click the desired table.

   If the table or view you want is not displayed in this list, add it by right-clicking in the list, selecting **Insert**, and typing the desired name.
4. To create a new type tree, in the **File name** field, specify the path and name (with an **.mtt** extension).

   or

   Click the browse button to select a type tree from the Save As dialog. Highlight it and click **OK**.
5. Specify the remaining options as desired. Refer to context-sensitive help for field-specific information.
6. Click **Generate**.

   The Database Interface Designer and the Type Tree Maker produce a type tree corresponding to the selected database table(s) or view(s).

The generated type tree is represented in the Navigator with the table/view icon next to the name.

From a stored procedure

Use the Database Interface Designer to generate type trees from the parameters of stored procedures.

An example of the Generate Type Trees from Stored Procedures dialog follows, along with a table describing its fields and their descriptions.

Field		Description
Database Name		(Display only) This is the name of the database from which you want to generate a type tree. (This name is automatically inserted based upon your selection in the Navigator.)
Stored Procedures		This list displays all of the stored procedures that have been defined in this database. Select one or more from this list. The list of stored procedures includes stand-alone procedures by default. (Procedures that are parts of a package are not listed.) If you need a procedure that is part of a package (whether this functionality is available is dependent upon your database), you may need to insert it into this list before selecting it. To insert a packaged stored procedure, right-click in the list, select **Insert**, and type the desired name. Use the following format: *schema.package.procedure*.
File name		This field displays a system-generated file name based upon the database selection in the Navigator. The default value is the database name with a type tree extension (**.mtt**) and a default path. To change any of this information, click the browse button.
Overwrite file		This check box determines whether you want to replace the existing type tree file (MTT) upon saving it. The default value is disabled.
Override type		This check box determines whether to replace existing types with matching, newly generated types and to add any non-matching new types to the type tree file (MTT). This is useful when you want to add types for additional stored procedures into an existing type tree. The default value is enabled.
Row group format		This list displays options to determine the row format in the generated type tree. The default value is **Delimited**.
	Delimited	Select this option to create rows delimited with a pipe character (|) or some other user-defined selection. This is the preferred option if you are using large column widths where the values may be smaller than the width of the columns. The pipe character is the default selection.

Field		Description
	Fixed	Select this option to create rows of fixed size in the generated type tree. There are certain situations in which you would want to select **Fixed**. • The query result contains a large number of rows where each row consists of small, fixed-length fields. In this situation, delimiters are unnecessary and they also consume memory. • You need to append fields in the output. This may be true if multiple fields have some meaning when concatenated together.
Delimiter		This is the value for the delimiter to be used when generating this type tree. For a list of available symbols, click the browse button. The Symbols dialog appears, allowing you to select the desired value. The default value is a pipe character (|).
Terminator		This is the value for the terminator to be used when generating this type tree. For a list of available symbols, click the browse button. The Symbols dialog appears, allowing you to select the desired value. The default value is **<CR><LF >**.
Release		This is the value for the release character to be used when generating this type tree. For a list of available symbols, click the browse button. The Symbols dialog appears, allowing you to select the desired value. The default value is an exclamation point (**!**).
Override Column Definitions		(For ODBC only) This check box indicates whether the database driver (with the check box disabled) or a command line override (with the check box enabled) is used to interpret columns. The default value is disabled. For more information about this field, see the adapter-specific documentation in each adapter reference guide.
Generate type trees in 1.4.0 format		(For Oracle adapter only) This check box determines whether you can generate type trees that are compatible with Version 1.4.0 maps. You might enable this functionality if you have legacy 1.4.0 maps that you want to maintain for some reason. The default value is disabled. For more information about this field, refer to adapter-specific documentation in each adapter reference guide.

Field	Description
Represent date/time columns as text items	This check box determines whether to automatically format this information in its database-specific date and time format (enabled) or as a text string (disabled). If you enable this check box (generating date and time as a text string), you may have to use either the `TEXTTODATE` or `TEXTTOTIME` function in a map rule to convert the text string to the database-specific date and time format. If you are generating new type trees, make sure this check box is disabled. The default is disabled. **Note:** Because of adapter-specific differences in date and time formats, see the adapter-specific instructions in each adapter reference guide.
Treat text item as	This list displays the languages available in which to handle text items. The default value is **Western**.

To generate a type tree from a stored procedure:

1. In the Navigator, highlight the icon of the database containing one or more stored procedures from which you want to generate a type tree.
2. From the **Database** menu, click **Generate Tree From → Procedure**.

   or

   In the Navigator, right-click the database and select **Generate Tree From → Procedure**.

   The Generate Type Tree from Stored Procedures dialog appears, including a list of stored procedures.
3. If you need to add a stored procedure because it is not displayed in the **Stored Procedures** list, you must insert it following the instructions provided in the Note below.

   or

   If you see one or more stored procedures in the **Stored Procedures** list that you want to select, if it is one, select it; if it is multiple procedures, press either **Shift** or **Ctrl** and click the desired stored procedures.

   Note: The list of stored procedures includes stand-alone procedures by default. (Procedures that are parts of a package are not listed.) If you need a procedure that is part of a package (whether this functionality is available is dependent upon your database), you may need to insert it into this list before selecting it. To insert a packaged stored procedure, right-click in the list, select **Insert**, and type the desired name. Use the following format: *schema.package.procedure*.
4. To create a new type tree, in the **File name** field, specify the path and name (with an **.mtt** extension).

   or

   Click the browse button to select a type tree from the Save As dialog. Highlight it and click **OK**.
5. Specify the remaining options as desired. Refer to context-sensitive help for field-specific information.
6. Click **Generate.**

   The Database Interface Designer and Type Tree Maker produce a type tree corresponding to the selected stored procedure(s).

The generated type tree is represented in the Navigator with the stored procedure icon next to the name.

From a message queue (Oracle AQ)

Use the Database Interface Designer to generate type trees for a message on an Oracle AQ message queue. An example of the Generate Type Tree from Queues dialog follows, along with a table describing its fields and their descriptions.

Field		Description
Database Name		This is the name of the database from which you want to generate a type tree. (This name is automatically inserted based upon your selection in the Navigator.)
Tables/Views		This list displays all of the queues that have been defined in this database. Select one or more from this list. If the queue you want is not displayed in this list, add it by right-clicking in the list, selecting **Insert**, and typing the desired name.
File name		This field displays a system-generated file name based upon the database selection in the Navigator. The default value is the database name with a type tree extension (**.mtt**) and a default path. To change any of this information, click the browse button.
Overwrite file		This check box determines whether you want to replace the existing type tree file (MTT) upon saving it. The default value is disabled.
Override type		This check box determines whether to replace existing types with matching, newly generated types and to add any non-matching new types to the type tree file (MTT). This is useful when you want to add types for additional stored procedures into an existing type tree. The default value is enabled.
Row group format		This list displays options to determine the row format in the generated type tree. The default value is **Delimited**.
	Delimited	Select this option to create rows delimited with a pipe character (|) or some other user-defined selection. This is the preferred option if you are using large column widths where the values may be smaller than the width of the columns. The pipe character is the default selection.

Field		Description
	Fixed	Select this option to create rows of fixed size in the generated type tree. There are certain situations in which you would want to select **Fixed**. • The query result contains a large number of rows where each row consists of small, fixed-length fields. In this situation, delimiters are unnecessary and they also consume memory. • You need to append fields in the output. This may be true if multiple fields have some meaning when concatenated together.
Delimiter		This is the value for the delimiter to be used when generating this type tree. For a list of available symbols, click the browse button. The Symbols dialog appears, allowing you to select the desired value. The default value is a pipe character (|).
Terminator		This is the value for the terminator to be used when generating this type tree. For a list of available symbols, click the browse button. The Symbols dialog appears, allowing you to select the desired value. The default value is **<CR><LF >**.
Release		This is the value for the release character to be used when generating this type tree. For a list of available symbols, click the browse button. The Symbols dialog appears, allowing you to select the desired value. The default value is an exclamation point (**!**).
Override Column Definitions		(For ODBC only) This field is not available for message queues.
Generate type trees in 1.4.0 format		(For Oracle adapter only) This check box determines whether you can generate type trees that are compatible with Version 1.4.0 maps. You might enable this functionality if you have legacy 1.4.0 maps that you want to maintain for some reason. The default value is disabled. For more information about this field, refer to adapter-specific documentation in each adapter reference guide.
Represent date/time columns as text items		This check box determines whether to automatically format this information in its database-specific date and time format (enabled) or as a text string (disabled). If you enable this check box (generating date and time as a text string), you may have to use either the `TEXTTODATE` or `TEXTTOTIME` function in a map rule to convert the text string to the database-specific date and time format. If you are generating new type trees, make sure this check box is disabled. The default is disabled. **Note:** Because of adapter-specific differences in date and time formats, see the adapter-specific instructions in each adapter reference guide.
Treat text item as		This list displays the languages available in which to handle text items. The default value is **Western**.

To generate a type tree from a queue:

1. In the Navigator, select the icon of the database from which you want to generate a type tree.
2. From the **Database** menu, click **Generate Tree From → Queue**.

   The Generate Type Tree from Queues dialog appears, including a list of queues.
3. If you need to add a queue because it is not displayed in the **Tables/Views** list, you must insert it following the information in the Note below.

   or

   If you see one or more queues in the **Tables/Views** list that you want to select, if it is one, select it. If it is multiple queues, press either **Shift** or **Ctrl** and click the desired queues.
4. To create a new type tree, in the **File name** field, specify the path and name (with an **.mtt** extension).

   or

   Click the browse button to select a type tree from the Save As dialog. Highlight it and click **OK**.
5. Specify the remaining options as needed.
6. Click **Generate**.

   The Database Interface Designer and the Type Tree Maker produce a type tree corresponding to the selected queue(s).

   Note: Type trees generated from queues are not graphically represented in the Navigator.

From a query

When generating a type tree from a query, information is used from your database and from the Database Interface Designer to create the type tree. If you change the definition of your query in the Database Interface Designer or edit that database definition, you must generate a new type tree to reflect the new information.

When attempting to generate a type tree, if your query is incorrectly specified, an error message from your database is received and a message displays, indicating that the column definitions for that query could not be accessed. In this scenario, a type tree is not generated.

Generating a type tree provides a way to test the syntax of your select statement. If you can successfully generate a type tree from a query, you know that the query has valid syntax.

An example of the Generate Type Trees From Query dialog follows, along with a table describing its fields and their descriptions.

Field	Description
File name	This field displays a system-generated file name based upon the database selection in the Navigator. The default value is the database name with a type tree extension (**.mtt**) and a default path. To change any of this information, click the browse button.

Field	Description
Overwrite file	This check box determines whether you want to replace the existing type tree file (MTT) upon saving it. The default value is disabled.
Override type	This check box determines whether to replace existing types with matching, newly generated types and to add any non-matching new types to the type tree file (MTT). This is useful when you want to add types for additional stored procedures into an existing type tree. The default value is enabled.
Row group format	This list displays options to determine the row format in the generated type tree. The default value is **Delimited**.
Delimited	Select this option to create rows delimited with a pipe character (|) or some other user-defined selection. This is the preferred option if you are using large column widths where the values may be smaller than the width of the columns. The pipe character is the default selection.
Fixed	Select this option to create rows of fixed size in the generated type tree. There are certain situations in which you would want to select **Fixed**. • The query result contains a large number of rows where each row consists of small, fixed-length fields. In this situation, delimiters are unnecessary and they also consume memory. • You need to append fields in the output. This may be true if multiple fields have some meaning when concatenated together.
Delimiter	This is the value for the delimiter to be used when generating this type tree. For a list of available symbols, click the browse button. The Symbols dialog appears, allowing you to select the desired value. The default value is a pipe character (|).
Terminator	This is the value for the terminator to be used when generating this type tree. For a list of available symbols, click the browse button. The Symbols dialog appears, allowing you to select the desired value. The default value is **<CR><LF >**.
Release	This is the value for the release character to be used when generating this type tree. For a list of available symbols, click the browse button. The Symbols dialog appears, allowing you to select the desired value. The default value is an exclamation point (**!**).
Override Column Definitions	(For ODBC only) This check box indicates whether the database driver (with the check box disabled) or a command line override (with the check box enabled) is used to interpret columns. The default value is disabled. For more information about this field, see the adapter-specific documentation in each adapter reference guide.

Field	Description
Generate type trees in 1.4.0 format	(For Oracle adapter only) This check box determines whether you can generate type trees that are compatible with Version 1.4.0 maps. You might enable this functionality if you have legacy 1.4.0 maps that you want to maintain for some reason. The default value is disabled. For more information about this field, refer to adapter-specific documentation in each adapter reference guide.
Represent date/time columns as text items	If you enable this check box (generating date and time as a text string), you may have to use either the `TEXTTODATE` or `TEXTTOTIME` function in a map rule to convert the text string to the database-specific date and time format. If you are generating new type trees, make sure this check box is disabled. The default is disabled. **Note:** Because of adapter-specific differences in date and time formats, see the adapter-specific instructions in each adapter reference guide.
Treat text item as	This list displays the languages available in which to handle text items. The default value is **Western**.

To generate a type tree from a query:

1. In the Navigator, highlight the icon of the query from which you want to generate a type tree.
2. From the **Query** menu, select **Generate Tree**.

   or

   In the Navigator, right-click the query and select **Generate Tree**.

   The Generate Type Tree from Query dialog appears, displaying the name of the selected type tree.
3. To create a new type tree, in the **File name** field, specify the path and name (with an **.mtt** extension).

   or

   Click the browse button to select a type tree from theSave As dialog. Highlight it and click **OK**.
4. Specify the remaining options as desired. Refer to context-sensitive help for field-specific information.
5. Click **OK**.

   The Database Interface Designer and the Type Tree Maker produce a type tree corresponding to the selected database query.

The generated type tree is represented in the Navigator with the query icon displayed next to the name.

Printing reports

You can print reports including detailed information about each specified database and query contained in an MDQ file. Reports contain information about the databases and queries that you designate. They can also include the time when the report was printed, the full path name of the MDQ file, the database definition, and definitions of each query and stored procedure.

You can only specify one database at a time.

To print a report:

1. Highlight the database within an MDQ file for which you want to print the information.
2. From the **File** menu, select **Print**.

   or

   In the Navigator, right-click the icon of the database and select **Print**.

   The Print dialog appears.
3. To print all of the entities displayed, skip to the next step.

   or

   To select certain entities to not be printed, disable the appropriate check box next to the entity name.
4. To use the default font for the report, skip to Step 6.

   or

   To change the font for the report, click **Font**.

   The Font dialog appears.
5. Change the settings as desired and click **OK**.

   The Font dialog is closed and you are returned to the Print dialog.
6. Click **OK**.

   The Print dialog appears.
7. Make any changes as desired and click **OK**.

   The report is printed.

Database Interface Designer trace files

The Database Interface Designer trace is a facility that produces information about accessing your database from the Database Interface Designer and then logs this information to a file. This can be useful in determining problems encountered when generating type trees or setting update keys using the Database Interface Designer.

If trace is enabled, a Database Interface Designer trace file (**.dbl**) is automatically created when you generate a type tree. This trace file is placed in the same directory as the MDQ file. The newly created Database Interface Designer trace file is named using the full name of the MDQ file with a **.dbl** extension. For example, if your MDQ file is named **Orders.mdq**, the trace file is named **Orders.dbl** and is located in the same directory. If the MDQ file has been newly created, the trace file is named its assigned name (**Database_QueryFile*n*.dbl**, where ***n*** represents the assigned sequential number) and resides in the directory in which the Database Interface Designer is installed.

Database Interface Designer trace

You can enable or disable trace for any MDQ file listed in the Navigator.

To enable or disable trace:

Note: Because this is a toggle command, use the same procedure for both enabling and disabling trace.

1. In the Navigator, highlight the MDQ file for which you want to enable a trace.
2. From the **File** menu, select **Trace**.

