IBM InfoSphere Federation Server

Version 10.1

Administration Guide for Federated Systems

SC19-3694-00

IBM InfoSphere Federation Server

Version 10.1

Administration Guide for Federated Systems

SC19-3694-00

Note

Before using this information and the product that it supports, read the information in "Notices and trademarks" on page 437.

Contents

Chapter 1. Federated systems

A *federated system* is a special type of distributed database management system (DBMS). A federated system consists of a DB2® instance that operates as a federated server, a database that acts as the federated database, one or more data sources, and clients (users and applications) that access the database and data sources.

With a federated system, you can send distributed requests to multiple data sources within a single SQL statement. For example, you can join data that is located in a DB2 table, an Oracle table, and an XML tagged file in a single SQL statement. The following figure shows the components of a federated system and a sample of the data sources you can access.

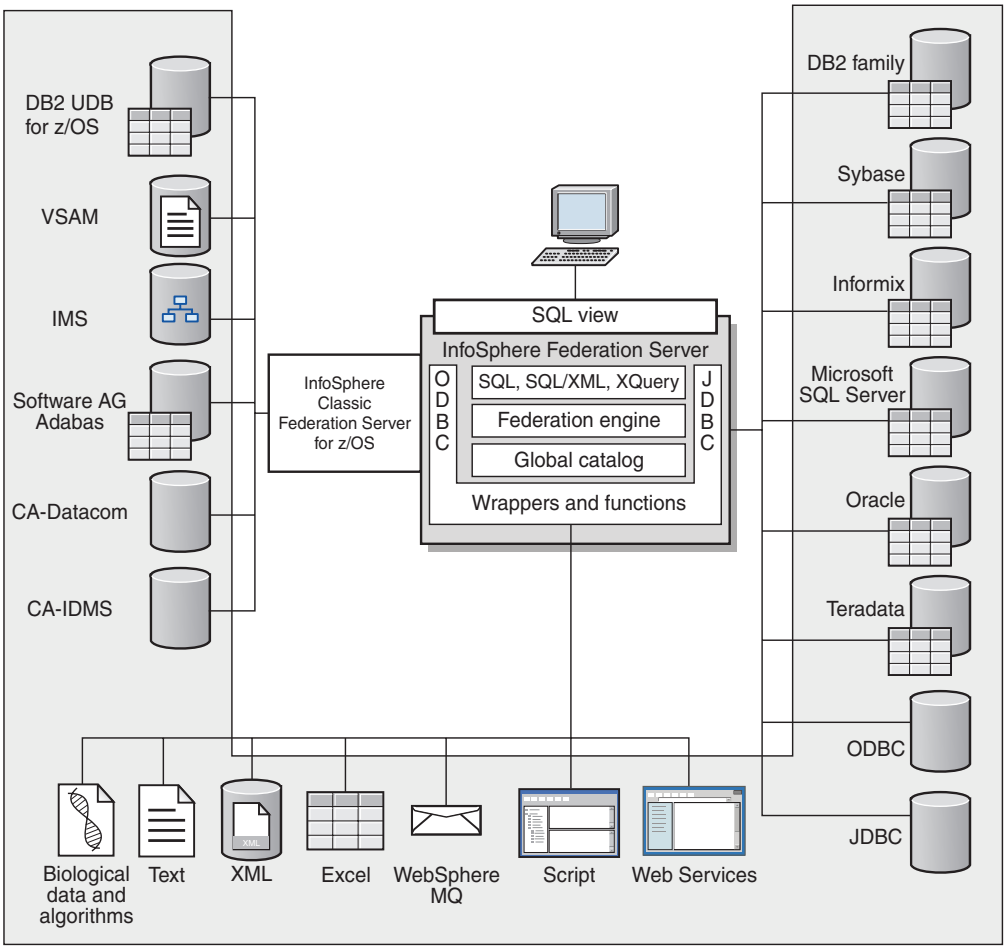


Figure 1. The components of a federated system

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

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

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

Supported data sources

There are many data sources that you can access using a federated system.

IBM® InfoSphere® Federation Server supports the data sources shown in the following table.

For the most up-to-date information about supported data sources, see the Data source requirements page on the Web.

Table 1. Supported data sources.

Data source	Supported versions
BioRS	5.2, 5.3
CA-Datacom	10, 11
CA-IDMS	16, 17
DB2 for Linux, UNIX, and Windows	8.1.x, 8.2.x, 9.1, 9.5, 9.7, 9.8, 10.1
DB2 for z/OS®	7.x, 8.x, 9.x, 10x
DB2 for System i®	5.2, 5.3, 5.4, 6.1
DB2 Server for VSE and VM	7.2 , 7.4
Flat files	
IMS™	10.1, 11.1
Informix®	Informix XPS 8.50, 8.51 and Informix IDS IDS 7.31, IDS 9.40, IDS 10.0, 11.1, 11.5, 11.7
JDBC	3.0 or later
Microsoft Excel	2000, 2002, 2003, 2007
Microsoft SQL Server	Microsoft SQL Server 2000/SP4, 2005, 2008, 2008R2
MQ	MQ7
Netezza	4.6, 5.0, 6.0
ODBC*	3.0
OLE DB	2.7, 2.8
Oracle	10g, 10gR2, 11g, 11gR1, 11gR2
Software AG Adabas	8.1.3, 8.1.4, 8.2.2
Sybase	Sybase ASE 12.5, 15.0, 15.5
Teradata	2.5, 2.6, 12, 13
VSAM	• CICS® Transaction Server V3.1, V3.2, V4.1 • Native VSAM access for the supported z/OS levels • Transactional VSAM access with DFSMS Transactional VSAM Services (DFSMStys) at the supported z/OS levels
Web Services	WSDL 1.0, 1.1 SOAP 1.0, 1.1

Table 1. Supported data sources. (continued)

Data source	Supported versions
XML	XML1.0, XML1.1

* ODBC can be used to access RedBrick 6.20.UC5 and 6.3, InfoSphere Classic Federation Server for z/OS V8.2 and above, and Netezza among other data sources.

The federated server

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

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

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

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

What is a data source?

In a federated system, a *data source* can be a relational database (such as Oracle or Sybase) or a nonrelational data source (such as an XML tagged file).

Through some data sources you can access other data sources. For example, with the ODBC wrapper you can access IBM InfoSphere Classic Federation Server for z/OS data sources such as DB2 for z/OS, IMS, CA-IDMS, CA-Datacom, Software AG Adabas, and VSAM.

The method, or protocol, used to access a data source depends on the type of data source. For example, DRDA is used to access DB2 for z/OS data sources.

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

The federated database

To end users and client applications, data sources appear as a single collective database in the DB2 database system. Users and applications interface with the *federated database* that is managed by the federated server.

The federated database contains a system catalog that stores information about data. The federated database system catalog contains entries that identify data sources and their characteristics. The federated server consults the information stored in the federated database system catalog and the data source wrapper to determine the best plan for processing SQL statements.

The federated system processes SQL statements as if the data from the data sources were ordinary relational tables or views within the federated database. As a result:

- The federated system can correlate relational data with data in nonrelational formats. This is true even when the data sources use different SQL dialects, or do not support SQL at all.
- The characteristics of the federated database take precedence when there are differences between the characteristics of the federated database and the characteristics of the data sources. Query results conform to DB2 semantics, even if data from other non-DB2 data sources is used to compute the query result.

  Examples:

  – The code page that the federated server uses is different than the code page used that the data source uses. In this case, character data from the data source is converted based on the code page used by the federated database, when that data is returned to a federated user.
  – The collating sequence that the federated server uses is different than the collating sequence that the data source uses. In this case, any sort operations on character data are performed at the federated server instead of at the data source.

Wrappers and wrapper modules

Wrappers are mechanisms by which the federated database interacts with data sources. The federated database uses routines stored in a library called a *wrapper module* to implement a wrapper.

These routines allow the federated database to perform operations such as connecting to a data source and retrieving data from it iteratively. Typically, the federated instance owner uses the CREATE WRAPPER statement to register a wrapper in the federated database. You can register a wrapper as fenced or trusted using the DB2_FENCED wrapper option.

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

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

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

- It connects to the data source. The wrapper uses the standard connection API of the data source.
- It submits queries to the data source.
  - For data sources that support SQL, the query is submitted in SQL.
  - For data sources that do not support SQL, the query is translated into the native query language of the source or into a series of source API calls.
- It receives results sets from the data source. The wrapper uses the data source standard APIs for receiving results set.
- It responds to federated database queries about the default data type mappings for a data source. The wrapper contains the default type mappings that are used when nicknames are created for a data source object. For relational wrappers, data type mappings that you create override the default data type mappings. User-defined data type mappings are stored in the global catalog.
- It responds to federated database queries about the default function mappings for a data source. The federated database needs data type mapping information for query planning purposes. The wrapper contains information that the federated database needs to determine if DB2 functions are mapped to functions of the data source, and how the functions are mapped. This information is used by the SQL Compiler to determine if the data source is able to perform the query operations. For relational wrappers, function mappings that you create override the default function type mappings. User-defined function mappings are stored in the global catalog.

Wrapper options are used to configure the wrapper or to define how IBM InfoSphere Federation Server uses the wrapper.

Federated systems and DB2 pureScale

DB2 for Linux, UNIX, and Windows introduced the DB2 pureScale® Feature in Version 9.8. In Version 10, Federation Server functions are integrated with the DB2 pureScale environment.

You can use federation with DB2 pureScale to take advantage of the availability and scalability that the DB2 pureScale feature offers. For detailed information about the pureScale Feature, see Introduction to the IBM DB2 pureCluster Feature.

Federation functions can operate on any member of a DB2 pureScale instance. Those functions include the majority of functions that federation supports such as insert, update, and delete operations, and stored procedures.

With automatic client reroute and workload balancing, every member of a DB2 pureScale instance can run federated statements. It is important to install client libraries on all DB2 pureScale members.

How you interact with a federated system

Because the federated database is a DB2 family database, you can interact with it in a variety of ways.

You can interact with a federated system using by any one of these methods:

- The DB2 command line processor (CLP)
- Application programs

- DB2 family tools
- Web services providers

The steps in the federated documentation provide the commands and SQL statements that can be entered in the DB2 command line processor. The documentation indicates when tasks can be performed through the IBM Data Studio GUI.

IBM Data Studio

IBM Data Studio is a comprehensive data management solution that you can use to work with the federated objects that you create and manage to configure and administer IBM InfoSphere Federation Server.

IBM Data Studio consist of the following three components:

IBM Data Studio administration client
Provides a stand-alone integrated development environment with a small-footprint for database administration and routine development in programming languages other than Java.

IBM Data Studio full client
Provides an integrated development environment for database administration, routine development, and Java application development. It can be installed with other IBM software products to share a common environment.

IBM Data Studio web console
Provides a web-based tool with health and availability monitoring, job creation, and database administration functions.

If you previously used the Control Center tools, review the mapping between the recommended IBM Data Studio and IBM InfoSphere Optim™ tools and the Control Center tools. See Table of recommended tools versus Control Center tools.

DB2 command line processor (CLP)

You can perform most of the tasks necessary to setup, configure, tune, and maintain the federated system through the DB2 command line processor.

For example, you can use the DB2 command line processor to perform the following tasks:

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

Application programs

Applications do not require any special coding to work with federated data. Applications access the system just like any other DB2 client application. Applications interface with the federated database that is within the federated server.

To obtain data from data sources, applications submit queries in DB2 SQL to the federated database. The federated database then distributes the queries to the appropriate data sources, collects the requested data, and returns this data to the applications. However, since the federated database interacts with the data sources through nicknames, you need to be aware of:

- The SQL restrictions you have when working with nicknames

- How to perform operations on nicknamed objects

DB2 family tools

You can interact with a federated database using host and midrange tools.

Host and midrange tools that you can use to interact with a federated database include:

- DB2 tools that enhance performance and management for DB2 for z/OS and DB2 for Linux, UNIX, and Windows
- Interactive SQL (STRSQL) on DB2 for System i

Web services providers

You can interact with a federated database through web services providers.

Within federated systems, a Web services wrapper is available to access Web services with SQL statements on nicknames and views that invoke Web services. You can create a Web services wrapper and nicknames that specify input to the Web service and access the output from the Web service with SELECT statements.

The federated database system catalog

The federated database system catalog contains information about the objects in the federated database and information about objects at the data sources.

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

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

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

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

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

The SQL compiler

The DB2 SQL compiler gathers information to help it process queries.

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

The query optimizer

As part of the SQL compiler process, the query optimizer analyzes a query. The compiler develops alternative strategies, called access plans, for processing the query.

Access plans might call for the query to be:

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

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

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

The query optimizer generates access plan alternatives for processing a query fragment. The plan alternatives perform varying amounts of work locally on the federated server and on the remote data sources. Because the query optimizer is cost-based, it assigns resource consumption costs to the access plan alternatives. The query optimizer then chooses the plan that will process the query with the least resource consumption cost.

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

Compensation

The ability by IBM InfoSphere Federation Server to process SQL that is not supported by a data source is called *compensation*.

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

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

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

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

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

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

Pass-through sessions

You can submit SQL statements directly to data sources by using a special mode called *pass-through*.

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

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

Similarly, you can use a pass-through session to perform actions that are not supported by SQL, such as certain administrative tasks. However, you cannot use a

pass-through session to perform all administrative tasks. For example, you can create or drop tables at the data source, but you cannot start or stop the remote database.

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

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

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

SET PASSTHRU RESET
Terminates the current pass-through session.

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

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

Server definitions and server options

After wrappers are created for the data sources, the federated instance owner defines the data sources to the federated database.

The instance owner supplies a name to identify the data source, and other information that pertains to the data source. This information includes:

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

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

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

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

Some of the information within a server definition is stored as *server options*. When you create server definitions, it is important to understand the options that you can specify about the server.

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

User mappings

A *user mapping* is an association between an authorization ID on the federated server and the information that is required to connect to the remote data source.

To create a user mapping, you specify the following information in the CREATE USER MAPPING statement:

- local authorization ID
- local name of the remote data source server as specified in the server definition
- remote ID and password

For example, assume that you created a server definition for a remote server and specified 'argon' as the local name for the remote server. To give Mary access to the remote server, create this user mapping:

```
CREATE USER MAPPING FOR Mary
SERVER argon
OPTIONS (REMOTE_AUTHID 'remote_ID', REMOTE_PASSWORD 'remote_pw')
```

When Mary issues an SQL statement to connect to the remote server, the federated server performs these steps:

1. Retrieves Mary's user mapping
2. Decrypts the remote password 'remote_pw' that is associated with the remote server
3. Calls the wrapper to connect to the remote server
4. Passes the remote ID 'remote_ID' and the decrypted remote password to the wrapper
5. Creates a connection to the remote server for Mary

By default, the federated server stores user mapping in the SYSCAT.USEROPTIONS view in the global catalog and encrypts the remote passwords. As an alternative, you can use an external repository, for example a file or an LDAP server, to store user mappings. To provide the interface between the federated server and the external repository, you create a user mapping plug-in.

No matter how you store user mappings, carefully restrict access to them. If user mappings are compromised, data in the remote databases might be vulnerable to unauthorized activity.

In InfoSphere Federation Server Version 9.7 Fix Pack 2 and later, you can also create public user mappings to allow all local database users to access a data source through a single remote user ID and password.

Nicknames and data source objects

A *nickname* is an identifier that you use to identify the data source object that you want to access. The object that a nickname identifies is referred to as a*data source object*.

A nickname is not an alternative name for a data source object in the same way that an alias is an alternative name. A nickname is the pointer by which the federated server references the object. Nicknames are typically defined with the CREATE NICKNAME statement along with specific nickname column options and nickname options.

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

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

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

Nicknames for objects that use label-based access control (LBAC) are not cached. Therefore, data in the object remains secure. For example, if you use the Oracle (Net8) wrapper to create a nickname on a table that uses Oracle Label Security, the table is automatically identified as secure. The resulting nickname data cannot be cached. As a result, materialized query tables cannot be created on it. Using LBAC ensures that the information is viewed only by users who have the appropriate security privileges. For nicknames that were created before LBAC was supported, you must use the ALTER NICKNAME statement to disallow caching. LBAC is supported by both the DRDA (for data sources that use DB2 for Linux, UNIX, and Windows version 9.1 and later) and the Net8 wrapper.

Valid data source objects

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

Table 2. Valid data source objects

Data source	Valid objects
BioRS	BioRS databanks
DB2 Database for Linux, UNIX, and Windows	Nicknames, materialized query tables, tables, views
DB2 for System i	Tables, views, P/L (physical/logical) files and table types
DB2 for VM and VSE	Tables, views
DB2 for z/OS	Tables, views
Informix	Tables, views, synonyms
JDBC	All table types
Microsoft Excel	.xls files (only the first sheet in the workbook is accessed)
Microsoft SQL Server	Tables, views
ODBC	Tables, views

Table 2. Valid data source objects (continued)

Data source	Valid objects
Oracle	Tables, views, synonyms
Script	Scripts
Sybase	Tables, views
Teradata	Tables, views
Table-structured files	Text files that meet a specific format.
Web Services	Operations in a Web services description language file
XML-tagged files	Sets of items in an XML document

Nickname column options

You can supply the global catalog with additional metadata information about the nicknamed object. This metadata describes values in certain columns of the data source object. You assign this metadata to parameters that are called *nickname column options*.

The nickname column options tell the wrapper to handle the data in a column differently than it normally would handle it. The SQL complier and query optimizer use the metadata to develop better plans for accessing the data.

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

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

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

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

Data type mappings

The data types at the data source must map to corresponding DB2 data types so that the federated server can retrieve data from data sources.

Some examples of default data type mappings are:

- The Oracle type FLOAT maps to the DB2 type DOUBLE
- The Oracle type DATE maps to the DB2 type TIMESTAMP
- The DB2 for z/OS™ type DATE maps to the DB2 type DATE

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

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

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

Function mappings

For the federated server to recognize a data source function, the function must be mapped to an existing counterpart function in DB2 Database for Linux, UNIX and Windows.

IBM InfoSphere Federation Server supplies default mappings between existing data source functions and DB2 counterpart functions. For most data sources, the default function mappings are in the wrappers. The default function mappings to DB2 for z/OS functions are in the DRDA wrapper. The default function mappings to Sybase functions are in the CTLIB wrapper, and so forth.

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

Index specifications

When you create a nickname for a data source table, information about any indexes that the data source table has is added to the global catalog. The query optimizer uses this information to expedite the processing of distributed requests. The catalog information about a data source index is a set of metadata, and is called an index specification.

The query optimizer uses this information to expedite the processing of distributed requests.

A federated server does not create an index specification when you create a nickname for the following objects:

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

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

Federated stored procedures

Federated procedure access enables users of federated systems to access remote stored procedures at remote data sources.

A federated stored procedure is a local stored procedure that is mapped to a stored procedure at a data source. You use the CREATE PROCEDURE (Sourced) statement to register a federated stored procedure.

Collating sequences

The order in which character data is sorted in a database depends on the structure of the data and the collating sequence defined for the database.

Suppose that the data in a database is all uppercase letters and does not contain any numeric or special characters. A sort of the data should result in the same output, regardless of whether the data is sorted at the data source or at the federated database. The collating sequence used by each database should not impact the sort results. Likewise, if the data in the database is all lowercase letters or all numeric characters, a sort of the data should produce the same results regardless of where the sort actually is performed.

If the data consists of any of the following structures:

- A combination of letters and numeric characters
- Both uppercase and lowercase letters
- Special characters such as @, #, €

Sorting this data can result in different outputs, if the federated database and the data source use different collating sequences.

In general terms, a collating sequence is a defined ordering for character data that determines whether a particular character sorts higher, lower, or the same as another character.

How collating sequences determine sort orders

A collating sequence determines the sort order of the characters in a coded character set.

A character set is the aggregate of characters that are used in a computer system or programming language. In a coded character set, each character is assigned to a different number within the range of 0 to 255 (or the hexadecimal equivalent thereof). The numbers are called code points; the assignments of numbers to characters in a set are collectively called a code page.

In addition to being assigned to a character, a code point can be mapped to the character's position in a sort order. In technical terms, then, a collating sequence is the collective mapping of a character set's code points to the sort order positions of the set's characters. A character's position is represented by a number; this number is called the weight of the character. In the simplest collating sequence, called an identity sequence, the weights are identical to the code points.

Example: Database ALPHA uses the default collating sequence of the EBCDIC code page. Database BETA uses the default collating sequence of the ASCII code page. Sort orders for character strings at these two databases would differ:

```
SELECT.....
  ORDER BY COL2

EBCDIC-Based Sort           ASCII-Based Sort

COL2                        COL2
----                        ----
V1G                         7AB
Y2W                         V1G
7AB                         Y2W
```

Example: Similarly, character comparisons in a database depend on the collating sequence defined for that database. Database ALPHA uses the default collating sequence of the EBCDIC code page. Database BETA uses the default collating sequence of the ASCII code page. Character comparisons at these two databases would yield different results:

```
SELECT.....
  WHERE COL2 > 'TT3'

EBCDIC-Based Results     ASCII-Based Results

COL2                     COL2
----                     ----
TW4                      TW4
X82                      X82
39G
```

Setting the local collating sequence to optimize queries

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

Then for each data source server definition, the COLLATING_SEQUENCE server option is set to 'Y'. This setting tells the federated database that the collating sequences of the federated database and the data source match.

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

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

Suppose that the data source is DB2 for z/OS. Sorts that are defined in an ORDER BY clause are implemented by a collating sequence based on an EBCDIC code page. To retrieve DB2 for z/OS data sorted in accordance with ORDER BY clauses, configure the federated database so that it uses the predefined collating sequence based on the appropriate EBCDIC code page.

Chapter 2. Modifying data source configurations

Over time you might find the need to modify the data source configurations for your federated system. For example, if your data sources change, you might need to update server definitions, nicknames, and user mappings. You might also need to add or remove access to data sources from your federated system.

The examples of altering federated objects explain how to alter objects by issuing SQL statements from the DB2 command line. You can also use the SQL and XQuery editor in Data Studio to issue SQL scripts. For the supported DB2 data sources and the Oracle data source, you can use the Administration Explorer in Data Studio that provides dialog boxes and wizards that guide you through the task of altering federated objects.

Altering wrappers

- **Altering a wrapper**

Altering server definitions and options

- **Altering server definitions and server options**
- **Using server options in server definitions**

Altering user mappings

- **Altering a user mapping**

Altering nicknames

- **Altering a nickname**

Dropping wrappers, server definitions, user mappings, and nicknames

- **Dropping a wrapper**
- **Dropping a server definition**
- **Dropping a user mapping**
- **Dropping a nickname**

Altering a wrapper

After you configure a wrapper, you can use the ALTER WRAPPER statement to modify the configuration based on your system requirements.

Before you begin

The authorization ID associated with the statement must have SYSADM or DBADM authority.

About this task

This task describes how to alter a wrapper from the DB2 command line. You can use the ALTER WRAPPER statement to add, set, or drop one or more wrapper options.

Restrictions:

- You cannot drop the DB2_FENCED wrapper option.
- The federated server cannot process an ALTER WRAPPER statement within a given unit of work if the unit of work already includes one of the following statements:
  - A SELECT statement that references a nickname for a table or view at the data source that the wrapper includes
  - An open cursor on a nickname for a table or view at the data source that the wrapper includes
  - An insert, delete, or update issued against a nickname for a table or view at the data source that the wrapper includes

Procedure

To alter a wrapper from the DB2 command line, issue the ALTER WRAPPER statement.

Altering a wrapper - examples

This topic provides examples of modifying wrapper options with the ALTER WRAPPER statement.

To change the DB2_FENCED option to 'Y' for the wrapper named drda, issue the following statement:

```
ALTER WRAPPER drda OPTIONS (SET DB2_FENCED 'Y');
```

To change the MODULE option to '/opt/odbc/lib/libodbc.a(odbc.so)' for the wrapper named odbc , issue the following statement:

```
ALTER WRAPPER odbc OPTIONS (SET MODULE '/opt/odbc/lib/libodbc.a(odbc.so)');
```

Altering server definitions and server options

You use the ALTER SERVER statement to modify a server definition. Some of the information within a server definition is stored as server options. When you modify a server definition, it is important to understand the options that you can specify about the server.

A server definition identifies a data source to the federated database. The server definition consists of a local name and other information about that data source server. The server definition is used by the wrapper when SQL statements that use nicknames are submitted to the federated database.

For relational data sources, server options can be set temporarily using the SET SERVER OPTION statement. This statement overrides the server option value in the server definition for the duration of a single connection to the federated database. For static SQL, using the SET SERVER OPTION statement affects only the execution of the static SQL statement. Using the SET SERVER OPTION statement has no affect on the plans that the optimize generates.

Modify a server definition when:

- You upgrade to a new version of the data source
- You want to make the same change to all of the server definitions for a specific data source type
- You want to add or change a server option on an existing server definition

Restrictions on altering server definitions

You need to be aware of several restrictions when you alter server definitions.

The following restrictions apply to altering server definitions:

- You cannot specify a wrapper in the ALTER SERVER statement that is not registered with the federated server.
- The federated server cannot process an ALTER SERVER statement within a given unit of work (UOW) under either of the following conditions:
  - The statement references a single data source, and the UOW already includes one of the following statements:
    - A SELECT statement that references a nickname for a table or view within the data source
    - An open cursor on a nickname for a table or view within the data source
    - An insert, delete or update issued against a nickname for a table or view within the data source
  - The statement references a category of data sources (for example, all data sources of a specific type and version), and the UOW already includes one of the following statements:
    - A SELECT statement that references a nickname for a table or view within one of the data sources
    - An open cursor on a nickname for a table or view within one of the data sources
    - An insert, delete or update issued against a nickname for a table or view within one of the data sources
- The federated server does not verify whether the server version that you specify matches the remote server version. Specifying an incorrect server version can result in SQL errors when you access nicknames that belong to the DB2 server definition. These SQL errors are most likely to occur when you specify a server version that is later than the actual version of the remote server. In that case, when you access nicknames that belong to the server definition, the federated server might send SQL to the remote server that the remote server does not recognize. On the ALTER SERVER statement, this situation applies only to the form of the statement that alters the server version (*server-name* VERSION server-version).
- You must specify the full server option name. For example, you cannot specify the abbreviation DB2_TWO_PHASE. Instead, you need to specify the full server option name DB2_TWO_PHASE_COMMIT.

Altering the data source version in a server definition

You can alter an existing server definition to change the version of the data source that the remote server uses.

Before you begin

The authorization ID issuing the ALTER SERVER statement must include either SYSADM or DBADM authority on the federated database.

About this task

Restrictions

See Restrictions on altering server definitions.

Procedure

To alter a server definition from the DB2 command line, issue the ALTER SERVER statement.
Example: You are working with a server definition for a Microsoft SQL Server Version 6.5 data source. The name that you assigned the server in the CREATE SERVER statement is SQLSVR_ASIA. If the Microsoft SQL Server is upgraded to Version 7.0, the following statement alters the server definition:

```
ALTER SERVER SQLSVR_ASIA VERSION 7
```

Altering all of the server definitions for a specific data source type

You can alter all of the existing server definitions for a particular type of data source in a single ALTER SERVER statement. Use a single statement when you want the same change to apply to all of the server definitions of the same type.

Before you begin

The authorization ID issuing the ALTER SERVER statement must include either SYSADM or DBADM authority on the federated database.

About this task

Restrictions

You can only set or drop server options using the ALTER SERVER statement for an entire type of data sources if the server options were added by a prior ALTER SERVER statement operation.

Procedure

To alter all of the existing server definitions for a particular type of data source, issue a single ALTER SERVER statement.
Example: Five Sybase servers are registered in the global catalog for your Sybase data sources. Whenever a user ID is sent to any of these Sybase servers for authentication, you want the federated server to always fold the user ID to uppercase. In addition, you want to set how long the federated server will wait for a response from these Sybase servers to an SQL statement. You specify the amount of time in seconds. You can modify all five server definitions at the same time the following ALTER SERVER statement:

```
ALTER SERVER TYPE sybase
   OPTIONS (ADD FOLD_ID 'U', ADD TIMEOUT '600')
```

Using server options in server definitions

There are general server options and server options that are for specific data source types.

Before you begin

The authorization ID issuing the ALTER SERVER statement must include either SYSADM or DBADM authority on the federated database.

About this task

Restrictions

See “Restrictions on altering server definitions” on page 21.

Server options are set to values that persist over successive connections to the data source. These values are stored in the federated system catalog.

Procedure

To do this task from the command line prompt, issue the ALTER SERVER statement.