Viewing Database Interface Designer trace files

After trace has been enabled and a Database Interface Designer trace file has been generated, the contents of this file can be viewed in a trace window in the Database Interface Designer.

To view trace files:

1. In the Navigator, highlight the MDQ file for which you want to view the results of a trace.
2. From the **View** menu, select **Trace File**.

   A trace window appears, showing the results of the last trace.

   Note: If a trace file has not yet been generated, a blank window appears.
3. To define how you want to view multiple trace windows simultaneously, from the **Window** menu, enable either **Cascade**, **Tile Horizontally**, **Tile Vertically** or **Arrange Icons**.

In addition to the Database Interface Designer trace that is applicable only when working in the Database Interface Designer environment, you can also generate an additional database trace file that records activities occurring during map execution.

Finding text in trace files

Use the Find command to locate specific text in a Database Interface Designer trace window. This is helpful when analyzing errors or searching for specific information. The Find command is implemented as a result of values entered in the Find dialog. An example of this dialog follows, along with a table describing its fields and their descriptions.

Field	Description
Find what	Enter the text upon which to search. The default value is blank.
Match whole word only	This check box determines whether to use the value entered in the **Find what** field to match only whole words against in the trace file. (For example, if this was enabled and **text** was your find value, `texts` would not be a match.). The default value is disabled.

Field		Description
Match case		This check box determines whether to use the value entered in the **Find what** field to exactly match its case in the trace file. (For example, if this was enabled and **Text** was your find value, `text` would not be a match.) The default value is disabled.
Direction		Select the option to determine the direction of the find, that is whether to start at the beginning point and search backward (**Up**) or forward (**Down**). The default selection is **Down**.
	Up	Select this option to begin the find at this point and go backwards in the trace file. The default value for this is cleared.
	Down	Select this option to begin the find at this point and go forward in the trace file. The default value for this is selected.

To use the Find command:

1. Select the trace window containing the trace file upon which you want to perform a text search.

   Note: Ensure the trace window containing the desired trace file is the active window.

   or

   From the **Window** menu, select the name of the trace file.
2. From the **Edit** menu, select **Find**.

   The Find dialog appears.
3. In the **Find what** field, enter the text you want to find.
4. Make other selections as desired.
5. Click **Find Next**.

   If a match is found, the text is highlighted in the file in the trace window. Otherwise, there may be a message asking if you want to continue searching.
6. Repeat Step 5 until you have finished your search.
7. Click **Cancel**.

Chapter 4. Database type trees

Use the Database Interface Designer or **mtsmaker** to generate a type tree to be used in a map. You can also define your own type tree using the Type Designer. However, for the database adapters to correctly process your database data, your type tree must conform to the format described in this chapter. For information about using **mtsmaker** to generate type trees, see the Resource Adapters documentation.

Although you can manually create a type tree for a table or a query using the Type Designer, it is not recommended.

The following examples represent type trees generated by the Database Interface Designer from two queries (one SELECT and one calls a stored procedure), a table, a stored procedure, and a message queue.

As shown in these examples, the type trees generated from tables, views, queries, and stored procedures are very similar, while the type tree generated for message queues differs slightly.

In these example type trees:

- The type tree named **ActvProj.mtt** was created for a query that referenced a Microsoft Access table.
- The type tree named **Stores.mtt** was created for a table in a Microsoft SQL Server database.
- Two type trees were created for stored procedures in an Oracle database.
  - The **PymtInfo.mtt** type tree was generated for a query called **GetPaymentInfo** that calls an Oracle stored procedure to be used as a map data source.
  - The **ApplyPmt.mtt** type tree was generated for the **SCOTT.APPLY_PAYMENT** stored procedure to be used as a map output.
- The type tree named **Paulmsg.mtt** was generated from a message queue on the **PAUL_MSG_QUEUE**.

Table and query type tree structure

Type trees generated for a query and a table closely resemble one another.

Each type tree generated by the Database Interface Designer or **mtsmaker** for a database table or query (or a query that calls a stored procedure) will have the following standard characteristics:

- The root of the generated type tree is named **Data**.
- The generated type tree contains a category whose name corresponds to any one of the following:
  - the name of the table in the database
  - the name of the query in the MDQ file
  - the name specified using the Category Type (`-N`) parameter for **mtsmaker**

    The types defining the columns and rows returned by the database adapter stem from this type. In the examples, the categories are **ActiveProjects** and **stores**.

- Stemming from the category named for the table or query are types that define the contents of the table or the results of query execution.
- Each column in the database table or in the results of the query must have a corresponding item type with a name identical to that of the column in the database. These types stem from an item type (shown in each example) named **Column**.
- A group type named **Row** represents a single row of data-either the result of a SELECT statement or a row to be inserted/updated in a table or view.
- An implied group represents a collection of rows reflecting the results of a query, table, or view. This group is named either **DBSelect** for a query or **DBTable** for a table or view (or **DBProcedure** for a stored procedure.

Special characters in type names

Some characters used in column names are not valid within type names. The Database Interface Designer replaces these characters when it creates the type name for the column. If you create a type tree without using the Database Interface Designer or **mtsmaker**, you need to replace these characters in the type name.

Again, if you use the Database Interface Designer, these conversions are automatically performed. If you use **mtsmaker**, you must manually convert these characters.

Invalid Character(s)	Conversion to Type Name
space or hyphen (-) example: `My Column Name`	replaced with underscore (_) becomes: `My_Column_Name`
punctuation (other than `~ # % \ ` ? _`) example: `^^Total^Amount`	replaced with pound sign (#) becomes: `##Total#Amount`
digit (as first character of column name) example: `1st-Game-of-2`	prefixed with pound sign (#) becomes: `#1st_Game_of_2`

Characteristics of the DBSelect or DBTable type

In each type tree generated by the Database Interface Designer or **mtsmaker**, an implied group type represents the results of executing the query or the contents of the table.

- **DBSelect**

  The type tree defines a query used as a data source for a map.
- **DBTable**

  The type tree defines a table used as a data target for a map.

The **DBSelect** or **DBTable** type is an implied group with a series of row objects as its only component.

Components and format of the row type

As its name implies, the **Row** group represents a row (or tuple). The **Row** type is a group with an explicit format. The syntax of that format is based upon the options specified when the tree was created from the Database Interface Designer or the value passed to **mtsmaker** using the Format mtsmaker command (`-X`).

see the following table for information about specifying the value for the **Row group format** list when generating a type tree for a query or table.

Using Information

Database Interface Designer
Database/query files

mtsmaker
Resource Adapters documentation

There are two different row group formats:

- delimited row group format
- fixed row group format

You can specify the syntax for each of these when generating the type tree.

Delimited row group format

If you select **Delimited** as the value in the **Row group format** list in either the Generate Type Tree from Tables dialog, the Generate Type Tree from Query dialog, or the Generate Type Tree from Stored Procedures dialog, the Properties window in the generated type tree indicates that the **Group Subclass → Format** is an explicit group defined as having a delimited syntax.

In the **Group options** pane, which is located in the dialog for generating the type tree, you can specify the **Delimiter**, **Terminator**, and **Release** setting values to be used in the type tree. The default values are |, None, <CR><LF>, and !, respectively. The **Initiator** value cannot be specified; it is always set to None.

When using the `DBLOOKUP`, `DBQUERY`, `GET` and `PUT` map functions with the database adapter, the adapter will use the default values for the **Delimiter**, **Terminator**, and **Release** settings. This is because the data passed to and from the adapter is, in this case, not represented by the type trees generated in the Database Interface Designer. The adapter will use the default values for the **Delimiter**, **Terminator**, and **Release** settings since it cannot access this information from a type tree.

Each component of **Row** is a type representing a column. These components appear in the component list in the same sequence as they appear in the query or table, as opposed to the type tree, which lists them in alphabetical order. In a delimited **Row**, each of the columns is defined as optional within the **Row** by specifying a range of (0:1) for each column component.

Fixed row group format

If you select **Fixed** as the value in the **Row group format** list in either the Generate Type Tree from Tables dialog, the Generate Type Tree from Query dialog, , or the Generate Type Tree from Stored Procedures dialog, the Properties window in the generated type tree indicates that the **Row** type is an explicit group defined as having a fixed syntax.

In the **Group options** pane, which is located in the dialog for generating the type tree, you can specify the **Terminator** setting values to be used in the type tree. The default value is <CR><LF>.

When using the `DBLOOKUP`, `DBQUERY`, `GET` and `PUT` map functions with the database adapter, the adapter will use the default values for the **Delimiter**, **Terminator**, and **Release** settings. This is because the data passed to and from the adapter is, in this case, not represented by the type trees generated in the Database Interface Designer. The adapter will use the default values for the **Delimiter**, **Terminator**, and **Release** settings since it cannot access this information from a type tree.

Defining the column type(s)

Each named column in the database has a corresponding item type under the **Column** type in the type tree with the same name. For example, if the **DEMO.DEPARTMENT** table in the database has three columns (**DEPARTMENT_ID**, **LOCATION_ID**, and **NAME**), the type tree will have three identically named items (subject to restrictions described in ″Special Characters in Type Names″) under the **Column** type.

Each item is defined according to the data type and length information returned to the database adapter by the database driver. For more information about the correspondence between database data types and item formats, see the specific adapter reference guides.

The minimum size of item types for columns in a delimited **Row** is 0 unless otherwise specified by the database driver. The minimum size of item types for columns in a fixed **Row** equals the maximum size.

Binary column types

If a database column corresponds to a binary type, your type tree contains additional types so that the binary data can be correctly interpreted. The method used to define binary columns allows a distinction to be made between a **NULL** column and a binary value of 0.

A category called **SizedGroup** is added to the tree. This category has an item subtype called **Sizeof** and group subtypes with the same names as the item types for each binary column.

The following changes are made to the type tree to account for columns representing binary data:

- Two variable length binary items are created as subtypes of the **Column** type to represent the value of the two binary type columns.
- The **Sizeof** item is a subtype of the **SizedGroup** category and is defined as a character integer item.
- Also, as a subtype of the **SizedGroup** category, two groups given the same name as the binary column type are created, each group including two components. The first component of each group is the **Sizeof** item; the second, the variable length binary item (either).
- The formats of those group types defined as subtypes of the **SizedGroup** category are defined in the same way as the format for the **Row** group type.

  If the type tree has a delimited **Row** group format, the group types for the binary columns are also defined as delimited groups with the pipe character (|) as an infix delimiter. In this example, the **Sizeof** component is defined as being required in the component list. However, the binary item type is optional (with a range of 0:1).

  You cannot generate a fixed **Row** group format type tree for a table or query containing variable length binary data.

  If the type tree has a fixed **Row** group format, the group types for the binary columns are also defined as fixed groups. In this situation, both the **Sizeof** component and the binary item type are required.
- In the group type component list, the **Sizeof** component has the **Sized** attribute, indicating that the **Sizeof** item contains the size of the binary item that follows it.

When mapping to a table or view containing binary columns, use the `SIZE` function to assign the size of the binary data to the **Sizeof** data item.

- The components for the **Row** type are the group types under the **SizedGroup** category for the binary columns.

Column types for expressions

Each constant or function specified in a query is given a name based upon the name returned from the database driver to the database adapter. If the database driver does *not* return a name for a column in the results, the item type created for the column is given a name beginning with Expr and is concatenated with a number beginning with 1000 for the first constant or function, 1001 for the second, and so on.

For example, the type tree generated for the following query

```
SELECT MIN(salary), MAX(salary) FROM Employee
```

would have a **Row** with two columns representing the result of the SQL MIN and MAX functions. In the type tree, the items created to represent the results of these functions would be Expr1000 and Expr1001 for the MIN and MAX functions, respectively.

Specifying column aliases

When you want to control the name of the generated item to represent a particular column or expression in a query, use the AS keyword to specify an alias for that column.

For example, you could specify the query as:

```
      select min(salary) as min_salary, max(salary) as max_salary from employee
```

which would result in a type tree containing a **Row** with two columns representing the result of the SQL MIN and MAX functions. In the type tree, the items created to represent the results of these functions would be **min_salary** and **max_salary**. An example of this follows.

You can also use the AS keyword to specify an alias for a column having a name containing any of the characters that are invalid in a type name. Or, you can specify an alias for a column having a name longer than 32 characters (which is the limit for the size of a type name).

The following example results in a type tree having a **Row** with three columns.

```
Select SomeReallyReallyReallyLongColumnName AS
SomeLongColumnName,
      1st-Payment AS Payment#1,
      2nd-Payment AS Payment#2
      from some_table
```

The items created to represent the results of this query would be **SomeLongColumnName**, **Payment#1**, and **Payment#2**.

Stored procedure type tree structure

The type trees generated by the Database Interface Designer for stored procedures used for outputs from a map are slightly different from the ones generated for tables, queries, and queries calling stored procedures (that are used as the source of data for a map).

Major differences between a type tree for a stored procedure that will be used as output and a type tree for a table that will also be used as output are:

- Whereby table and query type trees have a **Column** item type from which stems all of the individual column types, the type tree for a stored procedure has an **Argument** item type from which stems item types representing each of the arguments passed to the stored procedure.
- Instead of a **Row** group, the type tree for a stored procedure has a **ProcedureCall** type. The **ProcedureCall** group represents the set of arguments passed to the stored procedure for each execution of the procedure.

  The **ProcedureCall** group is defined in the same way as a **Row** type for a table or a query in the type tree. Its group format is determined by the value selected in the **Row group format** list in the Generate Type Tree from Tables dialog or the Generate Type Tree from Query dialog, respectively. The selected group format determines the terminator and release characters.
- Similar to the **DBTable** or **DBSelect** types in type trees for tables or queries, the type tree for a stored procedure has a **DBProcedure** type, which is an implied group consisting of a series of **ProcedureCalls**. The stored procedure is called once for each **ProcedureCall** in the **DBProcedure** output.

  For information about using stored procedures as a data target, see″Using stored procedures.″ For information about generating type trees for a stored procedure, see″From a stored procedure″.

Oracle AQ Message type tree structure

The type trees generated by the Database Interface Designer for Oracle AQ messages used as output from a map are different from those generated for tables, queries, or stored procedures. For information about these type trees, see the Oracle AQ Adapter documentation.

Chapter 5. Database sources and targets

This chapter explains how to use the Map Designer to configure a map having a database source or target. For detailed information about using the Map Designer and defining sources and targets in input and output cards, see the Map Designer documentation.

Using a database as a source

After you have used the Database Interface Designer to define a query for a database, you can use that query as an input source.

Note: You can use either a standard SQL SELECT query or a query using a stored procedure. For specific information about using stored procedures, refer to ″Using stored procedures.″

To use a database query as a source:

1. Define the database using the Database Interface Designer.
2. Define a query to extract data from the database.
3. Generate a type tree for the query.
4. Use the Map Designer to create a map with an input card referencing the query.

Defining a database source in the Map Designer

When you generate the type tree from a query, one of the types in the tree is a group representing the results from executing the query (for example, **DBSelect**). Select this type as the input card type when defining an input card for the executable map.

After you have defined the query and have generated the type tree, using a database as an input is very similar to using a file as input.

Using the Map Designer, specify **Database** as the **GET → Source** setting in the input card and supply the name of the MDQ file containing the definition of the database and query you want to use. see the following procedure and example.

To define a database as a data source:

Note: For more information about the map settings in this procedure, see the Map Designer documentation. This procedure specifically addresses database-specific parameters and settings.

1. In the Map Designer, when defining the settings for **SourceRule**, select **Database** as the **GET → Source** setting.

   The settings displayed in the **Input Card** dialog change to display the database adapter settings for a source.
2. From the **Card** menu, select **New**.

   The Add Input Card dialog appears.
3. For the **CardName** setting, enter a name for the card that describes the data object represented by this card.

4. For the **TypeTree** setting, select the type tree containing the group type that defines the desired query.
5. For the **TypeName** setting, select the group type from the type tree that defines the desired query (for example, `DBSelect`).
6. For the **TypeName** setting, select the group type from the type tree that defines the desired query (for example, `DBSelect`).
7. Specify the **FetchAs**, **WorkArea**, **FetchUnit**, and all **Backup** settings as desired. For detailed information about these settings, see the Map Designer documentation. For database adapter-specific information about source settings, see the adapter-specific reference guide.
8. For the **GET → Source** setting, select **Database** from the list.