Example:

- You created a server definition for an Informix server using the server name of INFMX01. You now want to change the DB2_MAXIMAL_PUSHDOWN option to Y. The statement to alter the server definition is:

  ```
  ALTER SERVER INFMX01 OPTIONS (SET DB2_MAXIMAL_PUSHDOWN 'Y')
  ```

- You created a server definition for an Oracle server using the server name of ORCL99. You now want to add the FOLD_ID and FOLD_PW options to the definition. The statement to alter the server definition is:

  ```
  ALTER SERVER ORCL99 OPTIONS (ADD FOLD_ID 'U', FOLD_PW 'U')
  ```

- You want to set the timeout value to the number of seconds the CTLIB wrapper should wait for a response from the Sybase server. You use the TIMEOUT server option to set this value. The statement to alter the server definition is:

  ```
  ALTER SERVER SYBSERVER OPTIONS (ADD TIMEOUT '60')
  ```

Changing server options temporarily for relational data sources

The SET SERVER OPTION statement overrides the server option value in the server definition for the duration of a single connection to the federated database. The overriding value is not stored in the global catalog.

About this task

For static SQL, using the SET SERVER OPTION statement affects only the execution of the static SQL statement. Using the SET SERVER OPTION statement has no affect on the plans that the optimize generates.

Procedure

To set a server option value temporarily for a relational data source, use the SET SERVER OPTION statement.

Example:

```
SET SERVER OPTION PLAN_HINTS TO Y' FOR SERVER SYB_SERVER
```

The hierarchy of server option settings

When you have the same server option set with one value for a data source type and set with another value on a specific data source server, there is a hierarchy among the settings.

For example, the PLAN_HINTS server option is set to 'Y' for the data source type SYBASE. However, the PLAN_HINTS server option is set to 'N' in the server

definition for a specific Sybase data source server PURNELL. The setting for the specific data source server overrides the setting for the data source type. This configuration causes PLAN_HINTS to be enabled at all Sybase data source servers except PURNELL.

Altering a user mapping

A user mapping is the association between the authorization ID at the federated server and the authorization ID at the data source. User mappings are needed so that distributed requests can be sent to the data source.

Before you begin

If the authorization ID issuing the statement is different than the authorization ID that is mapped to the data source, then the authorization ID issuing the statement must include either SYSADM or DBADM authority on the federated database.

About this task

The ALTER USER MAPPING statement is used to change the authorization ID or password that is used at the data source for a specific federated server authorization ID.

Restrictions

The federated server cannot process an ALTER USER MAPPING statement within a given unit of work (UOW) if the UOW already includes one of the following statements:

- A SELECT statement that references a nickname for a table or view at the data source the mapping includes
- An open cursor on a nickname for a table or view at the data source that the mapping includes
- An insert, delete, or update issued for a nickname of a table or view at the data source the mapping includes

Procedure

To alter a user mapping from the DB2 command line, issue the ALTER USER MAPPING statement.
Example: Jenny uses the federated server to connect to a Sybase server called SYBSERVER. She accesses the federated server with the authorization ID of *jennifer*. The authorization ID *jennifer* is mapped to the authorization ID *jenn* on the Sybase server. The authorization ID for Jenny on the Sybase server is changed to *jen123*. The ALTER USER MAPPING statement to map *jennifer* to *jen123* is:

```
ALTER USER MAPPING FOR jennifer SERVER SYBSERVER
   OPTIONS (SET REMOTE_AUTHID 'jen123')
```

Tomas uses the federated server to connect to an Oracle server called ORASERVER. He accesses the federated server with the authorization ID of *tomas*. The authorization ID *tomas* is mapped to the authorization ID *tom* on the Oracle server. The password for Tomas on the Oracle server is changed. His new password is day2night. The ALTER USER MAPPING statement to map *tomas* to the new password is:

```
ALTER USER MAPPING FOR tomas SERVER ORASERVER
   OPTIONS (SET REMOTE_PASSWORD 'day2night')
```

The REMOTE_AUTHID and REMOTE_PASSWORD user options are case sensitive unless you set the FOLD_ID and FOLD_PW server options to 'U' or 'L' in the CREATE SERVER statement.

Altering a nickname

Nicknames are identifiers that are used to reference an object that you want to access at a data source. You can change the data source column names that are stored in the global catalog and set column options by altering the nicknames.

Before you begin

The privileges held by the authorization ID of the statement must include at least one of the following:

- SYSADM or DBADM authority
- ALTER privilege on the nickname specified in the statement
- CONTROL privilege on the nickname specified in the statement
- ALTERIN privilege on the schema, if the schema name of the nickname exists
- Definer of the nickname as recorded in the DEFINER column of the catalog view for the nickname

About this task

You might want to alter a nickname to:

- Alter the local column names for the columns of the data source object
- Alter the local data types for the columns of the data source object
- Add, set, or drop nickname and column options
- Add or drop a primary key
- Add or drop one or more unique, referential, or check constraints
- Alter one or more referential, check, or functional dependency constraint attributes

Restrictions

See “Restrictions on altering nicknames.”

Procedure

To alter a nickname from the DB2 command line, issue the ALTER NICKNAME statement with the appropriate parameters set.
When the data source object structure or content changes significantly, you should update the nickname statistics. Significant changes include the addition or removal of multiple rows.

Restrictions on altering nicknames

You need to be aware of several restrictions when you alter a nickname.

Column options

If one of the following options is set on a column, you cannot add any other options to that column:

- SOAPACTIONCOLUMN
- URLCOLUMN

- PRIMARY_KEY
- FOREIGN_KEY

For BioRS

- If you change the element name of a column by using the ELEMENT_NAME option, the new name is not checked to ensure that it is correct. An incorrect option might result in errors when the column is referenced in a query.
- If you make changes to the IS_INDEXED column option, the changes are not verified with the BioRS server. An incorrect option might result in errors when the column is referenced in a query.

Data types

- If you change the data type of a column, the new data type must be compatible with the data type of the corresponding data source column or element. Changing the local data type to a data type that is incompatible with the remote data type might cause unpredictable errors.
- The *local_data_type* cannot be LONG VARCHAR, LONG VARGRAPHIC, XML, or a user-defined data type.
- The *data_source_data_type* cannot be a user-defined type.
- You cannot override the existing local types or create new local types for some of the nonrelational data sources. Check the documentation for the specific data source wrapper for more information on this restriction.
- When the local specification of a column's data type is changed, the federated database manager invalidates any statistics (for example, HIGH2KEY and LOW2KEY) that are gathered for that column.
- The local type is set for the specific data source object when it is accessed with that nickname. The same data source object can have another nickname that uses the default data type mapping.

Indexes

The ALTER NICKNAME statement cannot be used to register a new data source index in the federated database. Use the CREATE INDEX statement with the SPECIFICATION ONLY clause to create an index specification.

LOCAL NAME and LOCAL TYPE parameters

- ALTER NICKNAME statement cannot be used to change the local names or data types for the columns in the nickname if:
  - The nickname is used in a view, SQL method, or SQL function
  - You define an informational constraint on the nickname
- The federated_column_options clause must be specified last if you also need to specify the LOCAL NAME parameter, the LOCAL TYPE parameter, or both in the ALTER NICKNAME statement.

Nicknames

The ALTER NICKNAME statement cannot be used to change the name of the BioRS databank that is referenced by or used in a BioRS nickname. If the name of a BioRS databank changes, you must drop the nickname and create the nickname again.

You cannot use the ALTER NICKNAME statement to disallow caching on a nickname with cache tables or materialized query tables. You must drop the cache tables and the materialized query tables before you disallow caching on the nickname.

Units of work

The federated server cannot process an ALTER NICKNAME statement within a given unit of work under any of the following conditions:

- If the nickname referenced in the ALTER NICKNAME statement has a cursor open on it in the same unit of work.
- If an insert, delete or update is issued in the same unit of work for the nickname that is referenced in the ALTER NICKNAME statement.

Altering nickname column names

You can alter a nickname to change the column names.

Before you begin

The authorization ID issuing the statement must include at least one of the following privileges:

- SYSADM or DBADM authority
- ALTER privilege on the nickname specified in the statement
- CONTROL privilege on the nickname specified in the statement
- ALTERIN privilege on the schema, if the schema name of the nickname exists
- Definer of the nickname as recorded in the DEFINER column of the catalog view for the nickname

About this task

When you create a nickname, the column names that are associated with the data source object are stored in the federated database. For some data sources, the wrapper specifies the column names. For other data sources, you must specify the column names when you create the nickname.

Restrictions

See "Restrictions on altering nicknames" on page 25.

Procedure

To alter nickname column names from the DB2 command line, issue the ALTER NICKNAME statement.

```
ALTER NICKNAME nickname
   ALTER COLUMN current_name
   LOCAL NAME new_name
```

Altering nickname options

Nickname options are parameters that you specify on the nickname when you issue the CREATE NICKNAME and ALTER NICKNAME statements. You can add, set, or drop nickname options by using the ALTER NICKNAME statement.

Before you begin

The authorization ID issuing the statement must include at least one of the following privileges:

- SYSADM or DBADM authority
- ALTER privilege on the nickname specified in the statement
- CONTROL privilege on the nickname specified in the statement

- ALTERIN privilege on the schema, if the schema name of the nickname exists
- Definer of the nickname as recorded in the DEFINER column of the catalog view for the nickname

About this task

Restrictions

See "Restrictions on altering nicknames" on page 25.

Procedure

To change a nickname option from the command line prompt, issue the ALTER NICKNAME statement.

```
ALTER NICKNAME nickname
    OPTIONS (SET option_name 'option_string_value')
```

Example: The nickname DRUGDATA1 is created for the table-structured file `drugdata1.txt`. The fully qualified path that was originally defined in the CREATE NICKNAME statement was `/user/pat/drugdata1.txt`. To change the FILE_PATH nickname option, issue the following statement:

```
ALTER NICKNAME DRUGDATA1 OPTIONS (SET FILE_PATH '/usr/kelly/data/drugdata1.txt')
```

Altering nickname column options

You can add, set, or drop nickname column options using the ALTER NICKNAME statement.

Before you begin

The authorization ID issuing the statement must include at least one of the following privileges:

- SYSADM or DBADM authority
- ALTER privilege on the nickname specified in the statement
- CONTROL privilege on the nickname specified in the statement
- ALTERIN privilege on the schema, if the schema name of the nickname exists
- Definer of the nickname as recorded in the DEFINER column of the catalog view for the nickname

About this task

You specify column information in the CREATE NICKNAME and ALTER NICKNAME statements by using parameters called nickname column options. You can specify any of these values in either uppercase or lowercase letters.

Restrictions

See "Restrictions on altering nicknames" on page 25.

Procedure

To alter nickname column options from the command line prompt, use the ALTER NICKNAME statement.
The first time that you add an option, use the ADD keyword to add the option. If the option was added previously, use the SET keyword to change the option.

Example 1: Specifying the NUMERIC_STRING column option with relational data sources
The NUMERIC_STRING column option applies to character type columns (CHAR and VARCHAR). Suppose that a data source has a collating sequence that differs from the federated database collating sequence. The federated server typically would not sort any columns containing character data at the data source. It would return the data to the federated database and perform the sort locally. However, suppose that the column is a character data type and contains only numeric characters ('0','1',...,'9'). You can indicate this by assigning a value of 'Y' to the NUMERIC_STRING column option. This gives the DB2 UDB query optimizer the option of performing the sort at the data source. If the sort is performed remotely, you can avoid the overhead of sorting the data at the federated server.
The nickname ORA_INDSALES for an Oracle table called INDONESIA_SALES. The table contains the column POSTAL_CODE with the data type of VARCHAR. Originally the column contained only numeric characters, and the NUMERIC_STRING column option was set to 'Y'. However, the column now contains a mixture of numeric and non-numeric characters. To change the NUMERIC_STRING column option to 'N', use this statement:

```
ALTER NICKNAME ORA_INDSALES ALTER COLUMN POSTAL_CODE
   OPTIONS (SET NUMERIC_STRING 'N')
```

Example 2: Specifying the VARCHAR_NO_TRAILING_BLANKS column option with relational data sources
The VARCHAR_NO_TRAILING_BLANKS column option can be used to identity specific columns that contain no trailing blanks. The SQL Compiler will factor in this setting when it checks for all operations (such as comparison operations) performed on columns.
The nickname ORA_INDSALES is for an Oracle table called INDONESIA_SALES. The table contains the column NAME with the data type of VARCHAR. The NAME column does not have trailing blanks. To add the VARCHAR_NO_TRAILING_BLANKS option to the nickname, use this statement:

```
ALTER NICKNAME ORA_INDSALES ALTER COLUMN NAME
   OPTIONS (ADD VARCHAR_NO_TRAILING_BLANKS 'Y')
```

Example 3: Specifying the XPATH column option with nonrelational data sources
The nickname EMPLOYEE is for an XML data source. An XPATH was specified for the *fnmae* column. To set the XPATH column option to a different path, use this statement:

```
ALTER NICKNAME EMPLOYEE ALTER COLUMN fname
   OPTIONS (SET XPATH './@first')
```

Dropping a wrapper

There are several reasons why you might want to drop a wrapper.

Before you begin

To issue the DROP WRAPPER statement, you must have SYSADM or DBADM authority.

About this task

Sometimes there is more than one wrapper that you can use to access a data source. The one you choose might depend on the version of the data source client software that you are using. Or it might depend on the operating system that you are using on your federated server. Suppose that you want to access two Oracle

tables and one Oracle view. You are using Oracle Version 10, and the operating system on your federated server is Windows. Originally you created the SQLNET wrapper. Since IBM InfoSphere Federation Server does not support the SQLNET wrapper, you can drop the SQLNET wrapper and create the NET8 wrapper.

Another reason to drop a wrapper is that you no longer need access to the data source that the wrapper is associated with. For example, suppose that your organization has a requirement to access client information in both Informix and Microsoft SQL server databases. You create one wrapper for the Informix data source and one wrapper for the Microsoft SQL Server data source. Later your organization decides to migrate all of the information from Microsoft SQL Server to Informix. You no longer need the Microsoft SQL Server wrapper and you can drop it.

Important: There are serious consequences when you drop a wrapper. Other objects that you registered with the federated server are impacted:

- All server definitions that are dependent on the dropped wrapper are also dropped.
- All objects that are dependent on the dropped server definitions are also dropped.
- All nicknames that are dependent on the dropped server definitions are also dropped. Dropping the nicknames dependent on the server definition affects the objects dependent on those nicknames:
  - Any index specifications dependent on the dropped nicknames are also dropped.
  - Any views dependent on the dropped nicknames are marked inoperative.
  - Any materialized query tables dependent on the dropped nicknames are also dropped.
- All packages and cached dynamic SQL statements dependent on the dropped nicknames are marked invalid, and remain invalid until the dependent objects are re-created.