   The **SourceRule** settings change to display the database adapter settings.
9. Specify the values as desired in the adapter settings and click **OK**.

Database GET> Source settings

After you select **Database** as the **GET → Source** setting and the database adapter settings become available for an input, configure the settings for a source that uses the database adapters.

Many of the adapter settings provide transactional control and connection management that allow you to control the connections made to your database. For information about specifying these particular **GET → Source** and **SourceRule** settings, see"Database Connections and Transactions" .

GET> Source> Command setting

The **GET> Source> Command** setting is used to enter the applicable database adapter commands for this source. For general information about these settings, see the Map Designer documentation. For specific information about using database-specific adapter commands, see the Resource Adapters documentation.

Using a database as a target

After using the Database Interface Designer to define tables, views, message queues, or stored procedures for a database, use the database to specify the database adapter as the data target.

To use a database table, view, message queue, or stored procedure as a target:

1. Define the database using the Database Interface Designer.
2. Generate a type tree from a table, view, stored procedure, or message queue.
3. Using the Map Designer, create a map with an output card that specifies the database table, view, message queue, or stored procedure as the data target.

Defining a database target in the map designer

When you generate the type tree for a table, view, message queue, or stored procedure, one of the types in the tree is a group representing the contents of the table, view, message queue, or stored procedure. For example, a type tree generated for a table or view is named **DBTable**. Use the Map Designer to select the appropriate type representing the contents as the output card type of the executable map.

After you have defined the database and have generated the type tree, using a database as an output is very similar to using a file as output.

Use the Map Designer to specify **Database** as the **PUT → Target** setting in the output card at the executable map level. Provide the name of the MDQ file containing the definition of the database, along with the table, view, message queue, or stored procedure to be used.

To define a database as a target:

Note: For more information about the map settings in this procedure, see the Map Designer documentation. This procedure specifically addresses database-specific parameters and settings.

1. In the Map Designer, select the To window.
2. From the **Card** menu, select **New**.

   The Add Output Card dialog is displayed, an example of which follows.
3. For the **CardName** setting, enter a name for the card.
4. For the **TypeTree** setting, select the type tree file containing the group type that defines the content of the desired output.
5. For the **TypeName** setting, select the group type from the type tree that defines the desired output (for example, **DBSelect**).
6. Specify all **Backup** settings as desired. For detailed information about these settings, see the Map Designer documentation.
7. For the **PUT → Target** setting, select **Database** from the list.

   The settings displayed in the Output Card dialog change to display the database adapter settings for a target.

   For database adapter-specific information about target settings, see the adapter-specific reference guide.
8. Specify the values as desired in the adapter settings and click **OK**.

Database PUT > Target settings

After you select **Database** as the **PUT → Target** setting and the database adapter settings for a target become available in the output card, you can configure the settings for a target that uses the database adapters.

Many of the adapter settings provide transactional control and connection management that allow you to control the connections made to your database. For information about specifying these particular **PUT → Target** and **TargetRule** settings, see ″Database connections and transactions″ .

PUT > Target > Command setting

The **PUT → Target → Command** setting is used to enter the database adapter commands applicable for this target. For general information about these settings, see the Map Designer documentation. For specific information about using database-specific adapter commands, see the adapter-specific documentation.

Database connections and transactions

WebSphere Transformation Extender maps allow superior transactional control and connection management by providing settings that allow the scope of transactions to be controlled by the definitions provided in a map. It is important to understand how these connections to databases are controlled and how this relates to transactional scope.

Transactional control

Scope is one of the settings that may be specified for a source (**GET** → **Source** → **Transaction** → **Scope**) or target (**PUT** → **Target** → **Transaction** → **Scope**) in a map. For information about **Scope** settings, see the Map Designer documentation.

There is one restriction for **Scope**. If the value of the **SourceRule** → **FetchAs** setting in an input card is set to **Burst**, the **Scope** setting is always **Map**. The adapter scope is determined to be **Map** because the context of the SELECT statement cannot be maintained after the transaction has been terminated through a commit or rollback. Therefore, if **Burst** is the value for the **SourceRule** → **FetchAs** setting in an input card, any value in the **Scope** setting as well as any adapter command (`-CCARD` or `-CSTMT`) specified in the **GET> Source** → **Command** setting is ignored.

Additionally, the global transaction management functionality is available for certain adapters by using the Global Transaction Management (`-GTX`) adapter command in the **Command** setting. For more information about this functionality, see the Global Transaction Management documentation.

Database connection management

Connection management results in fewer connections being made to the database, which may improve performance. The WebSphere Transformation Extender engine reuses existing connections whenever possible. With global transaction management for those adapters able to take advantage of this functionality, connections have an even bigger role in maintaining data integrity. For more information about global transaction management, see the Global Transaction Management documentation.

There are two situations in which connection sharing occurs:

- between cards and maps within a map
- between multiple maps running serially within the Launcher

When connection sharing occurs between cards and maps within a map, the connection is described as active as long as there are one or more cards or rules that access a database and that have yet to be committed or rolled back. (For example, an active transaction exists.) When there are no such cards or rules, the connection is inactive, yet still alive.

When using a `DBLOOKUP`, `DBQUERY`, `GET` or `PUT` function, the **Transaction** → **Scope** setting defaults to **Map** when the map **SourceRule** → **FetchAs** setting is set to **Integral**. The **Transaction** → **Scope** setting will change to **Burst** when the map **SourceRule** → **FetchAs** setting is set to **Burst**; the **Transaction** → **OnFailure** setting is always **Rollback**.

Connection factors

The factors determining whether a connection is reused are the following:

- the database type
- the connection string (if applicable), datasource, or database name
- the user ID
- the **OnFailure** setting (**Commit** or **Rollback**)
- the **Transaction** → **Scope** setting
- the **SourceRule** → **FetchAs** setting
- whether `-CCARD` or `-CSTMT` are specified within the adapter's command

Also, whether a connection will be part of a global transaction is determined by the usage of the Global Transaction (`-GTX`) adapter command in the **Command** setting.

Connection rules

Existing connections are reused whenever possible while adhering to certain connection rules. The rules are as follows:

- An inactive connection is reused if the database type, connection string, and user ID of the new card or rule match those of the previous card or rule that had established the connection.
- In addition to the database type, connection string and user ID, an active connection is reused only if the **OnFailure** and **Transaction → Scope** settings match in the previous card or rule and the new card or rule.
- If a card or rule has `-CSTMT` set within the adapter's command, this connection may be shared with any other card or rule that has `-CSTMT` set or that has an **Transaction → Scope** setting of **Card** (or `-CCARD` within the adapter's command).
- If **Burst** is the **SourceRule → FetchAs** setting for an active card, it (the card) is always executed in its own transaction. It establishes a connection that cannot be shared by any other card for the duration of the map.

Connection example

A map has four database input cards. Assume the connections specified (datasource, userID, and so on) are the same for each.

In this example,

- A connection will be made and a transaction started for **Card 1**.
- Because **Card 2** has **Transaction → Scope** set to **Card**, this initial connection cannot be shared. Therefore, another connection will be made for this card.
- For **Card 3**, because its **OnFailure** setting differs from the setting for **Card 1**, the connection established in **Card 1** cannot be reused. However, the connection established for **Card 2** is inactive (because **Transaction → Scope** was set to **Card** and the card has completed) and, therefore, will be reused for **Card 3**. A new transaction is started on this existing connection.
- **Card 4** is able to share the connection established by **Card 1** and can be part of the same transaction because all of its settings match.

If the map fails, the transaction containing both **Card 1** and **Card 4** will be rolled back and the transaction for **Card 3** will be committed.

Chapter 6. Updating database tables

This chapter discusses how you can designate specific columns in your database to be updated with data produced by a mapping operation.

Using key and update columns

The data produced by a mapping operation can be inserted as new rows in a database table or can update only specific columns in a table as designated.

Designate particular columns in a table to be used as key columns determining whether output data updates an existing row or whether it is inserted as a new row in the table or view. In addition to specifying key columns, you can designate the columns in a table that will be affected by any update operation. You define the columns to update.

The term key in this usage does not necessarily mean that the column is part of a database primary or foreign key.

For any update to be performed, update mode must be specified for the target. This is accomplished by specifying the Update adapter command (`-UPDATE`) in the **PUT → Target → Command** settings using the Map Designer or Integration Flow Designer, or by passing it with the override execution command on the command line at execution time. For more information about how to enable update mode, refer to ″Specifying update mode″ .

Defining key and update columns

To update a table from a map, use the Database Interface Designer to specify the columns in a table to be used as key columns and to specifically specify the columns that will be updated.

To designate columns as key columns or columns to update:

1. In the Navigator, select the database containing the table for which you want to set update keys.
2. From the **Database** menu, choose **Set Update Keys**.

   The Set Table Update Key Columns dialog appears.
3. From the **Table name** list, select the table containing the columns to be updated.

   The columns in the table appear in the **Columns** list.
4. To set a column as a key column, select it from the **Columns** list and click the right arrow button associated with the **Key columns** list.

   The selected column moves to the **Key columns** list.
5. To designate a column as a column to update, select it from the **Columns** list and click the right arrow button associated with the **Columns to update** list.

   The selected column moves to the **Columns to update** list.
6. To specify all non-key columns as **Columns to update**, click the ″all″ button associated with the **Columns to update** list.

   All of the columns in the **Columns** list move to the **Columns to update** list.

7. To remove columns from either the **Key columns** list or the **Columns to update** list, select the column and click the associated ″delete″ button.

   The selected columns are moved back to the **Columns** list.

A generated type tree with specified update keys is represented in the Navigator under **Tables**.

Specifying update mode

To update a table, you must specify update mode using the Map Designer, Integration Flow Designer, or the command line. By specifying update mode in one of these ways, an update operation can be performed in which each row produced by your map is analyzed and the update is performed based upon the update keys and columns defined in the Database Interface Designer.

Using the Map Designer or Integration Flow Designer

After key columns and update columns are defined for a table, specify the table as a target in the **TargetRule** settings in the Map Designer or in the **Output(s)** settings in the Integration Flow Designer.

You must specify `-UPDATE` in the **PUT → Target → Command** setting so that your specified columns are automatically updated.

Using an adapter command at execution time

The update mode for a table can be specified at execution time by using the command line to pass the `-UPDATE` database adapter command. Specify OFF or ONLY to control the updating operation. For more information about using database-specific adapter commands, see the Resource Adapters documentation.

Update key columns

When you specify a column as a key column, the value in that column is used to determine whether a row produced by a mapping operation should be inserted as a new row into the table or should be used to update existing row(s) in a table.

- If the values in the rows generated by the mapping operation match the values in the key columns as defined in the Database Interface Designer, the specified update columns are updated.
- If the values in the rows generated by the mapping operation do *not* match the values in the key columns, the rows are inserted as new rows into the table or view as defined in the Database Interface Designer.

  Note: You can override the behavior defined in the Database Interface Designer by using the command line to specify the appropriate database adapter commands for the database adapter. For information about the database-specific adapter commands, see the Resource Adapters documentation.
- If more than one column is designated as a key column, the values generated by a map must match the values in each designated key column for the row to be updated.
- If your map generates a row with key column values that match more than one existing record, all of the matching records in the columns you have specified as update columns are updated with the new row values.

- If your map produces multiple rows with the same values for the key columns, you lose any updates that are made as a result of all but the last row. The updates produced by each row are overwritten by the updates from subsequent rows with the same key values.

Example using update columns

When only some columns in a table are specified as columns to be updated, the values corresponding to all other columns are ignored when the update statement is built and executed. For any row produced by the map, if the values for the update key columns do not match any of the existing rows in the table or view, the values of all columns will be inserted in the new row.

In the following example, a table (**PersonalInfo**) has the following key columns and update columns defined in the Database Interface Designer.

When this map runs, because `-UPDATE` is enabled for the output, the database adapter will first go through the results of the map and update all rows in the table matching the key columns in the output produced. Essentially, the following SQL statements are executed:

```
UPDATE PersonalInfo
 SET FirstName='Karl', LastName='March', PhoneNumber='(847)
555-1234'
 WHERE ID = 10
UPDATE PersonalInfo
 SET FirstName='Janice', LastName='Armstrong',
 PhoneNumber='(203) 555-9898'
  WHERE ID=14
```

In the first UPDATE statement, because this statement does not find any rows to update, the following SQL statement is executed.

```
INSERT INTO PersonalInfo VALUES (10,'Karl', 'March',
'(847) 555-1234', '999-88-7766')
```

This execution creates a new row in the table for **Karl March**-including values for the **ID**, **FirstName**, **LastName**, **PhoneNumber**, and **SSN** columns.

In the second UPDATE statement, because an existing row has an **ID** value of **14**, only the values of the **FirstName**, **LastName**, and **PhoneNumber** columns are updated because of the settings specified in the **Columns to update** list in the Set Table Update Key Columns dialog of the Database Interface Designer. In this example, **Janice Taylor**'s last name changed to **Armstrong** and her telephone number is changed to **(203) 555-9898**. Her social security number remains unchanged because it is not a column that has been designated for update.

Chapter 7. Database functions

This chapter discusses two functions (DBLOOKUP and DBQUERY) that are designed exclusively for use with databases. These functions can be used in component rules in the Type Designer and map rules in the Map Designer when creating a map to be used with a database.

Accessing database information in a map rule

In many cases, database information for a map will be one of the following:

- the results of a database query or a stored procedure defined as the data source for an input card
- the rows to update or insert into a table defined as the target for an output card.

However, there are certain situations in which you might not want to define the entire query as a data source or the table as a target. For example, you may have a very large table that is used as a source for cross-reference in your map. However, because of its size, reading in and validating all of the information as an input card is impractical. In other situations, you may want to call one map from another using the RUN function when one of the data sources is a dynamically built query.

Two functions provide the ability to access database information from within a rule:DBLOOKUP and DBQUERY. The difference between the two functions is subtle and is based upon how the data resulting from either function is returned.

Using DBLOOKUP and DBQUERY

A single-text-item is returned by either the DBLOOKUP and DBQUERY function. If your SQL statement is a SELECT statement, the DBLOOKUP or DBQUERY function returns the results of the query in the same format as a query specified in a map input card. If your SQL statement is anything other than a SELECT statement, these functions return NONE.

Both functions return the results of a query (SQL SELECT statement) in the same format as a query specified for a map input card, using the delimited row format. However, the DBLOOKUP function strips off the last carriage return/line feed. Because this information is stripped, it is easier to make use of a single value extracted from a database.

DBLOOKUP and DBQUERY execute an SQL statement within a rule against the database. The SQL statement can be any statement permitted by your database management system or database-specific driver.

There are two syntax methods that can be used to specify the arguments for these two functions: Syntax1 and Syntax2.

Syntax1 - using a static MDQ file

Use Syntax1 to execute a SELECT statement that retrieves a specific column value from a large table in a database using the value of another input, as opposed to having to define the entire table as an input card and using other functions such as

LOOKUP, SEARCHDOWN, or SEARCHUP. For more information about these functions, see the *Functions and Expressions documentation*.

If column data from a table or database varies because it may be based upon a parameter file, use Syntax2. For more information, refer to ″Syntax2 - using dynamic adapter commands″ .

Syntax1 is a three-argument syntax, examples of which follow:

```
DBLOOKUP
   (SQL_statement,mdq_file,database_name[:adapter_commands])
DBQUERY
   (SQL_statement,mdq_file,database_name[:adapter_commands])
```

Using this syntax, DBLOOKUP and DBQUERY use the following arguments.

Argument	Explanation	Must Be
SQL_statement	single-text-expression	a valid SQL statement
mdq_file	single-filename	a string literal
database_name	single-database-name	a string literal
adapter_commands	adapter-commands	a valid adapter command(s)

The following table describes these arguments in more detail.