Procedure

To drop a wrapper, use the DROP statement.
Example: Drop the Microsoft SQL Server *MSSQLODBC3* wrapper:

```
DROP WRAPPER MSSQLODBC3
```

Dropping a server definition

Dropping a server definition deletes the definition from the global catalog. The data source object that the server definition references is not affected. You can drop a server definition by using the DROP statement from the DB2 command line processor.

Before you begin

You must have SYSADM or DBADM authority to drop a server definition.

About this task

When you no longer need to access a data source server, drop the server definition from the federated database. When you drop a server definition, other objects that you registered with the federated server are impacted:

- All user-defined function mappings, user-defined data type mappings, and user mappings that are dependent on the dropped server definition are also dropped.
- All nicknames that are dependent on the dropped server definition are also dropped. Dropping the nicknames dependent on the server definition, affects the objects dependent on those nicknames:
  - Any index specifications dependent on the dropped nicknames are also dropped.
  - Any views dependent on the dropped nicknames are marked inoperative.
  - Any materialized query tables dependent on the dropped nicknames are also dropped.
- All packages and cached dynamic SQL statements dependent on the dropped nicknames are marked invalid, and remain invalid until the dependent objects are re-created.

Restrictions

The federated server cannot process a DROP SERVER statement within a given unit of work (UOW) under either of the following conditions:

- The statement references a single data source, and the UOW already includes one of the following statements:
  - A SELECT statement that references a nickname for a table or view within the data source
  - An open cursor on a nickname for a table or view within the data source
  - An insert, delete or update issued against a nickname for a table or view within the data source
- The statement references a category of data sources (for example, all data sources of a specific type and version), and the UOW includes one of the following statements:
  - A SELECT statement that references a nickname for a table or view within one of the data sources
  - An open cursor on a nickname for a table or view within one of the data sources
  - An insert, delete or update issued against a nickname for a table or view within one of the data sources

Procedure

To delete a server definition, issue the DROP statement:
To delete a server definition, issue the DROP statement:

```
DROP SERVER server_name
```

where *server_name* identifies the server definition to be dropped.
Example: An Informix server uses the server name INFMX01. The following DROP statement drops the server definition:

```
DROP SERVER INFMX01
```

Dropping a user mapping

When a user no longer needs access to a remote data source, drop the user mapping between the federated server and the remote data source server. If the user is mapped to more than one data source server, you will need to drop each mapping separately.

Before you begin

To issue the DROP USER MAPPING statement, the authorization ID of the DROP statement must have SYSADM or DBADM authority, if this authorization ID is different from the federated database user ID specified in the user mapping. Otherwise, if the authorization ID and the user ID in the user mapping match, no authorities or privileges are required.

Procedure

To drop a user mapping, issue the DROP statement:

```
DROP USER MAPPING FOR user_ID SERVER local_server_name
```

where:

- *user_ID* is the authorization ID for the user on the federated server
- *local_server_name* is the local name that is used to identify the remote data source server in the server definition.

Dropping a nickname

Dropping a nickname deletes the nickname from the global catalog on the federated server. The data source object that the nickname references is not affected.

Before you begin

The nickname must be listed in the catalog.

The privileges that must be held by the authorization ID of the DROP statement when dropping nicknames must include one of the following:

- SYSADM or DBADM authority
- DROPIN privilege on the schema for the nickname
- Definer of the nickname as recorded in the DEFINER column of the catalog view for the nickname
- CONTROL privilege on the nickname

About this task

When you drop a nickname, other objects that you registered with the federated server are impacted:

- Dropping a nickname affects the objects dependent on those nicknames:
  - Any index specifications dependent on the dropped nicknames are also dropped.
  - Any views dependent on the dropped nicknames are marked inoperative.
  - Any materialized query tables dependent on the dropped nicknames are also dropped.
- All packages and cached dynamic SQL statements dependent on the dropped nickname are marked invalid, and remain invalid until the dependent objects are re-created.

Restrictions

For nicknames to relational data sources, the federated server cannot process the DROP NICKNAME statement within a given unit of work (UOW) under either of the following conditions:

- A nickname referenced in the statement has a cursor open on it in the same UOW.
- A nickname referenced in this statement is already referenced by a SELECT statement in the same UOW.
- An insert, delete or update is issued in the same UOW for the nickname referenced in the statement.

For nicknames to nonrelational data sources, the federated server cannot process the DROP NICKNAME statement within a given unit of work (UOW) under any of the following conditions:

- A nickname referenced in this statement has a cursor open on it in the same UOW.
- A nickname referenced in this statement is already referenced by a SELECT statement in the same UOW.

Procedure

To delete a nickname, issue the DROP statement:

```
DROP NICKNAME nickname
```

where *nickname* identifies the nickname to be dropped.

Chapter 3. Data type mappings in a federated system

The data types at a data source must map to corresponding DB2 data types. This mapping enables the federated server to retrieve data from the data source.

The federated database supplies a set of default data type mappings for some data sources. For other data sources you must provide the data type mappings that you want to use. For nonrelational data sources, you cannot override the existing data type mappings or create new mappings.

Some examples of default data type mappings are:

- The Oracle type FLOAT maps by default to the DB2 type DOUBLE
- The Oracle type DATE maps by default to the DB2 type TIMESTAMP
- The DB2 for z/OS type DATE maps by default to the DB2 type DATE

Nicknames that are created after a mapping is changed use the new type mapping. Nicknames that are created before the mapping is changed use the default data type mapping.

If you already created the nicknames, you can update the existing nicknames in one of the following ways:

- You can alter each nickname
- You can drop and re-create each nickname

DB2 federated servers do not support mappings for the following data types:

- The local data type cannot be LONG VARCHAR, LONG VARGRAPHIC, or a user-defined data type.
- The remote data type cannot be a user-defined type.

  However, you can use a cast function to convert the user-defined data type in a view at the remote data source to a built-in or system data type. You can then create a nickname for the view. For most data sources, if you create such views, the views have no statistics or indexes and you cannot update the views.

Data type mappings and the federated database global catalog

Local data type definitions are stored in the SYSCAT.COLUMNS catalog view of the federated database global catalog.

When you write a CREATE NICKNAME statement, you specify a data source object that the nickname represents. In most cases, the federated server defines a DB2-supported data type for each column or field in that data source object. For some nonrelational data sources, you must supply the DB2 data type.

For relational data sources, to determine which local data type to store in the SYSCAT.COLUMNS catalog view, the federated server looks for forward data type mapping information in the wrappers and in the SYSCAT.TYPEMAPPINGS catalog view. Mappings in the SYSCAT.TYPEMAPPINGS catalog view take precedence over the default mappings in the wrappers. If you create alternative mappings to override the default data type mappings, the federated server uses the alternative mappings. If multiple mappings apply to a column, the federated server uses the most recently created mappings.

For nonrelational data sources, to determine which local data type to store in the SYSCAT.COLUMNS catalog view, the federated server looks for data type mapping information in the wrappers. Depending on the nonrelational data source, the degree to which you can modify the data types defined by the wrapper varies. For some nonrelational data sources, you do not specify any columns. The wrapper defines the data types. For other data sources you can override the data types. And for other data sources, you must specify the column data types on the CREATE NICKNAME statement.

When you write CREATE TABLE transparent DDL for relational data sources, specify DB2 data types in the statement. The federated server checks for information about the reverse data type mappings between the federated database and the data source. The federated server looks for this information in the wrapper and the SYSCAT.TYPEMAPPINGS catalog view.

When values from a data source column are returned to the federated database, the values conform fully to the DB2 data type that the data source column is mapped to. If this mapping is a default mapping, the values also conform fully to the data source type in the mapping. For example, if an Oracle table with a FLOAT column is defined to the federated database, the default mapping of Oracle FLOAT to DB2 DOUBLE automatically applies to that column. The values that are returned from the column conform fully to both the FLOAT and DOUBLE data types.

When to create alternative data type mappings

You can create alternative data type mappings for relational data sources.

You might want to create alternative data type mappings in the following situations:

- To override a default data type mapping

  For some wrappers, you can change the format or length of values that are returned. You can change the format or length by changing the DB2 data type that the values must conform to. For example, the Oracle DATE data type is used as a timestamp and contains the century, year, month, day, hour, minute, and second. By default, the Oracle DATE data type maps to the DB2 TIMESTAMP data type. To return only the hour, minute, and second information, you can override the default data type mapping so that the Oracle DATE data type maps to the DB2 TIME data type. When the Oracle DATE columns are queried, only the time portion of the Oracle timestamp values is returned to the federated server.
- When a default mapping does not exist

  If a default data type mapping is not available for a data source data type, you must create a mapping for the new data type.

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

Data type mappings for nonrelational data sources

For some nonrelational data sources, the data type mappings are not in the wrappers. In some cases, you must specify the local type information in the CREATE NICKNAME statement.

The following example shows how column data types are specified in the CREATE NICKNAME statement for some of the nonrelational data sources:

```
CREATE NICKNAME DRUGDATA1
   (Dcode Integer NOT NULL, Drug CHAR(20), Manufacturer CHAR(20))
   FOR SERVER biochem_lab
   OPTIONS (FILE_PATH '/usr/pat/DRUGDATA1.TXT', COLUMN_DELIMITER ',',
   SORTED 'Y', KEY_COLUMN 'DCODE', VALIDATE_DATA_FILE 'Y')
```

Forward and reverse data type mappings

Forward type mappings and reverse type mappings are the two kinds of mappings between data source data types and federated database data types.

A *forward type mapping* is a mapping from a remote data type to a comparable local data type. Forward type mappings are used when you a create a nickname for a data source object. The comparable local type for each column in the data source object is stored in the global catalog.

A *reverse type mapping* is a mapping from a local data type to a comparable remote data type. Reverse type mapping is used with transparent DDL.

Figure 2 shows forward and reverse data type mapping.

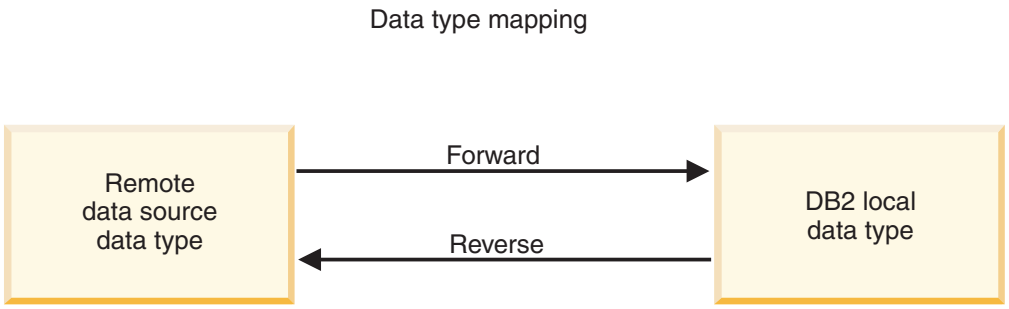


Figure 2. Forward and reverse data type mappings

Creating data type mappings

Use the CREATE TYPE MAPPING statement to create a data type mapping. You can run the statement from the command line processor, or include it in an application program.

Before you begin

The privileges held by the authorization ID associated with the statement must have SYSADM or DBADM authority.

About this task

See the CREATE TYPE MAPPING statement for specific use information.

Restrictions

- The *local_data_type* value cannot be LONG VARCHAR, LONG VARGRAPHIC, or a user-defined data type.
- The *data_source_data_type* value cannot be a user-defined type.

- For nonrelational data sources, the degree to which you can override existing data type mappings or create mappings is limited.

Procedure

Run the CREATE TYPE MAPPING statement to create a data type mapping.
To specify a server type in the CREATE TYPE MAPPING statement, you must use one of the following as the *server-type* value:

Table 3. Valid server types

Data source	Server type
DB2 Database for Linux, UNIX, and Windows	DB2/CS DB2/UDB DB2/NT DB2/SUN DB2/HP DB2/HPUX DB2/AIX DB2/6000 DB2/PE DB2/PTX DB2/SCO DB2/LINUX DB2/EEE DB2/2
DB2 for z/OS	DB2/MVS DB2/ZOS DB2/390
DB2 Server for VSE and VM	DB2/VM DB2/VSE SQL/DS
DB2 for System i	DB2/400 DB2/ISERIES
Oracle	ORACLE
Informix	INFORMIX
ODBC	ODBC
Microsoft SQL Server	MSSQLSERVER
JDBC	JDBC
Teradata	TERADATA
Sybase	SYBASE

Creating a type mapping for a data source data type - example

In this example, all Oracle tables and views that use the Oracle NUMBER data type must map to the DB2 DECIMAL(8,2) data type. The Oracle NUMBER data type is mapped by default to the DB2 DOUBLE data type, a floating decimal data type.

Use the ALTER NICKNAME statement to change the local types of existing nicknames. You must modify each nickname separately to change the local data type to DECIMAL(8,2). If the nicknames do not exist, create a data type mapping that specifies the data source type. Ensure that the data source wrapper is created before running the CREATE TYPE MAPPING statement. To create the type

mapping from Oracle NUMBER data type to the DB2 DECIMAL(8,2) data type, run the CREATE TYPE MAPPING statement, for example:

```
CREATE TYPE MAPPING MY_ORACLE_DEC FROM SYSIBM.DECIMAL(8,2)
    TO SERVER TYPE ORACLE TYPE NUMBER
```

MY_ORACLE_DEC
The name that you give to the type mapping. The name cannot duplicate a data type mapping name that already exists in the catalog.

FROM *SYSIBM.DECIMAL(8,2)*
The local DB2 schema and the local data type. If the length or precision and scale are not specified, then these values are determined from the source data type.

TO SERVER TYPE *ORACLE*
The type of data source.

TYPE *NUMBER*
The data source data type that you are mapping to the local data type. User-defined data types are not allowed.

The DB2 DECIMAL(8,2) data type is defined locally for the Oracle columns. When you create nicknames on Oracle tables and views that contain NUMBER columns, the Oracle NUMBER data type maps to the DB2 DECIMAL(8,2) data type.

Creating a type mapping for a data source data type and version - example

In this example, Oracle tables and views exist on different versions of the Oracle server. For all tables and views on Oracle Version 8.0.3 servers, columns that use the Oracle NUMBER(23,3) data type must map to the DB2 DECIMAL(8,2) data type.