Argument
Description

SQL_statement
SQL statement as a text string. It can be any valid SQL statement permitted by your database management system and supported by your database-specific driver.

In addition to a fixed SQL statement, this argument can be a concatenation of text literals and data objects, enabling the concatenation of data values into your SQL statement.

mdq_file
Name of a database/query file (MDQ) produced by the Database Interface Designer. It contains the definition of the database against which the SQL SELECT statement is to be executed. If the MDQ file is in a directory other than the directory of the map, the complete path must be specified.

Note: The MDQ file is accessed at map build time and is not needed at runtime.

database_name
Name of a database in the MDQ file as defined in the Database Interface Designer.

Note: This name is case-sensitive and must exactly match the name as defined in the Database Interface Designer.

adapter_commands
Name of an adapter command or commands. This name is an optional argument that appears after *database_name*, which allows you to specify database adapter commands in component rules in the Type Designer and map rules in the Map Designer.

Note: The rules are defined and compiled into your map at design time.

Note: All adapter commands (for example,-CS) can be either uppercase or lowercase, but not mixed case.

Syntax2 - using dynamic adapter commands

Use Syntax2 to execute a SELECT statement that retrieves a specific column value from a table or database when the database, table, or other database parameters may vary. This can be used when parameters are being supplied by a parameter file.

```
DBLOOKUP( SQL_statement,adapter_commands )
DBQUERY( SQL_statement,adapter_commands )
```

Using this syntax and the Syntax 2 formatting issues (″Syntax2 formatting issues″), DBLOOKUP and DBQUERY use the following arguments.

Argument
Explanation

SQL_statement
single_text_expression

adapter_commands

Either
-MDQ *mdq_file* -DBNAME *db_name*or
-DBTYPE *database_type* [*database_adapter_commands*]

The following table describes these arguments in more detail.

Argument
Description

SQL_statement
SQL statement as a text string. It can be any valid SQL statement permitted by your database management system and supported by your database-specific driver.

In addition to a fixed SQL statement, this argument can be a concatenation of text literals and data objects, enabling the concatenation of data values into your SQL statement.

adapter_commands
Either the -MDQ or the -DBTYPE adapter command can be used for this argument.

-MDQ This adapter command is followed by the name of the MDQ file produced by the Database Interface Designer. This MDQ file contains the definition of the database against which the SQL statement is to be executed. If the MDQ file is in a directory other than the directory of the map, the path must be specified.

The MDQ file name is followed by the -DBNAME adapter command and the name of the database in the MDQ file, as defined in the Database Interface Designer.

Using this syntax, the specified MDQ file must be available at run time.

-DBTYPE

This adapter command is followed by the type of database (for example, ORACLE or ODBC) followed, optionally, by database-specific adapter commands.

This syntax does not use an MDQ file because the database-specific adapter commands provide the information required to connect to the database. For more information about your particular database adapter and database-specific adapter commands, see the adapter-specific documentation in the adapter reference guides.

Syntax2 formatting issues

When using Syntax2 for DBLOOKUP or DBQUERY, your function must conform to these rules:

- All adapter commands (for example, -DBTYPE) can be either upper or lower case, but not mixed case.
- A space is required between the adapter command and its value (for example, -DT database_adapter).
- The order of the adapter commands is not important.

Examples

The section presents different DBLOOKUP and DBQUERY examples.

Example 1 - obtaining a single column value

Assume that you have a table named **PARTS** that consists of the following data:

PART_NUMBER	PART_NAME
1	1/4″ x 3″ Bolt
2	1/4″ x 4″ Bolt

Also assume that this database has been defined in a file named **mytest.mdq** using the Database Interface Designer. The name of the database, as specified in the MDQ file, is **PartsDB**. Notice the difference between the returned values from the execution of the following two Syntax1-formatted functions.

Function	Returns PART_NAME
DBLOOKUP (″SELECT PART_NAME from PARTS where PART_NUMBER =1″, ″mytest.mdq″, ″PartsDB″)	1/4″ x 3″ Bolt
DBQUERY (″SELECT PART_NAME from PARTS where PART_NUMBER =1″, ″mytest.mdq″, ″PartsDB″)	1/4″ x 3″ Bolt<cr/lf> where <cr/lf> is a carriage return followed by a line feed

Using Syntax2, you can also specify the DBLOOKUP or DBQUERY functions as in the following examples.

```
DBLOOKUP("SELECT PART_NAME from PARTS where PART_NUMBER =1",
         "-MDQ mytest.mdq -DBNAME PartsDB")
DBQUERY("SELECT PART_NAME from PARTS where PART_NUMBER =1",
        "-MDQ mytest.mdq -DBNAME PartsDB")
```

Note that both the MDQ file name and database name are specified.

The examples below use the Syntax2 format to specify the database type and the appropriate database-specific adapter commands (in this example, using the -DBTYPE, -CONNECT, -USER, and -PASSWORD commands for an Oracle database):

```
DBLOOKUP("SELECT PART_NAME from PARTS where PART_NUMBER =1",
         "-DBTYPE ORACLE -CONNECT MyDatabase
          -USER janes -PASSWORD secretpw")
DBQUERY   ( "SELECT PART_NAME from PARTS where PART_NUMBER =1",
          "-DBTYPE ORACLE -CONNECT MyDatabase
          -USER janes -PASSWORD secretpw")
```

Example 2 - using DBQUERY to obtain multiple columns or rows

Assume that you have a table named **PARTS** that consists of the following data:

PART_NUMBER	PART_NAME
1	1/4″ x 3″ Bolt
2	1/4″ x 4″ Bolt

Also assume that this database has been defined in a file named **mytest.mdq** using the Database Interface Designer. The name of the database, as specified in the MDQ file, is **PartsDB.**

Using the Syntax1 format, the following DBQUERY function:

```
DBQUERY ("SELECT * from PARTS", "mytest.mdq", "PartsDB")
```

returns:

```
1|1/4" x 3" Bolt<cr/lf>2|1/4" x 4" Bolt<cr/lf>
```

where <cr/lf> is a carriage return followed by a line feed.

Notice that if the same function was executed using DBLOOKUP, the results would be:

```
1|1/4" x 3" Bolt<cr/lf>2|1/4" x 4" Bolt
```

The difference between the two results is that the final carriage return/line feed is stripped off the end of the results of the DBLOOKUP function.

Using Syntax2, you can obtain the same results as in the previous DBQUERY function as shown in the following examples:

```
DBQUERY ("SELECT * from PARTS where PART_NUMBER =1",
         "-MDQ mytest.mdq -DBNAME PartsDB")
```

or:

```
DBQUERY         ("SELECT * from PARTS where PART_NUMBER =1",
          "-DBTYPE ORACLE -CONNECT MyDatabase -USER janes
          -PASSWORD secretpw")
```

Example 3 - using DBQUERY to provide map input to RUN function

In another example, assume that you have an input file containing one order record. To map that order to another proprietary format, you also have a **PARTS** table with pricing information for every part for every customer-a very large table. Rather than using the entire **PARTS** table as the input to your map, you might

include the RUN function with your Syntax1-formatted DBQUERY to dynamically select only those rows from the **PARTS** table corresponding to the customer in the order file, as follows:

```
RUN("MapOrder.MMC",
      "-IE2'" + DBQUERY ("SELECT * FROM PARTS WHERE CustID =" +
      CustomerNo:OrderRecord:OrderFile +
      " ORDER BY PARTNO",
      "PartsDB.MDQ",
      "PartsDatabase") +"'")
```

Example 4 - using WORD to parse multi-column output from DBQUERY

In certain situations, you may want to use one of the database functions, rather than a database source, due to the size of a cross-reference table. However, you need the function to return the data from several different columns.

For example, assume you need to create a map that processes inventory requests, one order at a time, using a messaging system. Within the transformation of your data, you need to reference the item master table that contains hundreds of thousands of rows. However, for each item within the inventory request, you need to get the internal item number, vendor ID, and description column value. You can select from several available options, such as:

- Use a database source.

  Define a query for only those columns needed. Use this for a database source that you can then use within a LOOKUP, EXTRACT, or SEARCHUP/SEARCHDOWN function. However, due to the size of the item master table, this might mean validating hundreds of thousands of rows to find the item information for only a few items.

- Use multiple DBLOOKUP functions.

  Assuming that the internal item number, vendor ID, and description column values are going to be used within different outputs, you could use three separate DBLOOKUP functions to get the appropriate column value for each item. However, this means executing three SQL statements to access the same row within the item master table.

- Use the DBLOOKUP and WORD functions.

  Assuming that a functional map will be used to build an object containing the three desired columns, a DBLOOKUP could be used as an argument to the functional map that retrieves the desired column values. see the following example:

  ```
  =F_MakeOne ( Item Set:SomeInput ,
     DBLOOKUP ( "SELECT INT_ITEM_NO, VENDOR, ITEM_DESC " +
                "FROM ITEM_MASTER WHERE ITEM_NO = '" +
                 CatalogID:.:SomeInput + "'" , "PRODXL.MDQ"
     "WDDM" ) )
  ```

The functional map **F_MakeOne** has two inputs: an **Item Set** and a text item (that is the result of the DBLOOKUP function). The text item will contain the three column values separated by the pipe character (|). An example follows:

```
ARQJ06X6|DFQCO|6' Jump Rope
```

Then, each rule requiring one of these pieces of data will use the WORD function to access the appropriate column's data. For example, if the input card for the results of the DBLOOKUP function was called **ItemData**, the rule using the vendor ID column would be:

```
=WORD ( ItemData , "|" , 2 )
```

The values of the other columns can be retrieved in a similar manner.

Example 5 - using DBQUERY with adapter commands to obtain a single column value

This example demonstrates how you can use the DBQUERY function to enter adapter commands as an argument following database_name using the Syntax1 format:

```
DBQUERY("SELECT PART_NAME from PARTS
   where PART_NUMBER =1",
   "mytest.mdq","PartsDB : -T -CS")
```

In the DBQUERY example above, the PART_NAME column value in the row that contains the PART_NUMBER column value of 1 will be retrieved from the PARTS table. The PARTS table is in the PartsDB Oracle database specified in the **mytest.mdq** file, which was produced by the Database Interface Designer.

This example uses optional database-specific adapter commands. The Trace (-T) adapter command will produce a database trace file. The Commit by Statement (-CS) adapter command will commit or rollback the transaction after the rule executes.

Uses

When you might want to use DBLOOKUP to execute an SQL statement:

- To retrieve a single column value from a database based upon another value in your data without the carriage return/line feed row terminator

When you might want to use DBQUERY to execute an SQL statement:

- To look up multiple column values for multiple rows in a database using a parameterized query based upon another value in your data
- When your SQL statement is a SELECT statement, the DBQUERY function can be used along with the RUN function to issue dynamic SELECT statements whose results can be used as input to a map.
- When using other SQL statements such as INSERT, UPDATE, DELETE, and so on.

For information about functions such as EXTRACT, LOOKUP, SEARCHUP, SEARCHDOWN, GET, and PUT, see the Functions and Expressions documentation.

Using bind values in database functions

For information about the availability and usage of the bind facility for your specific database and platform, see the database-specific adapter reference guides.

When the Database Management System (DBMS) receives an SQL request, the request is cached because many applications repeatedly issue the same SQL statement. If the SQL statement differs from one that the DBMS has recently processed, the statement is reevaluated. (The DBMS performs parsing, derives an execution plan, and so on.)

Similarly, if a DBLOOKUP or DBQUERY function repeatedly issues the same statement, the second and subsequent executions of the statement execute much faster than the first execution. However, if any element of the statement varies, the DBMS considers the statement to be new and does not take advantage of caching. For example, the two following statements are distinct to a DBMS:

```
SELECT * FROM MyTable WHERE CorrelationID=123
SELECT * FROM MyTable WHERE CorrelationID=124
```

Use the bind facility for DBLOOKUP and DBQUERY functions to submit such statements to the DBMS so that the statements are syntactically identical. By binding a value to a placeholder in the SQL statement, the actual syntax of the statement can be made static.

The syntax for specifying a value in the SQL statement as a bind value is:

```
:bind( value )
```

For example, to use a bind variable in the statement above, the SQL statement would be:

```
SELECT * FROM MyTable WHERE CorrelationID=:bind(123)
```

The database adapter strips out the :bind keyword and binds the value 123 to a placeholder in the statement.

The value in the parentheses is always a text item. Single quotation marks should not be specified around string literals. For example, if you had the statement:

```
SELECT Artist FROM CDList WHERE Title = 'Goodbye'
```

and you want to bind the value for the title, the syntax would be:

```
SELECT Artist FROM CDList WHERE Title=:bind(Goodbye)
```

Within the context of a DBLOOKUP or DBQUERY function, the elements of the statement to be bound are dynamic elements. For example, if the following call toDBLOOKUP is in a map:

```
DBLOOKUP         ("SELECT Name FROM MyTable WHERE ID="+ Item1:Row +
          "and CorrelationID= '" + Item2:Row + """,
          "DB.mdq",
          "MyDB")
```

The call could be modified to benefit from binding values as follows:

```
DBLOOKUP         ("SELECT Name FROM MyTable WHERE ID=:bind("+ Item1:Row +")
                       and CorrelationID=:bind(" + Item2:Row +")",
                     "DB.mdq",
                     "MyDB")
```

There is no performance benefit unless *all* values that change from one invocation of the statement to the next are bound. For example, if the ID value is bound and *not* the **CorrelationID** value, the statement will vary because the **CorrelationID** value varies.

Chapter 8. Using stored procedures

This chapter describes the various mechanisms for using stored procedures to access all types of parameters and return values from stored functions. You can call stored procedures when using DBQUERY and DBLOOKUP functions and when defining a query in the Database Interface Designer for a map data source. When defining a database as the target for an output card, you can also output to a stored procedure.

For information about the availability and usage of calling stored procedures, including the correct native syntax for these calls, refer to each database-specific adapter reference guide.

Calling stored procedures

When used as a source of data in one of the database functions (DBQUERY or DBLOOKUP), in a GET function, or in a query defined in the Database Interface Designer, there are two ways to call stored procedures:

- Use the database-independent syntax for calls.

  This provides a means of accessing the output parameters and return values from stored functions.
- Use the native syntax for the database.

  Using this method, you cannot access output parameters or return values from stored functions.

In addition to the above methods, when using Oracle object types as parameters in stored procedures, there is a special syntax that must be used. For more information about this, see ″Stored Procedures with Object Type Parameters″ .

Database-independent syntax for calls

The database-independent syntax for calling stored procedures is:

```
CALL [?=] procedure_name [(argument_list)]
```

The arguments are described below.

Option	Description	
?=	Indicates that the return value from a stored function should be returned	
procedure_name	The name of the stored procedure or function	
argument_list	A comma-separated list of arguments in which each argument is one of the following:	
	value	This is the value to be passed to an input parameter. If the value contains spaces, it must be surrounded by single quotation marks (for example, 'city of'). The quotation mark characters are not passed as part of the parameter.

Option	Description	
	?	Return the value from an output parameter.
	X	Do not return the value from an output parameter.
	?/*value*	This is the value to be passed to an In / Out parameter and the value to be returned from the parameter.

Examples using stored procedures

In the following example, there is a stored procedure named **DoSalaryIncrease** with the following parameters:

```
Employee ID -                   parm1 - In
Merit Increase (%) -            parm2 - Out
Salary -                        parm3 - In / Out
Effective Date -                parm4 - Out
```

Using the DBLOOKUP function, an example of the syntax used to call **DoSalaryIncrease** would be:

```
DBLOOKUP
("call DoSalaryIncrease("
 + EmpID:.:PayrollFile
 + ",?,?/"
 + Salary:.:PayrollFile + ",?)", "AdminDb.mdq", "HR_DB")
```

If **EmpID** has a value of **SM01930** and **Salary** has a value of **42,750**, the call syntax would expand to:

```
call DoSalaryIncrease (SM01930, ?, ?/42750, ?)
```

The output from this call is a single text item in which each field is delimited by the pipe character (|). In this example, there will be three fields in the output-one for each question mark character (?) placeholder. If you do not want to use all of the output parameters, you can use any other character in place of the question mark character. For example, if you did not use the **EffectiveDate** (parm4), the call could be changed to:

```
DBLOOKUP
("call DoSalaryIncrease ("+ EmpID:.:PayrollFile
 + ",?,?/" + Salary:.:PayrollFile + ",X)",
 "AdminDb.mdq", "HR_DB")
```

Once again, if **EmpID** has a value of **SM01930** and **Salary** has a value of **42,750**, the call syntax would expand to:

```
call DoSalaryIncrease (SM01930, ?, ?/42750, X)
```

In this example, the **EffectiveDate** (parm4) would not be returned.

Returning the Value from a stored function

If you want the value from a stored function, prefix the procedure or function name with ?=. For example:

```
DBLOOKUP("call ?= MyFunction('cat') , "My.mdq" , "MyDB")
```

Using a stored procedure as an input

Stored procedures can be used in input cards by specifying the call in the **Query** field in the New Query dialog of the Database Interface Designer.

After the stored procedure call is entered in the New Query dialog or the Edit/View Query dialog (if you are editing an existing query), you can generate a type tree for the stored procedure.

Generated type trees for stored procedures adhere to the same format as type trees for queries and will contain a **Row** group, the components of which correspond to the ? placeholders in the CALL statement. Return values from stored functions will typically have the field name **RETURN_VALUE** unless the database returns a specific name.

The call syntax for a stored procedure can be used effectively in input cards in combination with substitution variables to provide values to input parameters. For example, to call **GetPaymentInfo** in an input card, because **GetPaymentInfo** takes a single input parameter, first define the text in the **Query** field.

Next, provide the value on the command line for **parm1** at execution time as follows:

```
dtx MyMap.mmc -ID1 '-VAR parm1=2000-03-11'
```

Using a query to execute a stored procedure

If your database permits the execution of a SELECT statement in a stored procedure or function and can return values through subsequent row fetches, you can specify a procedure call using the native database syntax in the **Query** field of the **New Query** or Edit/View Query dialog using the Database Interface Designer.

For example, to call a stored procedure by means of ODBC using the native syntax, the query text you enter in the **New Query** or Edit/View Query dialog might be:

```
{call MyProc(-1)}
```

The query text for a call to a stored procedure must be in the native database syntax. For more information about the correct syntax of the procedure call using your particular database adapter, see the database-specific adapter reference guide.

To define a query using a stored procedure:

When you use a stored procedure to define a query, follow the same steps as when you use a SELECT or other statement:

1. Define the query in the **New Query** or Edit/View Query dialog of the Database Interface Designer.
2. Generate a type tree from the query.
3. In the Map Designer, specify the query when defining the data source for the executable map input card for the stored procedure.

Using a stored procedure as an output

Using the database adapters, you can call stored procedures in output cards.

Type trees generated for stored procedures contain an item type for each input argument. Instead of a **Row** type, these type trees have a **ProcedureCall** type that represents the set of arguments passed to the stored procedure for each execution of that procedure. The **DBProcedure** type consists of a series of **ProcedureCalls**. The stored procedure is called once for each **ProcedureCall** in the **DBProcedure** output. This information is highlighted in the following example.

There are two ways to call stored procedures in output cards:

- In the output card dialog in the Map Designer, specify `-PROC` *procedure_name* in the **PUT> Target → Command** setting. For more information, refer to ″Using a database as a target″.

  or

- When overriding an output card using database adapter commands on the command line, use the `-PROC` adapter command and specify the stored procedure name. For more information, see the Resource Adapters documentation.

The values for each parameter are passed to the stored procedure with the stored procedure being called for each row of data.

The types of the parameters may be IN, OUT, or IN/OUT. However, there is no mechanism to return values from output parameters. Any values passed to an OUT parameter will be ignored.

Stored procedures with object type parameters

The Oracle adapter provides full support for Oracle object types to be used as parameters to stored procedures or functions. Mapping to a stored procedure call in an output card is no different than mapping to a table; object types as parameters are fully supported.

Similarly, invoking stored procedures with object type parameters using the 'call' syntax is possible. For output parameters, there is no special syntax required. Using a question mark character (?) will result in the entire object being returned from the stored procedure call. However, for input parameters, a special syntax is required to specify the objects. The syntax rules are as follows:

- The object must be contained within square brackets. An example of this is: ″[.....]″
- Each element of the type is separated from other elements of the type by the pipe (`|`) delimiter character.
- Spaces are not allowed unless they are contained within the data itself.

see the following example. The type 'outer' is defined by the following 'create type' statements:

```
create type inner as object (
a_char varchar(10),
b_int  number(10,0));
create type outer as object (
x_inner inner,
y_date  date);
```

Assume that there is a table named 'object_holder' and a stored function 'insert_object' defined to insert an object of type 'outer' and to return the number of objects in the table. The SQL required to create this follows:

```
create table object_holder (
myobj outer);
create or replace function insert_object (obj in outer) return
number is
row_count number;
begin
insert into object_holder values (obj);
select count(*) into row_count from object_holder;
return row_count;
end;
```

To provide an object with the following attributes:

```
x_inner.a_char = 'hello'
x_inner.b_int = 23
y_date = 2000-10-12 04:59:23
```

use the following query:

```
call ?= insert_object([[hello|23]2000-10-12 04:59:23])
```

Note that the inner object is also delimited by square brackets.

Chapter 9. Database triggers

Data sources using database adapters can serve as input event triggers that are defined in the Database Interface Designer, enabled in the Integration Flow Designer, and executed by an Launcher.

For information about the availability and usage of triggering for your specific database and platform, see the database-specific adapter reference guide.

Database triggers overview

This introductory section provides information about the following topics:

- ″Database Support″

  This topic discusses those databases supporting triggering and the types of triggering supported.
- ″Installation Requirements″

  This topic discusses the database-specific scripts required to be run to enable database triggering.
- ″Tables Created for Triggering″

  This topic discusses the four tables created as a result of running the installation scripts for triggering.
- ″Maintaining Triggering Tables″

  This topic discusses table maintenance required in unusual situations such as unexpected shutdown of Launchers and working with truncated tables.

Database support

The event(s) that occur causing a trigger to run a map may be the result of a map or some other application.

The following databases support triggering:

- Oracle client and server (for Windows and UNIX)
- Microsoft SQL Server (for Windows only)

  For information about the versions of these databases that are supported in this release, see the release notes **readme.txt** file.

The Database Interface Designer allows two different triggering actions during event-based map execution:

- Table-based triggering
- Row-based triggering
  - Column-based triggering (an enhancement to row-based)

Installation requirements

Before you can take advantage of the database triggering functionality, either one of three database-specific installation scripts must be run by a system administrator:

Note: This is a one-time only operation except for a possible unexpected Launcher shutdown. For more information about that scenario, refer to ″Handling Unexpected Shutdowns″ .

- **m4ora.sql** (for Oracle)
- **m4ora_col.sql** (for Oracle)
- **m4sqlsvr.sql** (for Microsoft SQL Server)

Executing the respective script installs the database objects required for database triggering:

- tables that track ″watch″ events

  Four tables are added to your database. These are required for the triggering functionality. For more information about these tables, refer to ″Tables created for triggering″ .
- stored procedures that interface to the tables
- (for Microsoft SQL Server only) extended stored procedures that contain signaling logic
- (for Oracle only) public synonyms for tables and stored procedures
- (for Oracle only) sequences that generate unique trigger IDs

  Certain privileges must be granted to the user running the database card triggers. The specific details for this and additional information are presented at the beginning of each script.

Column-based triggering

To use column-based triggering (for Oracle only), run either of the following scripts:

Script **Use**

m4ora_col.sql
: if you are upgrading from a release prior to WebSphere Transformation Extender 8.0, and had been using Oracle database triggering

m4ora.sql
: to re-install the database objects required for column-based triggering

For more information, refer to ″Column-Based Triggering″.

Tables created for triggering

See the following table for information about the four tables that are created by executing the SQL script.

Table **Description**

Trigger_Server
: Used at startup, this table tracks the Launchers that are accessing this database. The Launchers must have unique machine name and TCP/IP address combinations. (In other words, triggering is only supported for one Launcher per machine.)

Trigger_Catalog
: Used at startup, this table tracks all triggers for all tables that have been defined on this database. This functionality allows table ″watches″ to be reused across multi-user/Launcher environments.

Trigger_Registry
Used during processing, this table registers every ″watched″ table for each Launcher. The ″watches″ on each card are placed here. In this registry is every map on every machine that is being ″watched″.

Trigger_Events
Used during processing, this table records all events that have occurred as a result of the table ″watches″ defined in the Trigger_Registry table.

Note: For Oracle only. If an Oracle table is truncated or dropped while having unprocessed row-based table entries, there may be map execution problems. To avoid these problems, you may have to perform some maintenance on this table. For information about how to do this, refer to ″Handling truncated tables″ .

Maintaining triggering tables

The current implementation of the triggering functionality should be self-maintaining. However, there are unusual situations in which some user maintenance might need to be performed as described in the following sections:

- ″Handling unexpected shutdowns″
- ″Handling truncated tables″

Handling unexpected shutdowns

Note: Make sure that there are no other Launcher systems running on the target DBMS before running the maintenance operations listed below.

If an Launcher system (containing database triggers) has encountered an unexpected shutdown and cannot be re-started and subsequently shut down in a normal sequence, you must re-run the triggering installation script (specific to your database) to clean up any triggering resources that have been left behind on the target DBMS. This is because when an Launcher is prematurely shut down, any defined database triggers will remain in operation on the backend DBMS (for example, Oracle) until the same system is re-started and then properly shut down.

The database triggers will continue to monitor for events while the Launcher is shut down, thus enabling the system to be fault-tolerant with respect to any database changes that occur during an unexpected downtime. This fault-tolerant behavior requires that a normal system shutdown eventually take place. Otherwise, the database triggers will continually remain in operation, taking up processing and space resources on the target DBMS.

If the triggering installation script cannot be run, a temporary solution would be to run the following SQL statements on the target DBMS:

1. DELETE FROM Trigger_Events;
2. DELETE FROM Trigger_Registry;
3. COMMIT;

Handling truncated tables

Note: For Oracle only.

If an Oracle table is truncated or dropped while having unprocessed row-based table entries, there may be map execution problems. To avoid these problems, you may have to perform some maintenance on the Trigger_Events table as shown in the following procedure.

Make sure you only delete those rows in the Trigger_Events table associated with the truncated table. Also, you can perform this procedure after dropping and recreating a table.

To delete table-specific truncated rows:

1. Enter the following SQL statement:

```
DELETE from Trigger_Events WHERE ID IN (
   SELECT ID FROM Trigger_Registry WHERE TriggerName IN (
      SELECT TriggerName FROM Trigger_Catalog
         WHERE      UPPER(SourceSchema) = UPPER('schema_name') AND
               UPPER(SourceTable) = UPPER('table_name')))
```

   where *schema_name* is the name of your schema and *table_name* is the name of the truncated table.
2. Commit the transaction (for example, `COMMIT;`).

Table-based triggering

A typical usage for table-based triggering would be to set up a trigger that will execute a map to run after another map has completed processing and has inserted rows into a table.

For information about the **Source → Transaction → Scope** setting, see ″Database sources and targets.″ For information about using the Commit by Statement adapter command (`-CSTMT`), see the Resource Adapters documentation.

To see how to construct the WHEN expression for table-based triggering, see the example in ″Table column format″.

Row-based triggering

The row-based triggering design enables a fault-tolerant, multi-server-aware triggering mechanism that can enable maps to selectively process those rows associated with a triggered watch event. In the preceding sentence, *fault-tolerant* means that data is not lost if a process fails, a connection is dropped, an Launcher stops, or some other unforeseen interruption occurs. *Multi-server-aware* means that Launchers on different machines can connect to and ″watch″ the same table.

The main difference between row-based and table-based triggering is that with row-based triggering, the query executes on only those rows that have been updated or inserted, not on the entire table. Additionally, batch-like processing is possible if the When clause is used to control the time and/or date at/upon which processing should occur.

When using column-based triggering, batch processing does not occur.

To see how to construct the WHEN expression for row-based triggering, refer to the example in ″SELECT 1 FROM Format″.

Column-based triggering

An enhancement to the row-based triggering functionality is column-based triggering. It enables you to trigger an event to occur on a row when a particular column condition is met. This functionality is available in both the **m4ora.sql** and

the **m4ora_col.sql** scripts. To use this functionality to trigger an Oracle database watch event for a map when a column condition is met, supply the column condition extension of the SQL WHEN expression in the Trigger Specification dialog of the Database Interface Designer. The portion of the WHEN expression that appears after **column condition** is the SQL clause that is passed to the Oracle database for processing.

Refer to Oracle documentation for the WHEN clause syntax when using it in a trigger definition.

For information on installing the **m4ora.sql** and the **m4ora_col.sql** scripts, refer to ″Installation Requirements″ .

To see how to construct the WHEN expression for column-based triggering, see the example in the Format of the When Expression section.

Issues for both row- and table-based triggering

The following list represents what is and is not supported in row- or table-based triggering:

- The appropriate database must be used (supported versions of either Oracle client and server [for Windows and UNIX] or Microsoft SQL Server [for Windows only]).
- The respective SQL file (refer back to ″Installation requirements″) must be installed and executed by your system administrator.
- SELECT statements containing Union or Intersect clauses are not supported.
- Case sensitivity for either column or table names is not supported.
- Only one Launcher per machine is supported.

Issues for row-based triggering only

The following list represents what is and is not supported in row-based triggering only:

- For Microsoft SQL Server, each table defined in the query must contain either an identity column or one numeric primary key.
- Queries that reference views or public synonyms are not supported.

Defining a trigger using a database source

Specify that you want a database source to be a trigger for a map when using a database adapter that supports triggering and an Launcher.

To define a trigger:

1. Use the Database Interface Designer to create an MDQ file containing the desired query.
2. From the Trigger Specification dialog in the Database Interface Designer, define the conditions under which changes in database tables will trigger a map. The trigger you define is associated with a query and is saved in the MDQ file. For more information about this, refer to ″Defining a trigger for a query″ .
3. Use the Map Designer to create an executable map with an input card that uses the MDQ file containing the trigger specification. For more information, see the Map Designer documentation.

Make sure that the input card corresponding to the database trigger is set to integral mode, not burst.

4. Use the Integration Flow Designer to create a system with the map component containing the trigger specification. Enable the **Input(s)** → **GET** → **Source** setting in the Launcher Settings dialog. For information about how to do this, refer to ″Using the Integration Flow Designer to enable triggers″ .

   The Integration Flow Designer does not perform any validation of a database source used as a trigger. Ensure that the trigger has been defined using the Database Interface Designer prior to specifying it as an input event trigger.
5. Generate the Launcher system file (**.msl**) on the Launcher as appropriate. For more information about how to do this, see the Integration Flow Designer documentation.

Using a database as a map trigger

Database sources can be used as input events for the Launcher. Insertions, deletions (for table-based triggering only), or updates to database tables can be the trigger mechanism that will run a map.

A trigger specification is defined in the Database Interface Designer for an individual query and is stored with the query in the MDQ file. When a database source is specified as an input event in the Integration Flow Designer, the trigger specification is evaluated by the Launcher and is run accordingly.

Defining a trigger for a query

After you have defined a query in the Database Interface Designer, you can define a trigger specification that can be used to launch a map using that same query as a data source.

To define a trigger specification:

1. In the Database Interface Designer, select the query for which you want to define a trigger specification.
2. From the **Query** menu, select **Define Trigger**.

   The Trigger Specification dialog appears. When initially displayed, the lists on the left show all of the tables referenced by the SELECT statement of the selected query.
3. Determine whether you can define row-based triggering for this query as determined by the availability of the **Row-based triggering** check box.