The Oracle NUMBER(23,3) data type is mapped by default to the DB2 DECIMAL(23,3) data type. Use the ALTER NICKNAME statement to change the local types of existing nicknames. You must modify each nickname separately to change the local data type to DECIMAL(8,2). If the nicknames do not exist, create a data type mapping that specifies the data source type. Ensure that the data source wrapper is created before running the CREATE TYPE MAPPING statement. To map the Oracle NUMBER(23,3) data type to the DB2 DECIMAL(8,2) data type for Oracle servers using Version 8.0.3, run the CREATE TYPE MAPPING statement, for example:

```
CREATE TYPE MAPPING ORA_DEC FROM SYSIBM.DECIMAL(8,2)
    TO SERVER TYPE ORACLE VERSION 8.0.3 TYPE NUMBER(23,3)
```

ORA_DEC
The name that you give to the type mapping. The name cannot duplicate a data type mapping name that already exists in the catalog.

FROM *SYSIBM.DECIMAL(8,2)*
The local DB2 schema and the local data type. If the length or precision and scale are not specified, then these values are determined from the source data type.

TO SERVER TYPE *ORACLE*
The type of data source.

VERSION *8.0.3*
The version of data source server. You must specify the version. You can also specify the release and the modification of the release, as shown in this example.

TYPE *NUMBER(23,3)*
The data source data type that you are mapping to the local data type. User-defined data types are not allowed.

The federated database defines the DB2 DECIMAL(8,2) data type locally for the Oracle columns on Version 8.0.3 servers. Oracle tables and views on servers that do not use Version 8.0.3 instead use the default data type mapping. When you create nicknames on Oracle tables and views that contain NUMBER columns, the Oracle NUMBER data type maps to the DB2 DECIMAL(8,2) data type.

Creating a type mapping for all data source objects on a server - example

In this example, the server is defined to the federated database as ORA2SERVER. Each table contains a column with an Oracle DATE data type.

The Oracle DATE data type contains the century, year, month, day, hour, minute, and second. The Oracle DATE data type is mapped by default to the local DB2 TIMESTAMP data type. However, when you query any object on this server, the result set must return only the time information (hour, minute, and second).

Use the ALTER NICKNAME statement to change the local types of existing nicknames. You must modify each nickname separately to change the local data type to TIME.

If the nicknames do not exist, create a data type mapping that specifies the data source type.

To map the Oracle DATE data type to the DB2 TIME data type for the ORA2SERVER, issue the following statement:

```
CREATE TYPE MAPPING ORA2_DATE FROM SYSIBM.TIME
   TO SERVER ORA2SERVER TYPE DATE
```

ORA2_DATE
The name that you give to the type mapping. The name cannot duplicate a data type mapping name that already exists in the catalog.

FROM *SYSIBM.TIME*
The local DB2 schema and the local data type. If the length or precision and scale are not specified, then these values are determined from the source data type.

TO SERVER *ORA2SERVER*
The local name of the data source server.

TYPE *DATE*
The data source data type that you are mapping to the local data type. User-defined data types are not allowed.

The federated database locally defines the DB2 TIME data type for the Oracle columns of data type DATE.

When you create nicknames on Oracle tables and views that contain DATE columns, the Oracle DATE data type maps to the DB2 DECIMAL(8,2) data type.

Data source objects on other Oracle servers are not affected by this data type mapping.

Casting between data types

You can cast a data type from the source data type to the target data type.

Casting between data types can occur implicitly or explicitly.

- Implicit casting is the automatic conversion of data of one data type to data of another data type based on an implied set of conversion rules. This automatic conversion occurs in support of weak typing.
- Explicit casting supports strong typing. Strong typing requires matching data types. You must explicitly convert one or both data types to a common data type before performing comparisons or assignments.

Federation support for casting between data types enables federated queries on nicknames to access to servers that support both weak typing and strong typing.

Cast functions cannot be pushed down to remote servers if a counterpart function does not exist at the remote server. Overuse of casting can lead to performance problems.

Examples: Explicit casting

```
UPDATE nickname SET varcharcol = CAST(intcol AS varchar(10))

SELECT REAL(varchar_col) FROM nickname1;

SELECT VARCHAR(double_col) FROM nickname1;
```

Examples: Implicit casting

Example 1:

```
UPDATE nickname SET varcharcol = intcol;
```

In this assignment operation, the statement pushed down to the remote server is equivalent to the following statement:

```
UPDATE nickname SET varcharcol = varchar(intcol);
```

Example 2:

```
INSERT INTO nickname (varcharcol) SELECT intcol FROM nickname1;
```

In this assignment operation, the statement pushed down to the remote server is equivalent to the following statement:

```
INSERT INTO nickname (varcharcol) SELECT varchar(intcol) FROM nickname1;
```

Example 3:

```
SELECT * SELECT nickname SELECT intcol = varcharcol;
```

In this comparison operation, the statement pushed down to the remote server is equivalent to the following statement:

```
SELECT * SELECT nickname SELECT intcol = CAST(varcharcol AS decfloat)
```

TIMESTAMP data type support

The TIMESTAMP data type is parameterized to control the precision of fractional seconds.

For the DB2 for Linux, UNIX, and Windows data source, remote timestamps map to TIMESTAMP(*p*), where *p* represents the precision and specifies the number of fractional seconds. The range of *p* is 0 to 12, inclusive.

For other data sources, remote timestamps map to TIMESTAMP with a default precision of 6. For these data sources, you can take advantage of TIMESTAMP(*p*) support by using mappings similar to the samples provided for default forward type mappings.

When a nickname TIMESTAMP column has a smaller precision than its corresponding remote table column, and the remote table column contains non-zero digits in the extra fractional digits, the result of predicates using that nickname column can be unpredictable. The result of predicates using that nickname column works correctly if the predicate is not pushed down.

Altering a local type for a data source object

You use the ALTER NICKNAME statement instead of the CREATE TYPE MAPPING statement to change a local data type.

Before you begin

The authorization ID issuing the statement must include at least one of the following privileges:

- SYSADM or DBADM authority
- ALTER privilege on the nickname specified in the statement
- CONTROL privilege on the nickname specified in the statement
- ALTERIN privilege on the schema, if the schema name of the nickname exists
- Definer of the nickname as recorded in the DEFINER column of the catalog view for the nickname.

About this task

When you create a nickname, the data types that are associated with the data source object are stored in the federated database. For some data sources, the wrapper specifies the data types for you. For other data sources, you must specify the data types when you create the nickname.

You can specify a local type for a column of a specific data source object.

Important: Changing the local data type can result in errors or loss of information if you change the local data type for a column to a type that differs greatly from its remote type.

Restrictions

See "Restrictions on altering nicknames" on page 25.

Procedure

To change the local data type from command line prompt, issue the ALTER NICKNAME statement from the command line.
For example:

```
ALTER NICKNAME nickname ALTER COLUMN column_name
   LOCAL TYPE data_type
```

To treat the contents of a local column that has a character data type as bit (binary) data, use the FOR BIT DATA clause in the ALTER NICKNAME statement. When you use this clause to change the local data type of a column, code page conversions are not performed when data is exchanged with other systems. Comparisons are done in binary, irrespective of the remote database collating sequence.

Altering a local type for a data source object - examples

This topic provides examples that show how to change the data types for a data source object.

Example: A numeric data type mapping

In an Oracle table for employee information, the BONUS column is defined with a data type of NUMBER(32,3). The Oracle data type NUMBER(32,3) is mapped by default to the DB2 data type DOUBLE, a double-precision floating-point number data type. A query that includes the BONUS column might return values that look like this:

```
5.0000000000000E+002
1.0000000000000E+003
```

The scientific notation indicates the number of decimal places and the direction that the decimal point should be moved. In this example +002 signifies that the decimal point should be moved two places to the right, and +003 signifies that the decimal point should be moved three places to the right.

Queries that include the BONUS column can return values that look like dollar amounts. You change the local definition for the BONUS column in the table from the DOUBLE data type to DECIMAL data type. Use a precision and scale that reflect the format of actual bonuses. For example, if the dollar portion of the bonuses would not exceed six figures, map NUMBER(32,3) to DECIMAL(8,2). Under the constraint of this new local type, queries that include the BONUS column return values like this:

```
 500.00
1000.00
```

The nickname for the Oracle table is ORASALES. To map the BONUS column in the ORASALES table to the DB2 DECIMAL (8,2) data type, issue the following ALTER NICKNAME statement:

```
ALTER NICKNAME ORASALES ALTER COLUMN BONUS
   LOCAL TYPE DECIMAL(8,2)
```

ORASALES
: The nickname that you defined for the Oracle table.

ALTER COLUMN *BONUS*
: The name of the column that is defined locally in the federated database SYSCAT.COLUMNS catalog view.

LOCAL TYPE *DECIMAL(8,2)*
Identifies the new local type for the column.

This mapping applies only to the BONUS column in the Oracle table that is identified by the nickname ORASALES. All other Oracle data source objects that include the BONUS column use the default data type mapping for the Oracle NUMBER data type.

Example: A date data type mapping

The nickname for an Oracle table named SALES is ORASALES. The SALES table contains a column that is the Oracle DATE data type. The default type mapping for the Oracle DATE data type is to the DB2 TIMESTAMP data type. However, you want to display only the date value when you retrieve data from this column. You can alter the nickname for the SALES table to change the local type to the DB2 DATE data type.

```
ALTER NICKNAME ORASALES ALTER COLUMN ORDER_DATE
   LOCAL TYPE DATE
```

Example: A data type mapping for a nonrelational data source

The nickname for a table-structured file named `drugdata1.txt` is DRUGDATA1. The `drugdata1.txt` file contains a column that lists pharmaceutical drug names. The column name is DRUG. The DRUG column was originally defined as a CHAR(20). The length of the column must be changed to CHAR(30). You can alter the nickname for the `drugdata1.txt` file to change the mapping to the correct length:

```
ALTER NICKNAME DRUGDATA1 ALTER COLUMN DRUG
   LOCAL TYPE CHAR(30)
```

Altering LONG data types to VARCHAR data types

To enable insert and update operations for LONG data types, you can alter the LONG data types to the VARCHAR data type.

Table 4 lists the long data type by data source that you can alter.

Table 4. LONG data types by data source that can be altered to the VARCHAR data type

Data source	Remote data type	Length	Local default data type	ALTER to VARCHAR
DRDA	LONG VARCHAR	1–32672	CLOB	VARCHAR
	LONG VARCHAR FOR BIT DATA	1–32672	BLOB	VARCHAR FOR BIT DATA
Informix	BYTE	1–32672	BLOB	VARCHAR FOR BIT DATA
	TEXT	1–32672	CLOB	VARCHAR
Microsoft SQL Server	IMAGE	1–32672 host vars; 1–8000 literal	BLOB	VARCHAR FOR BIT DATA
	TEXT	1–32672 host vars; 1–8000 literal	CLOB	VARCHAR

Table 4. LONG data types by data source that can be altered to the VARCHAR data type (continued)

Data source	Remote data type	Length	Local default data type	ALTER to VARCHAR
Oracle NET8	LONG	1–32672 host vars; 1–4000 literal	CLOB	VARCHAR
	LONG RAW	1–32672 host vars; 1–4000 literal	BLOB	VARCHAR FOR BIT DATA
Sybase CTLIB	IMAGE	1–32672	BLOB	VARCHAR FOR BIT DATA
	TEXT	1–32672	CLOB	VARCHAR
Teradata	BYTE	32673–64000	BLOB	VARCHAR FOR BIT DATA(32672)
	CHAR	32673–64000	CLOB	VARCHAR(32672)
	VARBYTE	32673–64000	BLOB	VARCHAR FOR BIT DATA(32672)
	VARCHAR	32673–64000	CLOB	VARCHAR(32672)

Chapter 4. Mapping functions and user-defined functions

Function mappings associate federated server functions and user-defined functions with the existing functions at the data source.

Function mappings in a federated system

IBM InfoSphere Federation Server supplies default mappings between existing data source functions and DB2 counterpart functions.

For the federated server to use a data source function, a mapping is needed from a DB2 function or a function template to the data source function.

The default function mappings are in the wrapper modules.

For nonrelational data sources, you cannot override the existing function mappings or create new mappings.

How function mappings work in a federated system

When you submit queries to the federated server that contain one or more functions, the federated server checks for information about the mappings between the DB2 functions and the data source functions.

The federated server checks two places for mapping information:

- The wrapper. The data source wrapper contains the default function mappings.
- The SYSCAT.FUNCMAPPINGS catalog view. This view contains entries you create that override or augment the default function mappings that are in the wrapper. It also contains new mappings that you create when there is no default function mapping. When multiple mappings can be applied to a function, the most recently created one is applied.

Function mapping options specify information about the function and the potential cost of processing a function at the data source. Function mapping options provide information such as:

- Name of the remote data source function
- The estimated number of instructions processed the first and last time that the data source function is invoked.
- Estimated number of I/Os performed the first and last time that the data source function is invoked.
- Estimated number of instructions processed per invocation of the data source function.

When you create a function mapping, you are mapping from a DB2 function or function template to a counterpart function at the data source. When a DB2 counterpart function does not exist, or when you want to force the federated server to use the data source function, you can create a function template to act as the counterpart.

Why function mappings are important

Function mappings are one of several important inputs to the pushdown analysis performed by the query optimizer.

In deciding on the best query access plan, the query optimizer factors the capabilities of the data source to perform a particular type of SQL function or operation. If the function does not have a mapping, the function will not be sent to the data source for processing. Functions and other operations that can be pushed down to the data source improve performance.

If the data source has a similar function to a DB2 function, but it returns slightly different results, creating a function mapping might improve performance. For example, the Informix STDEV (standard deviation) function produces different results than the DB2 STDDEV function for some sets of input data. For this reason the Informix wrapper does not have a default mapping between these two functions. If you do not care about the result set differences, you might improve the performance of queries that access Informix data sources and use the DB2 STDDEV function. By creating a function mapping between the Informix STDEV and the DB2 STDDEV function, you provide the query optimizer the choice of sending the processing of that function down to the data source.

When to create function mappings

When a default function mapping is not available for a data source function, you can create a function mapping.

One reason that a function mapping is not available is that the federated database does not have a function that corresponds to the data source function.

Another reason a mapping is not available is that the data source has a similar function to a DB2 function, but it does not return the same results. If the data source returns slightly different results or different results for certain sets of input data, the wrappers do not normally map to these functions. However, if you do not care about the differences in the result sets, then you can create a mapping between the functions. Creating a mapping might improve performance.

Use function mappings when:

- A new built-in function becomes available at the data source
- A new user-defined function becomes available at the data source
- A DB2 counterpart function does not exist
- A counterpart function exists but returns slightly different results, which you do not care about

The settings for function mappings are stored in the SYSCAT.FUNCMAPPINGS catalog view.

When you create a function mapping, it is possible that the return values from a function evaluated at the data source will be different than the return values from a compatible function evaluated at the federated database. IBM InfoSphere Federation Server uses the function mapping, but it might result in an SQL syntax error or unexpected results.

User-defined functions in applications

Application developers often need to create their own suite of functions specific to their application or domain. They can use user-defined scalar functions for this purpose.

For example, a retail store could define a PRICE data type for tracking the cost of items that it sells. This store might also want to define a SALES_TAX function. This function would take a given price value as input, compute the applicable sales tax, and return this data to the requesting user or application.

These functions can operate over all database types, including large object types and distinct types. User-defined functions allow queries to contain powerful computation and search predicates to filter irrelevant data close to the source of the data, thereby reducing response time. The SQL optimizer treats user-defined functions exactly like built-in functions such as SUBSTR and LENGTH. You can develop applications using different application language environments, such as C, C++, and COBOL. The applications can share a set of SQL user-defined functions even though they are developed using different application language environments.

User-defined functions can manipulate data and perform actions. For example, you might enable a user-defined function to send an electronic message or to update a flat file.

In DB2, user-defined functions can include:

- Functions that you define from scratch.
- Functions in the SYSFUN schema. Examples include mathematical functions such as SIN, COS, and TAN; scientific functions such as RADIANS, LOG10, and POWER; and general purpose functions such as LEFT, DIFFERENCE, and UCASE.

Requirements for mapping user-defined functions

Before you can invoke a data source user-defined function in a federated system, the federated database must associate the data source function with a function specification stored in the global catalog on the federated server.

There are two conditions under which the federated database can associate a function specification with a data source function:

- The federated database has a function whose signature corresponds to that of the signature of the data source function. A *signature* consists of a function name and function input parameters. Signatures *correspond* when both the following conditions are true:
  - They contain the same names and the same number of parameters
  - The data type of each parameter in one signature is the same as (or can be converted to) the data type of the corresponding parameter in the other signature.
- If the federated database does not have a function with the requisite signature, you can define a function template that contains this signature. You then map the function template to the data source function that you want to invoke.

The settings for function mappings are stored in the SYSCAT.FUNCMAPPINGS catalog view.

Creating function mappings

Use the CREATE FUNCTION MAPPING statement to specify alternative function mappings that override the default function mappings.

When you create alternative function mappings, the entries appear in the SYSCAT.FUNCMAPPINGS catalog view.

You can also use the CREATE FUNCTION MAPPING statement to specify function mapping options. When you specify function mapping options, the information appears in the SYSCAT.FUNCMAPOPTIONS catalog view.

With the CREATE FUNCTION MAPPING statement, you can:

- Create a function mapping for all data sources of a specific type. For example, all Informix data sources.
- Create a function mapping for all data sources of a specific type and version. For example, all Informix 9 data sources.
- Create a function mapping for a specific server.
- Provide function mapping statistical information to the optimizer
- Disable a default function mapping or a function mapping that you defined.

You can issue the CREATE FUNCTION MAPPING statement in the command line processor. You can also embed the CREATE FUNCTION MAPPING statement in an application program.

Specifying function names in the CREATE FUNCTION MAPPING statement

The values that you enter in the CREATE FUNCTION MAPPING statement depend on whether the functions that you are mapping together have the same name or different names.

The privileges held by the authorization ID of the statement must have SYSADM or DBADM authority.

Mapping functions with the same name

You can create a mapping between two functions (or a DB2 function template and a data source function) that have the same name.

Procedure

To map two functions with the same name, issue the CREATE FUNCTION MAPPING statement.

Example: You want to map a user-defined function named MYFUN at an Informix data source to the DB2 user-defined function named TINA.MYFUN. The Informix data source server is called INFORMIX2. The following statement maps the function:

```
CREATE FUNCTION MAPPING FOR TINA.MYFUN(SYSTEM.INTEGER) SERVER INFORMIX2
```

Mapping functions with different names

You can create a mapping between two functions (or a DB2 function template and a data source function) that have different names.

About this task

To create a mapping between two functions with different names, issue the CREATE FUNCTION MAPPING statement:

Procedure

1. Assign the name of the DB2 function or function template to the function_name parameter.
2. Specify a function mapping option called REMOTE_NAME and assign the name of the data source function to this option. The REMOTE_NAME must be less than 255 characters.

Example

Example: You want to map a user-defined function named UPPERCASE at an Oracle data source to the DB2 function UCASE(CHAR). The Oracle data source server is called ORACLE2. You decide to name this function mapping ORACLE_UPPER. The syntax would be:

```
CREATE FUNCTION MAPPING ORACLE_UPPER FOR SYSFUN.UCASE(CHAR)
    SERVER ORACLE2 OPTIONS
    (REMOTE_NAME 'UPPERCASE')
```

Creating a function mapping for a specific data source type

You can create a mapping to a function for all data sources of a specific type.

Before you begin

The privileges held by the authorization ID of the statement must have SYSADM or DBADM authority.

About this task

Restrictions

You cannot override the existing function mappings or create new mappings for nonrelational data sources.

Procedure

To map a DB2 function template to a data source function, use the CREATE FUNCTION MAPPING statement.

Example: Map a DB2 function template to an Oracle user-defined function for all Oracle data sources

```
CREATE FUNCTION MAPPING MY_ORACLE_FUN1
     FOR NOVA.STATS ( DOUBLE, DOUBLE )
     SERVER TYPE ORACLE
     OPTIONS (REMOTE_NAME 'STAR.STATISTICS')
```

The template is called STATS and belongs to a schema called NOVA. The Oracle user-defined function is called STATISTICS and belongs to a schema called STAR.

Creating a function mapping for a specific data source type and version

You can create a mapping to a function for all data sources that use a specific version of the data source type.

Before you begin

The privileges held by the authorization ID of the statement must have SYSADM or DBADM authority.

About this task

Restrictions

You cannot override the existing function mappings or create new mappings for nonrelational data sources.

Procedure

To create a mapping for a specific data source type and version, use the CREATE FUNCTION MAPPING statement.

Example: Map a DB2 function template to a Sybase user-defined function for all Sybase data sources that use Version 15

The template is called SYB_STATS and belongs to a schema called EARTH. The Sybase user-defined function is called STATISTICS and belongs to a schema called MOON. The CREATE FUNCTION MAPPING is:

```
CREATE FUNCTION MAPPING SYBASE_STATS
   FOR EARTH.SYB_STATS ( DOUBLE, DOUBLE )
   SERVER TYPE SYBASE VERSION 15
   OPTIONS (REMOTE_NAME 'MOON.STATISTICS')
```

Creating a function mapping for all data source objects on a specific server

You can create a mapping to a function that is used by all data sources objects on a specific remote server.

Before you begin

The privileges held by the authorization ID of the statement must have SYSADM or DBADM authority.

About this task

Restrictions

You cannot override the existing function mappings or create new mappings for nonrelational data sources.

Procedure

To create a function mapping for all data source objects on a specific server, use the CREATE FUNCTION MAPPING statement.

Example: Map a function template called BONUS to a user-defined function called BONUS

You only want the mapping to apply to an Oracle data source server called ORA_SALES. Because the function names are the same, you do not need to specify the REMOTE_NAME function mapping option.

```
CREATE FUNCTION MAPPING BONUS_CALC FOR BONUS()
 SERVER ORA_SALES
```

Function templates

The federated server recognizes a data source function when there is a mapping between the data source function and a DB2 counterpart function at the federated database.

You can create a function template to act as the DB2 counterpart function when no counterpart exists.

A *function template* is a DB2 function that you create for the purpose of forcing the federated server to invoke a data source function. However, unlike a regular function, a function template has no executable code. When the federated server receives queries that specify the function template, the federated server will invoke the data source function.

The function template is created with the CREATE FUNCTION statement using the AS TEMPLATE parameter.

After you create a function template, you must then create the function mapping between the template and the data source function. A function mapping is created using the CREATE FUNCTION MAPPING statement.

Creating function templates

The federated server recognizes a data source function when there is a mapping between the data source function and a counterpart function at the federated database. You can create a function template to act as the counterpart when no counterpart exists.

Before you begin

The privileges held by the authorization ID of the statement must include at least one of the following:

- SYSADM or DBADM authority
- IMPICIT_SCHEMA authority on the databases, if the implicit or explicit schema name of the function does not exist
- CREATEIN privilege on the schema, if the schema name of the function exists

About this task

Restrictions

If the data source function has input parameters:

- The DB2 counterpart function must have the same number of input parameters that the data source function has.

- The data types of the input parameters for the DB2 counterpart function must be compatible with the corresponding data types of the input parameters for data source function. The data type cannot be LONG VARCHAR, LONG VARGRAPHIC, or a user-defined type.

If the data source function has no input parameters, the DB2 counterpart function cannot have any input parameters.

Procedure

1. Use the CREATE FUNCTION statement with the AS TEMPLATE parameter. For example:

```
CREATE FUNCTION BONUS ()
  RETURNS DECIMAL(8,2)
  AS TEMPLATE
  DETERMINISTIC
  NO EXTERNAL ACTION
```

   BONUS ()
   The name you give to the function template.

   RETURNS *DECIMAL(8,2)*
   The data type of the output.

   AS TEMPLATE
   Indicates that this is a function template, not a function.

   DETERMINISTIC
   Specifies that the function always returns the same results for a given set of argument values.

   NO EXTERNAL ACTION
   Specifies that the function has no external impact on objects that are not managed by the database manager.

   You must specify the DETERMINISTIC and NO EXTERNAL ACTION clauses according to whether the function itself is deterministic and whether it causes any external action. Otherwise, restrictions will be imposed on the SQL operations that are supported with this function template.

2. After you create a function template, you must then create the function mapping between the template and the data source function. A function mapping is created using the CREATE FUNCTION MAPPING statement. For example:

```
CREATE FUNCTION MAPPING MY_INFORMIX_FUN FOR BONUS()
  SERVER TYPE INFORMIX OPTIONS (REMOTE_NAME 'BONUS()')
```

   MY_INFORMIX_FUN
   The name you give to the function mapping. The name cannot duplicate a function mapping name that is already described in the federated database global catalog. It must be unique.

   FOR *BONUS()*
   The local DB2 function template name. Include data type input parameters in parenthesis.

   SERVER TYPE *INFORMIX*
   Identifies the type of data source which contains the function that you want to map to.

   OPTIONS *(REMOTE_NAME 'BONUS()')*
   An option that identifies the name of the remote data source function that you are mapping to the local DB2 function template.

Disabling a default function mapping

Default function mappings cannot be dropped. However, you can render them inoperable by disabling them.

Before you begin

The privileges held by the authorization ID of the statement must have SYSADM or DBADM authority.

Procedure

To disable a default function mapping, the CREATE FUNCTION MAPPING statement specifies the name of the DB2 function and sets the DISABLE option to 'Y'.

Example: Disable a default function mapping between the DB2 SIN function and a similar function on Oracle data sources

When a query that requests Oracle data and that references SIN is processed, either function might be invoked. The function invoked depends on which function is estimated by the query optimizer to require less overhead.

To ensure that the DB2 SIN function is invoked and that the Oracle SIN function is not invoked, you must disable the default function mapping. Use the following syntax:

```
CREATE FUNCTION MAPPING FOR SYSFUN.SIN(INT)
TYPE ORACLE OPTIONS (DISABLE 'Y')
```

Dropping a function mapping

When you no longer require a function mapping that you created, you can delete the function mapping by running the **DROP** statement.

Before you begin

The privileges held by the authorization ID of the statement must have SYSADM or DBADM authority.

About this task

If you drop a function mapping that you created to override a default function mapping, the default function mapping will be used.

The function mappings are listed in the SYSCAT.FUNCMAPPINGS catalog view.

Procedure

To drop a function mapping that you created, use the DROP statement:

Example: Drop a function mapping called BONUS_CALC

```
DROP FUNCTION MAPPING BONUS_CALC
```

Chapter 5. Creating index specifications

For relational data sources, create index specifications to store information about the columns of remote indexes in the global catalog and to ensure optimal performance of search queries.

Index specifications in a federated system

In a federated system, you use the CREATE INDEX statement with a nickname to store information in the global catalog about the availability of an index on the remote object. The query optimizer uses this information to optimize queries.

When you issue a CREATE INDEX statement

- If a nickname is created for a table, the CREATE INDEX statement collects index information about the index that was created on the remote table.
- If a nickname is created for a view, the CREATE INDEX statement references the nickname for the view and contains information about the index on the table underlying the view.

The index specification tells the federated server about the columns and their uniqueness properties that comprise a remote index. It does not tell the federated server about the statistical properties of the index, such as, the number of unique values of the index key.

You do not need to supply index specifications if the remote index was in place at the time that the nickname was created.

A federated server does not create an index specification when you create a nickname for:

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

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

Use index specifications with relational data sources. Creating an index specification for a nonrelational data source will not improve performance.

Creating index specifications for data source objects

When a nickname is created for a data source table, the federated server supplies the global catalog with information about any indexes that the data source table has. The optimizer uses this information to expedite the processing of distributed requests. This information is a set of metadata and is called an *index specification*.

Before you begin

The privileges held by the authorization ID of the statement must include at least one of the following:

- SYSADM or DBADM authority
- One of CONTROL privilege on the object or INDEX privilege on the object. And one of IMPLICIT_SCHEMA authority on the database, if the implicit or explicit schema name of the index does not exist, or CREATEIN privilege on the schema, if the schema name of the index refers to an existing schema.

About this task

The federated server does not create an index specification if:

- A nickname is created for a table that has no index.
- A nickname is created for a data source object that does not contain indexes such as a view or Informix synonym.
- A nickname is created for a nonrelational object, for example, a table-structured file, Excel spreadsheet, or XML tagged file.
- The remote index is on a LOB column or an XML column.
- The remote index contains a total key length greater than 1024 bytes.
- The maximum number of key parts is more than 16.

In these circumstances the federated server does not store index specifications for the data source objects. However, for the first two items in the previous list you can supply the necessary index information to the global catalog. You can use the CREATE INDEX statement to specify the index information.