   If this is a new query and the table supports it, this check box will be enabled as the default value. (For old queries, this check box will be cleared as the default value, which means that you could use the functionality by enabling it. In another scenario, the entire check box could be disabled because this functionality is not available for the query and table being used.)
4. To specify the events, select a table name from the **Insert into** or **Delete from** lists.

   or

   Select a table name from the **Update of** list.

   The name of the table you select will be either the destination into which data will be inserted, from which data will be deleted, or in which data will be updated when the specified event occurs.

   Note: Delete from cannot be used with row-based triggering.

5. To move the table name, in the text box area corresponding to the selected table name (either **Insert into**, **Delete from**, or **Update of**), click the arrow button.

   The table name moves into the corresponding list on the right.
6. Select as many names as necessary to define the event(s) that must occur so that conditions can be met for the trigger specification.
7. If more than one table is added to a list, you can define conditions that must be met. Select either the **AND** or **OR** radio button located directly above each list. For more information, refer to ″Specifying a combination of different event classes″ .

   The **AND** option dictates that the condition is not met until the event occurs for *all* tables in the list when data is either inserted, deleted, or updated. The **OR** option dictates that the condition is met when the specified event occurs in only *one* table in the list.
8. To define an additional condition that will be evaluated only after the other specified event(s) has/have occurred, enter an expression in the **When** field. For more information, refer to ″Specifying when″ .
9. After you have finished, click **OK** to save your trigger specification.

   The query with the trigger specification is represented in the Navigator with an icon displayed next to the name.

   Note: To modify or delete an event, use the same procedure as indicated above. Select the table name in the list on the right and click the corresponding left arrow button. The **When** expression can also be edited or deleted as required.

Defining events

Use the Trigger Specification dialog to define a trigger specification for an individual query that will be stored with the query in the MDQ file. As you add and remove the table names, the lists on the right display the tables for which an **Insert into**, **Delete from**, or **Update of** event comprises the condition(s) that must be met for the trigger specification. Additionally, you can specify whether row-based triggering will be used. The different classes of events that you can specify are:

- **Insert into**

  The insertion of rows into the specified table serves as an input event trigger to the map that uses this query as a data source.
- **Delete from**

  The deletion of rows from the specified table serves as an input event trigger to the map that uses this query as a data source.

  Note: This functionality cannot be used with row-based triggering. As soon as you move a table to the right, the **Row-based triggering** check box is disabled.
- **Update of**

  The update of rows in the specified table will serve as an input event trigger to the map that uses this query as a data source.

Specifying a combination of different event classes

If you define your trigger specification using a combination of event classes (**Insert into, Delete from** or **Update of**), there is an implicit OR between the different event classes.

For example, if you defined events in all three event classes, the implied condition to be met for the trigger specification would be:

```
Insert into TableA
OR
Delete from TableB
OR
Update of TableC
```

When all events have occurred within only one of the specified event classes, the condition(s) has/have been met for the trigger specification and the map is run.

Specifying AND or OR

Within the **Insert into** and **Delete from** event classes, you can define multiple insert and delete events by specifying multiple tables. Similarly, you can define multiple **Update of** events by specifying multiple table names.

Using the **Delete from** event class disables row-based triggering. However, you can still define multiple **Insert Into** and **Update of** events using row-based triggering.

When you specify multiple table names within an event class, you must also specify how you want the condition within the event class to be met-either when all of the events occur for all of the specified tables or when one event occurs on any one entry in the list.

The options that dictate the condition(s) to be met when you set up multiple events are:

Option Description

AND This option dictates that all of the specified events must occur (for example, **TableA** AND **TableB**) for the condition to be met.

OR This option dictates that conditions are met when at least one event occurs (for example, **TableA** OR **TableB**).

Specifying when

After specifying at least one **Insert into**, **Delete from,** or **Update of** event, you can create an expression that must evaluate to true to satisfy the conditions of the trigger specification. Enter an expression in the **When** field that will be evaluated after the other events have occurred. If the **When** expression evaluates to true, the conditions of the trigger specification are met and the map is run. If the **When** expression is not true, the state is restored to what it was prior to the occurrence of any events. When row-based triggering is used, all of the changed rows will be batched together for subsequent processing after the event is re-triggered and the When clause has been satisfied.

When using column-based triggering, batch processing does not occur.

Format of the when expression

There are three basic formats that can be used for the **When** expression as specified in the **Trigger Specification** dialog:

- Clauses referencing table columns (“Table column format”)
- Clauses using the SELECT 1 FROM format to establish conditions of execution (“SELECT 1 FROM format”)
- Clauses using the column condition (“Column condition format”)

Table column format

The **When** expression in the **Trigger Specification** dialog can contain any SQL expressions that are valid for the database. If database columns are referenced in the expression, the column name must be qualified with the `tablename` and the `tablename.column_name` enclosed in square brackets ([ ]).

For example, if a map should be triggered only when there is a row in the **MyTable** table having a column entitled **Status** with a value of **Ready**, specify the **Insert into**, **Delete from**, and **Update of** events and enter the following expression in the **When** field:

```
[MyTable.Status] = 'Ready'
```

SELECT 1 FROM format

The **When** expression can support any valid SQL statement that begins with a SELECT 1 FROM clause. After a database event is detected on the DBMS, the entire statement (as it appears in the Trigger Specification dialog) is executed.

For example, if a database input card should be triggered for execution only during a certain time of day, the following statement could be entered:

Note: This example only applies to versions of the Oracle DBMS supported in this release.

```
select 1 from dual where
TO_CHAR(SYSDATE, 'HH24') > '00' AND
TO_CHAR(SYSDATE, 'HH24') < '06'
```

This **When** clause would restrict processing of any database ″watch″ events to the timeframe of between midnight and 6AM.

Column condition format

The **When** expression can support column-based triggering.

An example of the syntax of the column condition that you would specify in the Trigger Specification dialog of the Database Interface Designer is presented below:

```
column condition new.part_number = `S'
```

Because column-based triggering is based on row-based triggering, the **Row-based triggering** check box must be enabled. You would enter the column condition new.part_number = `S’clause into the **When** pane. The portion, new.part_number = `S’, specified after column condition, is the SQL clause that the Oracle database will process.

Specifying triggers on the command line

If a trigger specification for a query has not been defined in the Database Interface Designer or if you want to override conditions that are already defined in a trigger specification associated with a query, use the command line.

For more information about using the command line and specifying the Trigger adapter command (`-TR`), see the Resource Adapters documentation.

Using the Integration Flow Designer to enable triggers

After you have defined the trigger using the Database Interface Designer and have created an executable map using the Map Designer, use the Integration Flow Designer to define a system containing the associated map component and to enable the trigger for execution by the Launcher. For more information, see the Integration Flow Designer documentation.

To set the database source as a trigger:

1. In the Integration Flow Designer, display the Launcher Settings dialog for the map you want to trigger.
2. Set the **SourceEvent** setting value to **ON**.
3. Click **OK**.

   A trigger icon appears on the map component in the system diagram. When the map component appears in expanded state, a trigger icon also appears on the upper-left corner of the input card.

   The trigger icon provides a visual cue showing the inputs that serve as triggers.

Chapter 10. Debugging and viewing results

This chapter explains various troubleshooting tools available when you encounter problems using database objects as data sources or targets for a map or when using the Database Interface Designer to define databases and queries. Also presented are various methods for viewing data extracted from a database or loaded into a database.

Troubleshooting tools

If you receive an error while attempting to generate a type tree in the Database Interface Designer or if you run a map that uses database sources and/or targets and receive a runtime error or do not get the expected output, use any or all of the following troubleshooting tools:

- Map audit log (***map_name*.log**)

  For information about the map audit log and related settings, see the Map Designer documentation.
- Database Interface Designer trace (***mdq_file_name*.dbl**)
- Database execution trace file (***map_name*.dbl**)
- Database source and target data
- Database audit file (**audit.dbl**)
- DBMS trace utilities and SQL command tools

Database trace files

Use the information contained in database trace files (**.dbl**) as one of the primary tools to assist in troubleshooting. These files contain detailed information generated during Database Interface Designer activity and also during map execution. For example:

- The trace file recorded for a Database Interface Designer trace contains information about the activities taking place when generating type trees such as database connections, SQL statements executed, and so on.
- The database trace file produced at map execution time records detailed information about the database adapter activity such as records retrieved, data source and target activity, and so on.

Format of database trace files

Information displayed in the database trace file varies depending upon the database generating the trace file. Database adapter trace files contain what appears to be a range of numbers at the beginning of each line in the file. These numbers represent:

- the process ID

  This is the number representing the process that is running on the machine. If you were watching the processes running on your machine (viewing the **Processes** tab in the Task Manager window , the value displayed in the **PID** (process ID) column would match the value displayed in the database trace file produced as a result of this process.
- the thread ID

This ID distinguishes each map that is running concurrently. Each running map has its own thread.

Producing the database trace in the Database Interface Designer

Click the **Trace** tool or select **Trace** from the **File** menu to enable a database trace file (**.dbl**). If trace is enabled, a database trace file is automatically created when you generate a type tree. This file is placed in the same directory as the currently open MDQ file. The newly created database trace file is named using the full name of the MDQ file plus a **.dbl** extension. For example, if your MDQ file is named **Orders.mdq**, the trace file is named **Orders.dbl** and is located in the same directory. If your MDQ file is not yet named, the trace file is named **Database_QueryFile*1*.dbl** and resides in the same directory, typically the default installation directory.

Most problems encountered in the Database Interface Designer are relayed to the user in message dialogs. For example, if an incorrect user-ID or password is specified for a database, a dialog appears when attempting to generate type trees for tables in that database. The following example dialog reports an Oracle error ORA-01017 with the following message as returned by the database driver to the database adapter:

```
invalid username/password; logon denied
```

If trace is enabled when the message in this dialog appears, a corresponding message is also written to the database trace file (***.mdq_file_name*.dbl)**.

The following is a sample of a database trace file when generating a type tree that describes two tables in an Oracle database.

The contents of any trace file are database platform-specific.

```
<1776-940>:  Datalink: bocadb2\\Northwind
<1776-940>:  UserId  : test
<1776-940>:  Password:****
<1776-940>:  No existing connection was found.
<1776-940>:  Local transaction usage: Transaction ID 0x019287F4
<1776-940>:  Transaction started - ISOLATIONLEVEL_READCOMMITTED
<1776-940>:  Connection to SQL Server bocadb2 has been established.
<1776-940>:  Retrieving 1 rows per fetch.
<1776-940>:  The columns are of the following types:
<1776-940>:  Column 1 (CustomerID) type is nchar(5) [DBTYPE_WSTR].
<1776-940>:  Column 2 (CompanyName) type is nvarchar(40) [DBTYPE_WSTR].
<1776-940>:  Column 3 (ContactName) type is nvarchar(30) [DBTYPE_WSTR].
<1776-940>:  Column 4 (ContactTitle) type is nvarchar(30) [DBTYPE_WSTR].
<1776-940>:  Column 5 (Address) type is nvarchar(60) [DBTYPE_WSTR].
<1776-940>:  Column 6 (City) type is nvarchar(15) [DBTYPE_WSTR].
<1776-940>:  Column 7 (Region) type is nvarchar(15) [DBTYPE_WSTR].
<1776-940>:  Column 8 (PostalCode) type is nvarchar(10) [DBTYPE_WSTR].
<1776-940>:  Column 9 (Country) type is nvarchar(15) [DBTYPE_WSTR].
<1776-940>:  Column 10 (Phone) type is nvarchar(24) [DBTYPE_WSTR].
<1776-940>:  Column 11 (Fax) type is nvarchar(24) [DBTYPE_WSTR].
```

The following example was received when attempting to run a map to retrieve data from a table in an SQL Server database.

```
<1324-3020>: Validating the adapter command...
<1324-3020>: Database type is MS SQL Server 7
<1324-472>:  Connecting...
<1324-472>:  Datalink: MY_2000_SERVER\\test
<1324-472>:  UserId  : test
```

```
<1324-472>:  Password:****
<1324-472>:  OLE DB Error code: 0x80004005
<1324-472>:  [DBNMPNTW]Specified SQL server not found.
<1324-472>:  Returned status: (-3) No error text found
```

In this example, the following lines:

```
<1324-472>: OLE DB Error code: 0x80004005
<1324-472>: [DBNMPNTW]Specified SQL server not found.
```

indicate the OLE DB provider-specific error code and the corresponding error description. The last line (beginning with `Returned status:`) indicates the error code returned by the adapter that caused the map to fail.

This information can be used, along with the SQL Server documentation, to resolve the problem. In this particular example, the error was caused by typing the incorrect server name in the SQL Server **Server** settings in the Database Definition dialog, rather than by selecting the name from the list of servers. In this example, the server name was incorrectly entered as `MY_2000_SERVER`, instead of its actual name (`MY_2000_SRVR`).

Producing the database trace during map execution

There are several different ways to enable the database trace for reporting database-related information during map execution as discussed in the following sections:

- Using the Trace adapter command
- Source usage:
  - for a valid source
  - for a source with errors
- Target usage:
  - for a valid target
  - for a target with a missing required value
  - for a target using the Bad Data adapter command
  - for a target with `-UPDATE` off
  - for a target using the DBLOOKUP/DBQUERY functions

Using the Trace adapter command

To produce database trace information for specific database data sources or targets, use the Trace adapter command (`-TRACE`). For information about the syntax of this command, see the Resource Adapters documentation.

To produce the database trace file when using the `DBQUERY` or `DBLOOKUP` functions, use the Syntax2 format to call the function. For example, if output card 3 contains a `DBLOOKUP` function, call the function as

```
DBLOOKUP ( "SELECT ...", "-DBTYPE DB2 -SOURCE LAMBIN
                         -USER ARL97IN -TRACE").
```

Database trace for a valid source

To produce the following database trace file example, we want to produce a database trace for input card 2 (which is a database). To do this, you must include the Trace adapter command (`-TRACE`) in the **GET** → **Source** → **Command** setting. Upon execution, database trace information is generated for input card 2; the example follows.

Note: Line numbers are for reference purposes only.

```
01  Database type is Oracle
02  Status returned to engine: (0) Success
03  No existing connection was found.
04  Connection to Oracle has been established.
05  Interface library version 6.0(140)
06  Data being retrieved for input card 2.
07  Database adapter version 6.0(140)
08  Starting a database unload...
09  Host string:
10  Userid     : demo
11  Password   : ****
12  Query      : SELECT I.*, P.LIST_PRICE, P.MIN_PRICE, P.START_DATE, P.END_DATE
13         FROM ITEM I, PRICE P, PRODUCT PR
14         WHERE I.PRODUCT_ID = P.PRODUCT_ID and PR.PRODUCT_ID = 15  I.PRODUCT_ID
15         ORDER BY PR.PRODUCT_ID
16  Query size : 189
17  Output is to a buffer.
18  Statement execution succeeded.
19  The columns are of the following types:
20     Column 1 (ORDER_ID) type is NUMBER(4).
21     Column 2 (ITEM_ID) type is NUMBER(4).
22     Column 3 (PRODUCT_ID) type is NUMBER(6).
23     Column 4 (ACTUAL_PRICE) type is NUMBER(8,2).
24     Column 5 (QUANTITY) type is NUMBER(8).
25     Column 6 (TOTAL) type is NUMBER(8,2).
26     Column 7 (LIST_PRICE) type is NUMBER(8,2).
27     Column 8 (MIN_PRICE) type is NUMBER(8,2).
28     Column 9 (START_DATE) type is DATE.
29     Column 10 (END_DATE) type is DATE.
30  Number of buffers in fetch array = 574
31  Writing results to a buffer.
32  Retrieved 543 records (34451 bytes).
33  Status returned to engine: (0) Success
34  Cleaning up and closing the transaction...
35  The transaction was successfully committed.
36  Status returned to engine: (0) Success
37  Commit was successful.
38  Database disconnect succeeded.
```

This sample database trace file (***map_name*.dbl**) reveals important information, examples of which are described below.