Restrictions

There are some restrictions when creating an index specification on a nickname.

- If the bind option DYNAMICRULES BIND applies, the statement cannot be dynamically prepared. Also, you cannot use the INCLUDE, CLUSTER, PCTFREE, MINPCTUSED, DISALLOW REVERSE SCANS, and ALLOW REVERSE SCANS parameters in the CREATE INDEX statement.
- UNIQUE should be specified only if the data for the index key contains unique values for every row of the data source table. The uniqueness will not be checked.
- The sum of the stored lengths of the specified columns must not be greater than 1024.
- No LOB column or distinct type column based on a LOB can be used as part of an index. This restriction is enforced even if the length attribute of the column is small enough to fit within the 1024-byte limit.

Procedure

To create an index, you can embed the CREATE INDEX statement in an application program or issue the statement as a dynamic SQL statement from the command line.

When used with nicknames, the CREATE INDEX statement creates an index specification in the federated global catalog; it does not create an index on the data source table.

Use the following syntax to create an index specification:

```
CREATE INDEX index_name ON nickname
(column_name) SPECIFICATION ONLY
CREATE UNIQUE INDEX index_name ON nickname
(column_name DESC) SPECIFICATION ONLY
```

For an index specification, *column_name* is the name by which the federated server references a column of a data source table.

Creating index specifications on tables that acquire new indexes

For situations in which a table acquires a new index, you should create an index specification on the nickname that corresponds to the table.

Before you begin

The privileges held by the authorization ID of the statement must include at least one of the following:

- SYSADM or DBADM authority
- One of CONTROL privilege on the object or INDEX privilege on the object. And one of IMPLICIT_SCHEMA authority on the database, if the implicit or explicit schema name of the index does not exist, or CREATEIN privilege on the schema, if the schema name of the index refers to an existing schema.

About this task

There are several situations in which a table acquires a new index:

- You create a nickname for a table that does not have an index, but acquires an index later
- You create a nickname for a table that has an index, but acquires another index later

In these situations, you should create an index specification for the table so that the SQL Complier can use this information when processing queries that reference the table.

Restrictions

There are some restrictions when creating an index on a nickname.

- If the bind option DYNAMICRULES BIND applies, the statement cannot be dynamically prepared. Also, you cannot use the INCLUDE, CLUSTER, PCTFREE, MINPCTUSED, DISALLOW REVERSE SCANS, and ALLOW REVERSE SCANS parameters in the CREATE INDEX statement.
- UNIQUE should be specified only if the data for the index key contains unique values for every row of the data source table. The uniqueness will not be checked.

- The sum of the stored lengths of the specified columns must not be greater than 1024.
- No LOB column or distinct type column based on a LOB can be used as part of an index. This restriction is enforced even if the length attribute of the column is small enough to fit within the 1024-byte limit.

Procedure

The following examples describe how to create an index specification for a nickname that corresponds to a table that acquires an index.

Example: A table that has no index, later acquires an index

Suppose that you create the nickname *EMPLOYEE* for a data source table called CURRENT_EMP, which has no indexes. Sometime after this nickname is created, an index was defined on CURRENT_EMP using the WORKDEPT and JOB columns for the index key.

To create an index specification that describes this index, the syntax would be:

```
CREATE UNIQUE INDEX JOB_BY_DEPT ON EMPLOYEE
(WORKDEPT, JOB) SPECIFICATION ONLY
```

where *JOB_BY_DEPT* is the index name.

Example: A table acquires a new index

Suppose that you create the nickname *JP_SALES* for a table called JAPAN_SALES. A new index is later added to the table in addition to the ones it had when the nickname was created. The new index uses the MARKUP column for the index key.

To create an index specification that describes this index, the syntax would be:

```
CREATE UNIQUE INDEX JP_MARKUP ON JP_SALES (MARKUP) SPECIFICATION ONLY
```

where *JP_MARKUP* is the index name.

Creating index specifications on views

When a nickname is created for a view, the federated server is unaware of the underlying table (and its indexes) from which the view was generated. Create an index specification for the view so that the SQL Compiler can use this information when processing queries that reference the view.

Before you begin

The privileges held by the authorization ID of the statement must include at least one of the following:

- SYSADM or DBADM authority
- One of CONTROL privilege on the object or INDEX privilege on the object. And one of IMPLICIT_SCHEMA authority on the database, if the implicit or explicit schema name of the index does not exist, or CREATEIN privilege on the schema, if the schema name of the index refers to an existing schema.

About this task

Restrictions

There are some restrictions when creating an index on a nickname.

- If the bind option DYNAMICRULES BIND applies, the statement cannot be dynamically prepared. Also, you cannot use the INCLUDE, CLUSTER, PCTFREE, MINPCTUSED, DISALLOW REVERSE SCANS, and ALLOW REVERSE SCANS parameters in the CREATE INDEX statement.
- UNIQUE should be specified only if the data for the index key contains unique values for every row of the data source table. The uniqueness will not be checked.
- The sum of the stored lengths of the specified columns must not be greater than 1024.
- No LOB column or distinct type column based on a LOB can be used as part of an index. This restriction is enforced even if the length attribute of the column is small enough to fit within the 1024-byte limit.

Procedure

- Ensure that the column or columns that the table index is based on is part of the view.
- If you want to create index specifications for all indexes on the underlying table, each index specification must be created separately.

Example

Example: Create an index specification that describes the REGION index

Suppose that you create the nickname *JP_SALES2007* for a view called JAPAN_SALES2007. The underlying table for this view is the JAPAN_SALES table which contains several indexes: REGION, AMOUNT, SALES_REP. The CREATE INDEX statement you create will reference the nickname for the view and contain information about the index of the underlying table for the view.

```
CREATE UNIQUE INDEX JP_2007_REGION ON JP_SALES2007
(REGION) SPECIFICATION ONLY
```

where *JP_2007_REGION* is the index name, and *JP_SALES2007* is the nickname for the view JAPAN_SALES2007.

Creating index specifications on Informix synonyms

This topic describes the action that the federated server takes for Informix synonyms based on a table or on a view:

Before you begin

The privileges held by the authorization ID of the statement must include at least one of the following:

- SYSADM or DBADM authority
- One of CONTROL privilege on the object or INDEX privilege on the object. And one of IMPLICIT_SCHEMA authority on the database, if the implicit or explicit schema name of the index does not exist, or CREATEIN privilege on the schema, if the schema name of the index refers to an existing schema.

About this task

In Informix, you can create a synonym for a table or view. While the federated server allows you to create nicknames for Informix synonyms, the action that the federated server takes depends on whether the synonym is based on a table or a view:

- Suppose that a nickname is created for a synonym, and the synonym is based on an Informix table. If the federated server determines that the table the synonym refers to has an index, then an index specification is created for the synonym. If the table that the synonym refers to does not have an index, then no index specification is created for the synonym. However you can create an index specification manually, using the CREATE INDEX statement.
- Suppose that a nickname is created for a synonym, and the synonym is based on an Informix view. The federated server can not determine which underlying table or tables the view is based on. Therefore no index specification is created for the synonym. However you can create an index specification manually using the CREATE INDEX statement.

Restrictions

There are some restrictions when creating an index on a nickname.

- If the bind option DYNAMICRULES BIND applies, the statement cannot be dynamically prepared. Also, you cannot use the INCLUDE, CLUSTER, PCTFREE, MINPCTUSED, DISALLOW REVERSE SCANS, and ALLOW REVERSE SCANS parameters in the CREATE INDEX statement.
- UNIQUE should be specified only if the data for the index key contains unique values for every row of the data source table. The uniqueness will not be checked.
- The sum of the stored lengths of the specified columns must not be greater than 1024.
- No LOB column or distinct type column based on a LOB can be used as part of an index. This restriction is enforced even if the length attribute of the column is small enough to fit within the 1024-byte limit.

Procedure

The following examples describe how to create an index specification on a nickname that corresponds to an Informix synonym.

Example: A nickname is created on an Informix synonym that is based on a table

When the synonym is based on an Informix table that does not contain an index, you can create an index specification for the synonym to tell the optimizer which column or columns to search on to find data quickly. The statement you create will specify the nickname for the synonym, and you will supply information about the column or columns in the table that the synonym is based on.

In this example, you create the nickname *CONTRACTS* for a synonym called SALES_CONTRACTS. The table that this synonym is based on is called SALES2006_TABLE and contains several indexes: REGION, AMOUNT, SALES_REP. The CREATE INDEX statement you create will reference the nickname for the synonym and contain information about the index of the underlying table for the synonym.

To create an index specification that describes the REGION index, the syntax would be:

```
CREATE UNIQUE INDEX NORTHWEST_2006_REGION ON CONTRACTS (REGION) SPECIFICATION ONLY
```

where *NORTHWEST_2006_REGION* is the index name and *CONTRACTS* is the nickname for the synonym SALES_CONTRACTS.

Example: A nickname is created on an Informix synonym that is based on a view

You create the nickname *JP_SALES2007* for a synonym based on a view called JAPAN_SALES2007. The underlying table for this view is the JAPAN_SALES table which contains several indexes: REGION, AMOUNT, SALES_REP. The CREATE INDEX statement that you create will reference the nickname for the synonym and contain information about the index of the underlying table for the view.

When creating an index specification for a synonym based on a view, make certain that the column or columns the table index is based on, is part of the view. If you want to create index specifications for all indexes on the underlying table, each index specification must be created separately.

To create an index specification that describes the REGION index, the syntax would be:

```
CREATE UNIQUE INDEX JP_2007_REGION ON JP_SALES2007 (REGION) SPECIFICATION ONLY
```

where *JP_2007_REGION* is the index name and *JP_SALES2007* is the nickname for the view JAPAN_SALES2007.

Chapter 6. Developing federated procedures

Federated procedures enable you to invoke procedures at a data source as if the remote procedure is a local procedure.

Federated procedures

A *federated procedure* is a federated database object that references a procedure on a data source.

Federated procedures are not alternative names for data source procedures in the same way that aliases are alternative names. A federated procedure is defined at the federated database but calls a data source procedure when the federated procedure is invoked. Because the federated procedure is a federated database object, users and client applications can invoke the data source procedure logic by calling a federated procedure. The results of the data source procedure, such as the output parameters, are returned by the federated procedure. Using a federated procedure makes the location of the data source procedure transparent to users and client applications. You use the name of the federated procedure to call the data source procedure.

A federated procedure is to a remote procedure what a nickname is to a remote table. Nicknames and federated procedures are objects on the federated database. A nickname is an object that references an object, such as a table or view, on the data source. With a nickname, you query a data source object. With a federated procedure, you call a data source procedure.

You use the CREATE PROCEDURE (Sourced) statement to register a federated procedure and use the CALL statement to call a procedure. You can embed the CREATE PROCEDURE (Sourced) statement in an application program or issue the statement with dynamic SQL statements.

Parameters and data types

Federation uses three types of parameters: IN, OUT, and INOUT. When you create a federated procedure, the default forward data type mappings are used to map the data types for the parameters for the data source procedure to the federated data types. This mapping includes data types for the input and output parameters and data types for the columns in the result set. You can use the CREATE TYPE MAPPING statement to override the default type mapping for the data source parameters. However, the type mappings of the result set are not affected by user-defined type mappings. You can use all of the data types that are supported for nickname columns except LOB data types. Federated procedures do not support complex data types.

User mappings and authorization

To call a federated procedure, you must have the correct authorizations on the federated procedure and the data source procedure. When you call a federated procedure, the user mapping and privileges of the authorization ID that created the federated procedure are used to access the data source tables.

For example, the user ZELLER creates a federated procedure named FP1. The FP1 procedure references a Sybase procedure that accesses a Sybase table. The remote user ID in the user mapping for ZELLER has the privilege to update the Sybase table. The user ZELLER grants the EXECUTE privilege to the user BHATIA on the FP1 procedure. The user BHATIA must have a valid user mapping to a remote user ID that has EXECUTE privilege on the Sybase procedure that is referenced by the FP1 procedure. The remote user ID that user BHATIA is mapped to does not need to have SELECT privilege on the Sybase procedure. When the user BHATIA calls the FP1 procedure, the user BHATIA can update the table in Sybase.

Procedure calls and access levels

Procedure calls and access levels have these issues:

- By default, the CALL RESOLUTION IMMEDIATE clause is used with federated procedures when you issue the PRECOMPILE command. The CALL RESOLUTION DEFERRED clause is not supported with federated procedures.
- You cannot not see output when a federated procedure calls a data source procedure that prints to a buffer or the standard output.
- When you call a federated procedure from an external user-defined function, the federated procedure must not have the access level set to READS SQL DATA or MODIFIES SQL DATA. Federated access is blocked inside external functions.
- The limitations that apply to calling local procedures in pass-through mode also apply to federated procedures.
- Each argument in the CALL statement must be compatible with the corresponding parameter in the procedure. Federated procedures follow the same parameter assignment rules as local procedures.

Transactions

Consider the following issues when using federated procedures with transactions:

- The data source procedure that the federated procedure references must not issue a COMMIT or a ROLLBACK statement. Federation does not enforce this restriction and data inconsistencies might occur if the data source procedure issues a COMMIT or ROLLBACK statement.
- Federated procedures with the MODIFIES SQL DATA access level cannot be invoked inside of triggers, dynamic compound statements, SQL scalar, tables, row functions, and methods. After a SAVEPOINT statement is issued, you cannot call a federated procedure with the MODIFIES SQL DATA access level.
- Federated procedures do not support WITH HOLD cursors, scrollable cursors, or updatable cursors. If the data source procedure uses these types of cursors, you do not receive a warning or error message. Applications that access result sets with data source procedure that use these types of cursors might behave differently. Typically the cursors are returned without the hold capability and forward read-only cursors.

Catalog views

These system catalog views store information about federated procedures:

- SYSCAT.ROUTINES
- SYSCAT.ROUTINESFEDERATED
- SYSCAT.ROUTINEOPTIONS
- SYSCAT.ROUTINEPARMS
- SYSCAT.ROUTINEPARMOPTIONS

Creating federated procedures

You must create a federated procedure for each data source procedure that you want to invoke from the federated server.

Before you begin

- The data source procedure must already exist.
- The wrapper for the data source was registered, and the server definition was created.
- Any necessary user mappings were created.
- The privileges held by the authorization ID of the statement include at least one of the following:
  - IMPLICIT_SCHEMA privilege on the database, if the schema name of the procedure does not refer to an existing schema
  - CREATEIN privilege on the schema, if the schema name of the procedure refers to an existing schema
  - SYSADM or DBADM authority
- The privileges held at the data source by the authorization ID must also include the privilege to select the procedure's description from the remote catalog tables.