- Lines 1-5 show information about the connection made, identifying the database type as Oracle as well as the version number of the interface library. In this example, there was no existing database connection and a new connection was successfully established.
- Line 6 indicates that the data is being retrieved for input card number 2.
- Line 7 identifies the version of the database adapter for Oracle. Lines 8-11 identify the information used to make the connection with the Oracle database: the host string, user ID, and password. Because this information is for an input card, there is a message indicating that a database unload is being started.
- Next, the database trace file indicates that the query successfully executed (as indicated by the following text on Line 18: `Statement execution succeeded`) and includes descriptions of the ten columns to be retrieved.
- The results of the query are written to the buffer. 543 records (rows) were retrieved using this query.

Use the Trace adapter command (`-TRACE`) for as many database source or targets as desired. Note that the information for later cards is appended to the default database trace file unless a file name is specified using the `-TRACE filename` option as described in the Resource Adapters documentation.

The exact information in the database trace file varies, depending upon the information returned by different entities including the database driver, database source versus target, and so on.

Database trace for a source with errors

The example in the preceding section provided database trace information for a database source with no problems. The following is an example of information you may see when there *are* errors while accessing the database source.

```
Database type is Oracle
Status returned to engine: (0) Success
No existing connection was found.
Connection to Oracle has been established.
Interface library version 6.0(140)
Data being retrieved for input card 1.
Database adapter version 6.0(140)
Starting a database unload...
Host string:
Userid     : demo
Password   : ****
Query      : SELECT * FROM BONUSS
Query size : 20
Output is to a buffer.
Error in: oparse
Message : ORA-00942: table or view does not exist
Retrieved 0 records (0 bytes).
Error returned to engine: (-17) Failed to parse SQL statement
Cleaning up and closing the transaction...
The transaction was successfully committed.
Status returned to engine: (0) Success
Commit was successful.
Database disconnect succeeded.
```

The key information in the database trace is in the line defining the query (beginning with`Query`) and the four lines beginning with the `Error in:` line. In this example, the attempt to execute the query failed because the table specified in the query listed does not exist. This would result in a map return code of 12 and a message of:

```
Source not available
```

Upon receiving this error, verify that the query is correct and that the table referenced in the query exists using the Database Interface Designer or the SQL command tool provided with your database.

Database trace for a valid target

The information included in the database trace for an output (data target) is similar to that for a database source. The following example shows the **PUT → Target** settings for the first output card (which is a database) in a map called **UpdateMembershipDB**.

The Update adapter command (`-UPDATE`) specifies that the rows produced by the map should update existing rows or should insert new rows, based upon the update keys configured for the table in the Database Interface Designer. The Trace adapter command (`-TRACE`) generates the database trace information for this output card.

An example of the database trace information produced for this map follows.

```
Database type is Oracle
Status returned to engine: (0) Success
No existing connection was found.
Connection to Oracle has been established.
Interface library version 6.0(140)
Loading data for output card 1.
Database adapter version 6.0(140)
Starting database load...
Host string:
Userid     : eventmgt
Password   : ********
Update mode is on.
The columns are of the following types:
   Column 1 (MEMBERID) type is VARCHAR(10).
   Column 2 (ATTENDEEFIRSTNAME) type is VARCHAR(30).
   Column 3 (ATTENDEELASTNAME) type is VARCHAR(50).
   Column 4 (TITLE) type is VARCHAR(50).
   Column 5 (COMPANYNAME) type is VARCHAR(50).
   Column 6 (ADDRESS) type is VARCHAR(255).
   Column 7 (CITY) type is VARCHAR(50).
   Column 8 (STATEORPROVINCE) type is VARCHAR(20).
   Column 9 (POSTALCODE) type is VARCHAR(20).
   Column 10 (COUNTRY) type is VARCHAR(50).
   Column 11 (PHONENUMBER) type is VARCHAR(30).
   Column 12 (FAXNUMBER) type is VARCHAR(30).
   Column 13 (EMAILADDR) type is VARCHAR(50).
   Column 14 (MEMBERSINCE) type is DATE.
The insert statement to be executed is:
INSERT INTO Membership  VALUES
     (:a00,:a01,:a02,:a03,:a04,:a05,:a06,:a07,:a08,:a09,:a10,:a11,:a12,:a13)
The update statement to be executed is:
UPDATE Membership SET ATTENDEEFIRSTNAME=:a01,ATTENDEELASTNAME=:a02,TITLE=:a03,
     COMPANYNAME=:a04,ADDRESS=:a05,CITY=:a06,STATEORPROVINCE=:a07,
     POSTALCODE=:a08,COUNTRY=:a09,PHONENUMBER=:a10,FAXNUMBER=:a11,EMAILADDR=:a12
     WHERE (MEMBERID=:a00)5 rows inserted.
2 rows updated.
Database load complete.
Status returned to engine: (0) Success
Cleaning up and closing the transaction...
The transaction was successfully committed.
Status returned to engine: (0) Success
Commit was successful.
Database disconnect succeeded.
```

This sample database trace file (***map_name*.dbl**) reveals important information that is highlighted in **bold** type above and some of which is described below.

- After the logon information in the sample, a message indicates that `Update mode is on`. This will affect how the rows produced are handled. Because update mode is on (enabled by the `-UPDATE` adapter command), rows in the table are inserted or updated based upon the update keys defined in the Database Interface Designer.
- After the connection is made, the database trace shows the column definitions for the output table.
- Next, the database trace file displays the actual SQL statements to be executed using the data produced for the output card. Because update mode is enabled, the database trace file displays both an INSERT statement and an UPDATE statement. The UPDATE statement is executed for each row in the output for which a corresponding row is found in the database table, using the update key defined (which, in this example, is the **MemberID** column). The INSERT statement is executed for all rows in the output for which a corresponding row does not exist in the table.

- After the SQL statements in the sample, the database trace file indicates the number of rows that were both inserted (5) and updated (2).
- Finally, the database trace file indicates that database load was executed successfully (`Database load complete`) and that the database changes were committed (`The transaction was successfully committed`).

Database trace for a target: missing required value

The following database trace file example was produced by using almost the same map as in the preceding example except for the usage of a different input data file. This database trace file illustrates the information you might see if a database error occurred while attempting to insert or update rows in a table.

```
Database type is Oracle
Status returned to engine: (0) Success
No existing connection was found.
Connection to Oracle has been established.
Interface library version 6.0(140)
Loading data for output card 1.
Database adapter version 6.0(140)
Starting database load...
Host string:
Userid     : eventmgt
Password   : ********
Update mode is on.
The columns are of the following types:
   Column 1 (MEMBERID) type is VARCHAR(10).
   Column 2 (FIRSTNAME) type is VARCHAR(30).
   Column 3 (LASTNAME) type is VARCHAR(50).
   Column 4 (TITLE) type is VARCHAR(50).
   Column 5 (COMPANYNAME) type is VARCHAR(50).
   Column 6 (ADDRESS) type is VARCHAR(255).
   Column 7 (CITY) type is VARCHAR(50).
   Column 8 (STATEORPROVINCE) type is VARCHAR(20).
   Column 9 (POSTALCODE) type is VARCHAR(20).
   Column 10 (COUNTRY) type is VARCHAR(50).
   Column 11 (PHONENUMBER) type is VARCHAR(30).
   Column 12 (FAXNUMBER) type is VARCHAR(30).
   Column 13 (EMAILADDR) type is VARCHAR(50).
   Column 14 (MEMBERSINCE) type is DATE.
The insert statement to be executed is:
INSERT INTO Membership  VALUES
     (:a00,:a01,:a02,:a03,:a04,:a05,:a06,:a07,:a08,:a09,:a10,:a11,:a12,:a13)
The update statement to be executed is:
UPDATE Membership SET FIRSTNAME=:a01,LASTNAME=:a02,TITLE=:a03, COMPANYNAME=:a04,
     ADDRESS=:a05,CITY=:a06,STATEORPROVINCE=:a07, POSTALCODE=:a08,COUNTRY=:a09,
     PHONENUMBER=:a10,FAXNUMBER=:a11,EMAILADDR=:a12 WHERE (MEMBERID=:a00)
Error in: oexec
Message : ORA-01400: cannot insert NULL into
("EVENTMGT"."MEMBERSHIP"."LASTNAME")
The following values were being inserted:
  Column 1 MEMBERID : D190-0002
  Column 2 FIRSTNAME : Leverling
  Column 3 LASTNAME : NULL
  Column 4 TITLE : Vice President-New Products
  Column 5 COMPANYNAME : Northwind Traders
  Column 6 ADDRESS : 722 Moss Bay Blvd.
  Column 7 CITY : Kirkland
  Column 8 STATEORPROVINCE : WA
  Column 9 POSTALCODE : 98033
  Column 10 COUNTRY : USA
  Column 11 PHONENUMBER : (206) 555-3412
  Column 12 FAXNUMBER : (206) 555-3413
  Column 13 EMAILADDR : jleverling@northwind.com
  Column 14 MEMBERSINCE : 1999-07-09 22:03:03
```

```
Failed to insert a row (rc = -9).
Failed after 1 rows inserted.
Database load complete.
Error returned to engine: (-9) Failed to execute the SQL statement
Cleaning up and closing the transaction...
Transaction rollback was successful.
Status returned to engine: (0) Success
Commit was successful.
Database disconnect succeeded.
```

In this example, the **bold** text is information that can be provided when an error occurs. The first lines of this example provide the same information as was in the database trace file with no errors: connection parameters, update mode indicator, table column descriptions, SQL statements to be used, and so on.

However, after the UPDATE statement, the database trace file provides details to assist in determining the problem. The Message line indicates that an attempt was made to insert a row with a NULL value for the `EVENT.MEMBERSHIP.LASTNAME` column that cannot contain a NULL. To further identify the row in the output data causing the error, the database trace file lists the values for all of the columns in the row causing the error.

The remaining lines in the database trace file display information about the disposition of the entire database card transaction. The transaction failed after one row was inserted. The database adapter returns an error code of `-9` to the Launcher with a corresponding error message of `Failed to execute the SQL statement`. If you produced the audit log, you would see the following line in the execution summary section:

```
<TargetReport  card="1"  adapter="DB"  bytes="1218" adapterreturn="-9">
   <Message>Failed to execute the SQL statement</Message>
   <TimeStamp>22:07:32 January 8, 2004</TimeStamp>
</TargetReport>
```

Subsequently, the database trace file communicates that the database adapter is cleaning up and closing the transaction. Because the **OnFailure** setting was set to **Rollback** for this output card, the final entry in the database trace reveals that the transaction rollback was successful.

Database trace for a target - using the bad data adapter command (-BADDATA)

Use the Bad Data adapter command (`-BADDATA`) for a target (or in a `PUT` function). If any inserts, updates, or procedure calls fail to execute, the data that could not be processed is written to a file that you specify and processing continues.

The following database trace example was produced using the same map and data as in the preceding example, now additionally using `-BADDATA`.

```
.
.
Error in: oexec
Message : ORA-01400: cannot insert NULL into
("EVENTMGT"."MEMBERSHIP"."LASTNAME")
The following values were being inserted:
Column 1 MEMBERID : D190-0002
Column 2 FIRSTNAME : Leverling
Column 3 LASTNAME : NULL
Column 4 TITLE : Vice President-New Products
Column 5 COMPANYNAME : Northwind Traders
Column 6 ADDRESS : 722 Moss Bay Blvd.
Column 7 CITY : Kirkland   Column 8 STATEORPROVINCE : WA
```

```
Column 9 POSTALCODE : 98033
Column 10 COUNTRY : USA
Column 11 PHONENUMBER : (206) 555-3412
Column 12 FAXNUMBER : (206) 555-3413
Column 13 EMAILADDR : jleverling@northwind.com
Column 14 MEMBERSINCE : 1999-07-09 22:03:03
Failed to insert a row (rc = -9).
4 rows inserted.
2 rows updated.
1 rows were rejected.
Database load complete.
Warning returned to engine: (1) One or more records could not be processed
Cleaning up and closing the transaction...
The transaction was successfully committed.
Status returned to engine: (0)
SuccessCommit was successful.
Database disconnect succeeded.
```

All of the information in this trace file example is the same as the previous example except for the bold lines of text. In this example, because -BADDATA was used, the database adapter goes on to process all of the other rows produced for the target. Upon completion, four rows were successfully inserted, two rows were successfully updated, and one row was rejected. The rejected record was saved in the file specified with the Bad Data adapter command (-BADDATA). In this example, the rejected record was saved to a file called badstuff.txt.

The remaining lines in the database trace file indicate information about the disposition of the entire database card transaction. Notice that the use of -BADDATA allows the card to successfully complete. The map successfully completes and the transaction is committed.

If you produced the audit log, you would see the following line in the execution summary section:

```
<TargetReport card="1" adapter="DB" bytes="1218" adapterreturn="1">
  <Message>One or more records could not be processed</Message>
  <TimeStamp>22:07:32 January 8, 2004</TimeStamp>
</TargetReport>
```

Database trace for a target with -UPDATE off

The following example illustrates the database trace that can result from a common error encountered when developing maps have database targets.

```
Database type is ODBC
Status returned to engine: (0) Success
No existing connection was found.
Connection to datasource OracleProd has been established.
Interface library version 6.0(140)
Loading data for output card 1.
Database adapter version 6.0(140)
Starting database load...
Datasource: OracleProd
Userid    : EVENTMGT
Password  : ********
Update mode is off.
The columns are of the following types:
   Column 1 (MEMBERID) type is VARCHAR, precision is 10.
   Column 2 (ATTENDEEFIRSTNAME) type is VARCHAR, precision is 30.
   Column 3 (ATTENDEELASTNAME) type is VARCHAR, precision is 50.
   Column 4 (TITLE) type is VARCHAR, precision is 50.
   Column 5 (COMPANYNAME) type is VARCHAR, precision is 50.
   Column 6 (ADDRESS) type is VARCHAR, precision is 255.
   Column 7 (CITY) type is VARCHAR, precision is 50.
```

```
   Column 8 (STATEORPROVINCE) type is VARCHAR, precision is 20.
   Column 9 (POSTALCODE) type is VARCHAR, precision is 20.
   Column 10 (COUNTRY) type is VARCHAR, precision is 50.
   Column 11 (PHONENUMBER) type is VARCHAR, precision is 30.
   Column 12 (FAXNUMBER) type is VARCHAR, precision is 30.
   Column 13 (EMAILADDR) type is VARCHAR, precision is 50.
   Column 14 (MEMBERSINCE) type is TIMESTAMP, precision is 19.

The insert statement to be executed is:
INSERT INTO Membership  VALUES (?,?,?,?,?,?,?,?,?,?,?,?,?,?)
Error in SQLExecute
Message: ORA-00001: unique constraint (SYSTEM.UNIQUE_MEMBER_ID) violated
SQL State: 23000
The following values were being inserted:
   Column 1 MEMBERID : I978-1964
   Column 2 ATTENDEEFIRSTNAME : Jean
   Column 3 ATTENDEELASTNAME : Fresnière
   Column 4 TITLE : Marketing Assistant
   Column 5 COMPANYNAME : Mère Paillarde
   Column 6 ADDRESS : 43 rue St. Laurent
   Column 7 CITY : Montréal
   Column 8 STATEORPROVINCE : Québec
   Column 9 POSTALCODE : H1J 1C3
   Column 10 COUNTRY : Canada
   Column 11 PHONENUMBER : (514) 555-8054
   Column 12 FAXNUMBER : (514) 555-8055
   Column 13 EMAILADDR : fresnierej@paillarde.org
   Column 14 MEMBERSINCE : {ts '1999-07-09 23:18:04'}
Failed to insert a row (rc = -9).
Failed after 2 rows inserted.
Database load complete.
Error returned to engine: (-9) Failed to execute the SQL statement
Cleaning up and closing the transaction...
Transaction rollback was successful.
Status returned to engine: (0) Success
Commit was successful.
Database disconnect succeeded.
```

If you forget to specify the usage of the update setting (using `-UPDATE` either in the **PUT → Target → Command** setting in the Map Designer or Integration Flow Designer or in the command line), you may receive a database error resulting from the attempt to insert a row with a duplicate index.

The first entry in this example highlighted in **bold** type indicates that update mode is off. Because update mode is off, the database adapter attempts to insert a row into the database table for each row produced by the map. The `Message` entry contains the message returned by the database driver to the database adapter describing the cause of the error. In this example, the row in error would violate the **UNIQUE_MEMBER_ID** constraint defined for the table. The next lines show the column values for the row in which the error occurred and the final result of the database operation.

There are several possible methods for resolving this problem. Depending upon the desired behavior, you might:

- Enable update mode when executing the map by using the `-UPDATE ON` or `-UPDATE ONLY` adapter command.
- Use the Delete adapter command (`-DELETE`) to remove all rows from the output database table before inserting the rows resulting from map execution.
- Build logic into your map to ensure that there is no existing row in the table prior to generating an output row to be inserted. This might be accomplished by

either defining a query for the table being used as an input against which a LOOKUP or SEARCHUP function is performed or by using the DBLOOKUP function to check for an existing row.

- Use -BADDATA so that all rows with a unique **MEMBERID** are inserted. Those rows that would result in duplicate rows are then saved to a specified file.

Database trace for a target: DBLOOKUP/DBQUERY functions

The following is a selection from a database trace file (***map_name*.dbl**) for an output card using the GET function to execute a DELETE statement based upon a value in the input.

```
..Starting a database unload...
Host string:
Userid     : eventmgt
Password   : ********
Query      : DELETE FROM REGISTRATION WHERE REGISTRATIONSTATUS = 'Overflow'
Query size : 71Output is to a buffer.
Statement execution succeeded.
Retrieved 0 records (0 bytes).
Size of DBLOOKUP data is 0.
Warning returned to engine: (2) No data found
Cleaning up and closing the transaction.....
```

The following SQL statement to be executed:

```
DELETE FROM REGISTRATION WHERE REGISTRATIONSTATUS = 'Overflow'
```

and the resulting successful execution statement:

```
Statement execution succeeded
```

are included in this trace file as shown in the example above. Note that no records were retrieved because the DELETE statement does not return data.

The next series of statements are produced for an output card using a DBQUERY function to obtain several columns from a database table based upon specified values in the **MEMBERID** column in the input data.

```
..Starting a database unload...
Host string:
Userid     : eventmgt
Password   : ********
Query      : SELECT FIRSTNAME, LASTNAME , EMAILADDR FROM EVENTMGT.MEMBERSHIP
             WHERE MEMBERID = 'D191-0001'
Query size : 108
Output is to a buffer.
Statement execution succeeded.
The columns are of the following types:
   Column 1 (FIRSTNAME) type is VARCHAR(30).
   Column 2 (LASTNAME) type is VARCHAR(50).
   Column 3 (EMAILADDR) type is VARCHAR(50).
Number of buffers in fetch array = 492
Writing results to a buffer.
Retrieved 1 records (40 bytes).
Size of DBQUERY data is 40.
The following data was returned from a DBQUERY:

Nancy|Davolio|nancyd@cascadecoffee.com

Status returned to engine: (0) Success
Status returned to engine: (0) Success
Interface library version 6.0(140)
Data being retrieved for DBQUERY function.
Database adapter version 6.0(140)
```

```
Starting a database unload...
Host string: Userid     : eventmgt
Password   : ********
Query      : SELECT FIRSTNAME, LASTNAME , EMAILADDR FROM EVENTMGT.MEMBERSHIP
             WHERE MEMBERID = 'D191-0002'
Query size : 108
Output is to a buffer.
Statement execution succeeded.
The columns are of the following types:
   Column 1 (FIRSTNAME) type is VARCHAR(30).
   Column 2 (LASTNAME) type is VARCHAR(50).
   Column 3 (EMAILADDR) type is VARCHAR(50).
Number of buffers in fetch array = 492
Writing results to a buffer.
Retrieved 1 records (42 bytes).
Size of DBQUERY data is 42.
The following data was returned from a DBQUERY:
Janet|Leverling|jleverling@northwind.com
.
.
```

In this example, as in the previous one, the database trace file reports the actual query to be executed. In this example, each DBQUERY returns a single row containing the three columns from the **Membership** table for the specified **MEMBERID**. The data returned by the DBQUERY function conforms to the **Delimited Row group format** type tree definition, meaning that there will be a pipe character (|) delimiter between columns and a carriage return/line feed terminator for each row of data.

The size of the data returned by the DBQUERY is also listed. Note that the size of the DBQUERY data is specified as 42. This value represents the length of the data, including the carriage return/line feed that follows the row of data. There will be a similar set of entries in the database trace file for each DBQUERY executed.

Tracing database errors only

Depending upon the number of database sources and targets, the volume of database functions that are executed, and the database trace settings that are used, the database trace file can contain a large amount of data. Even when no database errors occur, the database trace file can become very large.

Use the Trace Error adapter command (-TRACEERR) to minimize the amount of information contained in the database trace file (***map_name*.dbl**). This file contains only the database errors occurring during the map execution.

When -TRACEERR is specified, the database trace file is produced using the full name of the MDQ with a **.dbl** file name extension instead of the usual trace (**.mtr**) extension. It is created in the directory in which the map is located. Use the -TRACEERR *filename* syntax as described in the *Resource Adapters documentation* to specify the name for the database trace file.

A full database trace is most beneficial during map development; however, it is preferable to use -TRACEERR in a production environment because only the database errors are reported.

For example, if you generate the full database trace for a map called **DBUpdate** that has three database input cards, it would produce the following 88-line **MultiDB's.dbl** file.

```
Database type is MS SQL Server
Status returned to engine: (0) Success
No existing connection was found.
Error in: dbopen
Error   : 5701
State   : 2
Message : Changed database context to 'master'.
Error in: dbuse
Error   : 5701
State   : 1
Message : Changed database context to 'pubs'.
A transaction was started.
Connection to SQL Server has been established.
Interface library version 6.0(140)
Data being retrieved for input card 1.
Database adapter version 6.0(140)
Starting a database unload...
Server\\Database: HP_NT\\pubs
Userid          : sa
Password        :
Query           : SELECT * FROM Authors
Query size      : 21
Output is to a buffer.
Statement execution succeeded.
The columns are of the following types:
   Column 1 (au_id) type is varchar(11).
   Column 2 (au_lname) type is varchar(40).
   Column 3 (au_fname) type is varchar(20).
   Column 4 (phone) type is char(12).
   Column 5 (address) type is varchar(40).
   Column 6 (city) type is varchar(20).
   Column 7 (state) type is varchar(2).
   Column 8 (zip) type is varchar(5).
   Column 9 (contract) type is bit.
Number of buffers in fetch array = 1
Writing results to a buffer.
Retrieved 23 records (1810 bytes).
Status returned to engine: (0) Success
Database type is MS SQL Server
Status returned to engine: (0) Success
Interface library version 6.0(140)
Data being retrieved for input card 2.
Database adapter version 6.0(140)
Starting a database unload...
Server\\Database: HP_NT\\pubs
Userid          : sa
Password        :
Query           : SELECT * FROM Publishers
Query size      : 24
Output is to a buffer.
Statement execution succeeded.
The columns are of the following types:
   Column 1 (pub_id) type is char(4).
   Column 2 (pub_name) type is varchar(40).
   Column 3 (city) type is varchar(20).
   Column 4 (state) type is varchar(2).
   Column 5 (country) type is varchar(30).
Number of buffers in fetch array = 1
Writing results to a buffer.
Retrieved 8 records (305 bytes).
Status returned to engine: (0) Success
Database type is MS SQL Server
Status returned to engine: (0) Success
Interface library version 6.0(140)
Data being retrieved for input card 3.
Database adapter version 6.0(140)
Starting a database unload...
```

```
Server\\Database: HP_NT\\pubs
Userid          : sa
Password        :
Query           : SELECT * FROM Title
Query size      : 19
Output is to a buffer.
Error in: dbsqlexec
Error   : 208
State   : 1
Message : Invalid object name 'Title'.
Error in: dbsqlexec
Error   : 10007
Message : General SQL Server error: Check messages from the SQL Server.
Failed to execute statement.
Invalid object name 'Title'.
Retrieved 0 records (0 bytes).
Error returned to engine: (-9) Failed to execute the SQL statement
Cleaning up and closing the transaction...
The transaction was successfully committed.
A transaction was started.
Status returned to engine: (0) Success
```

After reviewing this file, notice that the query for input card number 3 (SELECT * FROM Title) failed because it references an object name (the table name **Title**) that does not exist. Alternatively, you could use the Trace Error adapter command (-TRACEERR) for each of the three input cards and produce the following database trace file:

```
Invalid object name 'Title'.
```

This file can be used along with the following execution audit log information:

```
<SourceReport card="1" adapter="DB" bytes="1810" adapterreturn="0">
   <Message>Success</Message>
   <TimeStamp>01:02:00 January 9, 2004</TimeStamp>
</SourceReport>

<SourceReport card="2" adapter="DB" bytes="305" adapterreturn="0">
   <Message>Success</Message>
   <TimeStamp>01:02:00 January 9, 2004</TimeStamp>
</SourceReport>

<SourceReport card="3" adapter="DB"  bytes="0"  adapterreturn="-9">
   <Message>Failed to execute the SQL statement</Message>
   <TimeStamp>01:02:00 January 9, 2004</TimeStamp>
</SourceReport>
```

Use this information to determine whether the query defined for input card 3 references a table or view that does not exist.

Viewing database source and target data

When debugging a map that uses database sources or targets, you cannot view the database source or target data in the Map Designer by selecting **Run Results** from the **View** menu. Because the data retrieved from a database for an input or written to a database as an output is a snapshot of the data at a given time, it is not available to the Map Designer after the map runs.

However, you can use **Backup** settings to capture the data retrieved from or written to the database for debugging purposes.

Using backup settings

Backup settings are used to determine when, where, and how the data for a specific card should be copied to a specified backup file. These settings are configured in the **Input Card** and **Output Card** settings in the Map Designer and the Launcher or in Command Settings for the Integration Flow Designer. For more information about these settings, see the Map Designer documentation.

Capturing data not processed

The Bad Data adapter command (`-BADDATA`) can be used for a target (or in a `PUT` function). If any inserts, updates, or procedure calls fail to execute, the data that could not be processed is written to a file you specify and processing continues.

The `-BADDATA` command can be specified:

- as part of the **PUT> Target → Command** setting for a card target
- as part of the command line for a database target referenced in a `RUN` or `PUT` function
- on the command line or in a command file.

The rejected record(s) is/are saved in the file specified with the `-BADDATA` adapter command.

When the `-BADDATA` adapter command is used and one or more rows are rejected, the database adapter returns an adapter return code of 1 with a message of One or more records could not be processed as shown in the following TargetReport excerpt in the execution summary section:

```
<TargetReport card="1" adapter="DB" bytes="1218" adapterreturn="1">
   <Message>One or more records could not be processed</Message>
   <TimeStamp>22:07:32 January 9, 2004</TimeStamp>
</TargetReport>
```

Database audit files

Additional troubleshooting and diagnostic information is available in the database adapter audit. Use the Audit adapter command (`-AUDIT`) to create a file that records the adapter activity for each specified database activity. This command can be used for a source or target, or in a `DBLOOKUP`, `DBQUERY`, `GET`, or `PUT` function. This adapter command can be specified for individual input and output cards on a card-by-card basis or, optionally, as a global audit that encompasses all database activity for the entire map.

The default is to produce a file named audit.dbl in the directory in which the map is located. Optionally, you can append the audit information to an existing file or specify a name or the full path for the file. For more information, see the Resource Adapters documentation.

The database adapter audit file provides the following details for each database access:

- Execution Time (**Audit Time**)

  This is the amount of clock time (in seconds) the database adapter takes to execute the database action (for example, the retrieval of all rows for a data source, a single instance of a `DBLOOKUP` function, and so on).
- Adapter Return Code (**AC**)

This is the adapter return code as a result of executing the database action.

- Connection

  This information identifies the connection used for each database action. This can be helpful in determining ways to configure your map to minimize the number of database connections that must be made.

- Map Name (**Map**)

  This is name of the compiled map file.

- Access Type (**Input Card**, **Function**, or **Output Card**)

  This information identifies the type of database action (for example, whether the type is for an input card, output card, or for a `DBLOOKUP`, `DBQUERY`, `GET`, or `PUT` function).

- Additional Information

  This information is provided (when appropriate) for each database action such as the SQL statement associated with an input card, the name of the table or stored procedure for a database target, or the SQL statement executed by any of the functions for interfacing with data in a database (for example, `DBLOOKUP`, `DBQUERY`, `PUT`, and `GET`).

DBMS trace utilities and SQL command tools

The utilities and tools that are part of your relational database management system (RDBMS) will also be helpful during the troubleshooting process. For example, a map fails at runtime because a table name was invalid in an input that is a database source. Try to execute that same query using the tools included with your database to determine whether the query will run natively.

Trace utilities

For example, when using ODBC data sources on a Windows platform, enable ODBC tracing from the ODBC Data Source Administrator window. This creates a log of the calls to ODBC drivers as you use the Database Interface Designer to define databases and queries, and when you use an Launcher to run maps that have database sources or targets. Similar tracing tools are available for most of the other database systems. For information about the tracing capabilities available for your database system, see the documentation for your RDBMS.

SQL command tools

Each database management system includes some form of an SQL command tool. For example, ODBC data sources can be accessed using tools such as Microsoft Query to test queries and view information about your tables, views, and stored procedures. Oracle databases can be accessed using SQL Plus; Microsoft SQL Server data sources can be accessed with ISQL.

These tools provide the ability to determine the problem by testing your queries against the database using the same drivers being accessed by the database adapters.

For example, if you run a map that executes a query for a data source and it fails because one or more of the column names was invalid, you could copy the query text from the Database Interface Designer and test it using the SQL command tool for your database. Then, you could modify the query as necessary to get it to work

correctly. When you achieve the expected results, copy the SQL statement back to the query defined in the Database Interface Designer.

Similarly, if a database insert or update operation fails, you could try entering the corresponding INSERT or UPDATE statement into your database system's SQL command tool to help determine the cause of the failure.

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
577 Airport Blvd., Suite 800
Burlingame, CA 94010
U.S.A.

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

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

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

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

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

This information 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.

COPYRIGHT LICENSE:

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

If you are viewing this information softcopy, the photographs and color illustrations may not appear.

Programming interface information

Programming interface information, if provided, is intended to help you create application software using this program.

General-use programming interfaces allow you to write application software that obtain the services of this program's tools.

However, this information may also contain diagnosis, modification, and tuning information. Diagnosis, modification and tuning information is provided to help you debug your application software.

Warning: Do not use this diagnosis, modification, and tuning information as a programming interface because it is subject to change.

Trademarks and service marks

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

i5/OS
IBM
the IBM logo
AIX
AIX 5L
CICS
CrossWorlds
DB2
DB2 Universal Database
Domino
HelpNow
IMS
Informix
iSeries
Lotus
Lotus Notes
MQIntegrator
MQSeries
MVS
Notes
OS/400
Passport Advantage
pSeries
Redbooks
SupportPac
Tivoli
WebSphere
z/OS

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

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

Intel, Intel logo, Intel Inside, Intel Inside logo, Intel Centrino, Intel Centrino logo, Celeron, Intel Xeon, Intel SpeedStep, Itanium, and Pentium are trademarks or registered trademarks of Intel Corporation or its subsidiaries in the United States and other countries.

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

Linux is a trademark of Linus Torvalds in the United States, other countries, or both.

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

This product includes software developed by the Eclipse Project (http://www.eclipse.org/).

WebSphere Transformation Extender, Version 8.1

Index

Special characters

A

B

C

D

E

F

G

H

I

K

L

M

N

O

P

Q

R

S

T

U

V

W

Printed in USA