About this task

To create a federated procedure, you use CREATE PROCEDURE (Sourced) SQL statement

Procedure

Specify the CREATE PROCEDURE statement according to the options needed for a specific data source.

Data sources and federated procedures

Before you create federated procedures, review how federation supports data source procedures.

Federated procedures for DB2 data sources

Before you create a federated procedure, review how to specify options in the CREATE PROCEDURE statement.

When you create federated procedures, keep these issues in mind:

- DB2 procedures support the IN, OUT, and INOUT parameters.
- You must specify same options that the DB2 procedure specifies for the SQL access, deterministic, and external action clauses of the CREATE PROCEDURE statement.
- If two remote DB2 procedures have the same name, use the NUMBER OF PARAMETERS option to identify the procedure that you want to use.
- The federated database does not issue a COMMIT statement for federated procedures that are created for procedures in DB2 Universal Database™ for z/OS and that contain a COMMIT ON RETURN YES clause.

Example

This example shows how to use the CREATE PROCEDURE statement to create a federated procedure for a data source procedure on DB2. You can issue the CREATE PROCEDURE statement from the DB2 command line or create a federated procedure in Data Studio.

```
CREATE PROCEDURE PROC1 SOURCE KELLER.PROC1_DB2
     NUMBER OF PARAMETERS 3 FOR SERVER DB2_SERVER
     SPECIFIC MYPROC1 WITH RETURN TO CLIENT ALL
     MODIFIES SQL DATA DETERMINISTIC EXTERNAL ACTION;
```

PROC1
Required. Specifies the name of the federated procedure.

SOURCE KELLER.PROC1_DB2
Required. Specifies the schema and name for the DB2 procedure. For DB2 procedures, you specify a two-part name in the CREATE PROCEDURE statement. The format for this two-part name is *source_schema_name.source_procedure_name*.

NUMBER OF PARAMETERS 3
Specifies the total number of IN, OUT, and INOUT parameters that the DB2 procedure uses. Use this parameter when you have more than one procedure with the same schema name and procedure name. For example, if your schema is KELLER and you have a PROC1 procedure with three parameters and another PROC1 procedure with one parameter, the name for both of these procedures is KELLER.PROC1. The value for the NUMBER OF PARAMETERS in the data source procedure indicates which procedure you refer to in the CREATE PROCEDURE statement.

FOR SERVER DB2_SERVER
Required. Specifies a server definition where the federated procedure is created.

SPECIFIC MYPROC1
Specifies a unique name for the federated procedure that you are creating. This parameter is used only for federated procedures and is not associated with data source procedures. If you do not specify a unique name, a name is generated by the federated database manager.

WITH RETURN TO CLIENT ALL
Specifies that the result set is returned to the client application. Federation returns a maximum of one result set. If this parameter is not specified, the default is WITH RETURN TO CALLER ALL.

MODIFIES SQL DATA
Indicates the level of data access for SQL statements that are included in the federated procedure. If the clause specified does not match the DB2 procedure, an error message is returned. If you do not specify this clause, the clause for the DB2 procedure is used.

DETERMINISTIC
Specifies if the federated procedure always returns the same results for a given set of argument values. This parameter can improve the performance of the interaction between the federated server and the data source. If the clause specified does not match the DB2 procedure, an error message is returned. If you do not specify this clause, the clause for the DB2 procedure is used.

EXTERNAL ACTION
Specifies if the federated procedure takes an action that changes the state

of an object that is not managed by the database manager. If the clause specified does not match the DB2 procedure, an error message is returned. If you do not specify this clause, the clause for the DB2 procedure is used.

Federated procedures and Microsoft SQL Server

Before you create a federated procedure, review which parameters are supported, how result sets are returned, and what limitations exist.

Parameters

Microsoft SQL Server procedures support the use of optional INPUT and OUTPUT parameters. The SQL Server wrapper maps each INPUT parameter to a federated IN parameter and maps each OUTPUT parameter to a federated INOUT parameter. If you use optional parameters in an SQL Server procedure, you must count them when you specify the NUMBER OF PARAMETERS clause of the CREATE PROCEDURE statement.

Result sets

The SQL Server wrapper can return a result set and an OUTPUT parameter. When an SQL Server procedure returns both an OUTPUT parameter and a result set, only the parameter is returned. The result set is discarded, and you receive the SQL0464W error message. The DB2_RETURN_STATUS value is retrieved for procedures that return result sets, but always returns a value of zero (0), regardless of the actual return value from the procedure.

Limitations

The SQL Server wrapper cannot call a procedure in these situations:

- When another procedure is already called
- When another statement is executed during a single connection

To work around these limitations, you can define the federated procedure on another server. In this example, the following statements succeed if the nickname *sql_nick* and the procedure *sql_proc* are defined on the different servers. If the nickname and the procedure are defined on the same server, the statements fail.

```
DECLARE clientcur CURSOR FOR SELECT colsml,coldec,colvch,coltsp
FROM sql_nick OPEN clientcur; CALL sql_proc();
```

Example

This example shows how to use the CREATE PROCEDURE statement to create a federated procedure for a data source procedure on Microsoft SQL Server. Note that Microsoft SQL Server does not support the UNIQUE ID clause and does not support the source_package_name value of the SOURCE clause.

```
CREATE PROCEDURE PROC1 SOURCE BHATIA.PROC1_MSSQL
     NUMBER OF PARAMETERS 5 FOR SERVER MSSQL_SERVER
     SPECIFIC MYPROC1 WITH RETURN TO CLIENT ALL
     MODIFIES SQL DATA DETERMINISTIC EXTERNAL ACTION;
```

PROC1
Required. Specifies the name of the federated procedure.

SOURCE BHATIA.PROC1_MSSQL
Required. Specifies the name of the schema and procedure on the data source.

FOR SERVER MSSQL_SERVER
Required. Specifies a server definition where the federated procedure is created.

NUMBER OF PARAMETERS 5
Specifies the total number of IN and OUTPUT parameters that the data source procedure uses.

SOURCE BHATIA.PROC1_SQL
Specifies the schema and name for the data source procedure. The format for this two-part name is *source_schema_name.source_procedure_name*.

SPECIFIC MYPROC1
Specifies a unique name for the federated procedure that you are creating. This parameter is used only for federated procedures and is not associated with data source procedures. If you do not specify a unique name, a name is generated by the federated database manager.

WITH RETURN TO CLIENT ALL
Specifies that the result set is returned to the client application. Federation returns a maximum of one result set. If this parameter is not specified, the default is WITH RETURN TO CALLER ALL.

MODIFIES SQL DATA
Indicates the level of data access for SQL statements that are included in the federated procedure. If the clause specified does not match the data source procedure, an error message is returned. If you do not specify this clause, the clause for the data source procedure is used.

DETERMINISTIC
Specifies if the federated procedure always returns the same results for a given set of argument values. This parameter can improve the performance of the interaction between the federated server and the data source.

EXTERNAL ACTION
Specifies if the federated procedure takes an action that changes the state of an object that is not managed by the database manager.

Federated procedures and Oracle

You can create a federated procedure and a function mapping for the same Oracle function.

Use a function mapping when you need to use the Oracle function in SQL as a scalar function. Use a federated procedure when you need to use the Oracle function in CALL statements.

Oracle returns only a single value for functions. The return value for the federated procedure appears at the beginning of the parameter list as an extra OUT parameter. The name of the parameter is always DEFAULT. When you specify the NUMBER OF PARAMETERS clause in the CREATE PROCEDURE (Sourced) statement, do not count the return values.

For some Oracle data types, information about the precision, length, and scale is not stored in the Oracle catalog when the parameters of a procedure are declared. When a federated procedure is created, information about the Oracle procedure is gathered from the Oracle catalog. Because information about the precision, length, and scale is not stored in the Oracle catalog, federated procedures behave in the following way:

- Use the maximum length for the parameter data types.

- Map the Oracle NUMBER data types to the federated DOUBLE data type. You can change this mapping by overriding the default forward data type mapping for Oracle NUMBER.

Tip: Overriding the default forward data type mappings affects other federated DDL operations, such as CREATE NICKNAME. Therefore, before you create the procedure, change the type mapping. Create the procedure for the Oracle procedure with the new type mapping, and then DROP the new type mapping. Subsequent nicknames and procedures that you create will use the default type mapping.

Example

This example shows how to use the CREATE PROCEDURE statement to create a federated procedure for a data source procedure on Oracle. You can issue the CREATE PROCEDURE statement from the DB2 command line or create a federated procedure in Data Studio.

```
CREATE PROCEDURE PROC2 SOURCE ZELLER_SCHEMA.ORACLE_PKG9.PROC2
     NUMBER OF PARAMETERS 5 UNIQUE_ID '2' FOR SERVER ORA_SERVER
     SPECIFIC MYPROC1 WITH RETURN TO CLIENT ALL
     MODIFIES SQL DATA DETERMINISTIC NO EXTERNAL ACTION;
```

PROC2
Required. Specifies the name of the federated procedure.

SOURCE ZELLER_SCHEMA.ORACLE_PKG9.PROC2
Required. Specifies the schema, package, and name for the Oracle procedure or function. If the Oracle procedure or function is in a package, you must specify a three-part name in the CREATE PROCEDURE statement. The format for this three-part name is *source_schema_name.source_package_name.source_procedure_name*. If the Oracle procedure or function is not in a package, you must specify a two-part name in the CREATE PROCEDURE statement. The format for this two-part name is *source_schema_name.source_procedure_name*.

NUMBER OF PARAMETERS 5
Specifies the total number of IN, OUT, and INOUT parameters that the Oracle procedure uses. Use this parameter when you have more than one procedure with the same schema name and procedure name. For example, if your schema is ZELLER and you have a PROC1 procedure with two parameters and another PROC1 procedure with three parameters, the name for both of these procedures is ZELLER.PROC1. The value for the NUMBER OF PARAMETERS in the data source procedure indicates which procedure you refer to in the CREATE PROCEDURE statement. Oracle REFCURSOR parameters must be included in the NUMBER OF PARAMETERS count.

UNIQUE_ID '2'
Specifies the unique identifier for the Oracle procedure. Use the UNIQUE_ID parameter only when the schema name, the procedure name, and the number of parameters do not uniquely identify an Oracle procedure. The UNIQUE ID is the value in the ALL_ARGUMENTS.OVERLOAD column in the Oracle system catalog. If you do not specify the UNIQUE ID parameter, the federated server detects the overloaded procedures and returns an error. Use this option only with Oracle procedures.

FOR SERVER ORA_SERVER
Required. Specifies a server definition where the federated procedure is created.

SPECIFIC MYPROC1
Specifies a unique name for the federated procedure that you are creating. This parameter is used only for federated procedures and is not associated with data source procedures. If you do not specify a unique name, a name is generated by the federated database manager. This parameter is optional.

WITH RETURN TO CLIENT ALL
Specifies that the result set is returned to the client application. Federation returns a maximum of one result set. If this parameter is not specified, the default is WITH RETURN TO CALLER ALL.

MODIFIES SQL DATA
Indicates the level of data access for SQL statements that are included in the federated procedure. If the clause specified does not match the Oracle procedure, an error message is returned. If you do not specify this clause, the clause for the Oracle procedure is used.

DETERMINISTIC
Specifies if the federated procedure always returns the same results for a given set of argument values. This parameter can improve the performance of the interaction between the federated server and the data source.

NO EXTERNAL ACTION
Specifies if the federated procedure takes an action that changes the state of an object that is not managed by the database manager.

Overloaded procedures in federated systems

Overloaded procedures are procedures that have identical names and schemas. Overloaded procedures can have a different number of parameters or different parameter signatures. The purpose of overloading a procedure is to create similar versions of a procedure.

The federated server allows overloaded procedures only if each procedure has a different number of parameters.

Oracle allows overloaded procedures if each procedure has a different number of parameters or if the parameter types are different. You can create federated procedures for Oracle overloaded procedures.

To distinguish procedures that use the identical name, schema name and number of parameters, you must specify the UNIQUE ID when you create the federated procedure.

You can determine the UNIQUE ID by querying the Oracle catalog.

Query the Oracle catalog

To determine the UNIQUE ID, you can query the OVERLOAD column in the Oracle SYS.ALL_ARGUMENTS catalog. The UNIQUE ID value is a character literal that contains a number, such as 1. You can use the passthru mode on the federated server or query the Oracle client directly.

For example, to query the Oracle catalog and display the signature and overload column for a procedure that begins with HJZ, use the following SELECT statement:

```
SELECT owner, package_name, object_name, overload, position, argument_name, in_out,
data_type
   FROM all_arguments aa
   WHERE object_name like 'HJZ%'
   ORDER BY owner, package_name, object_name,overload, position;
```

The following output from the above query shows that the HJZ_PACK1 package contains three procedures that use the name HJZTEST1. You determine the number of procedures by looking at the OBJECT_NAME AND OVERLOAD columns. The first procedure has one IN parameter with a number data type. The second procedure has one IN parameter with a character data type. The third procedure has one OUT parameter with a character data type and one IN parameter with a number data type. The output shows that there are two procedures in the HJZ_PACK1 package that use the name HJZTEST3. There is also a procedure with the name HJZTEST1 that is not in the package. This last procedure has one IN parameter that uses a number data type.

```
OWNER    PACKAGE_NAME OBJECT_NAME OVERLOAD POSITION ARGUMENT_NAME IN_OUT DATA_TYPE
-------- ------------ ----------- -------- -------- ------------- ------ ---------
J15USER1 HJZ_PACK1    HJZTEST1    1        1        A             IN     NUMBER
J15USER1 HJZ_PACK1    HJZTEST1    2        1        A             IN     CHAR
J15USER1 HJZ_PACK1    HJZTEST1    3        0        -             OUT    CHAR
J15USER1 HJZ_PACK1    HJZTEST1    3        1        A             IN     NUMBER
J15USER1 HJZ_PACK1    HJZTEST3    1        1        A             IN     NUMBER
J15USER1 HJZ_PACK1    HJZTEST3    1        2        B             OUT    NUMBER
J15USER1 HJZ_PACK1    HJZTEST3    2        1        A             IN     CHAR
J15USER1 HJZ_PACK1    HJZTEST3    2        2        B             OUT    CHAR
J15USER1 -            HJZTEST1    -        1        A             IN     NUMBER
9 record(s) selected.
```

To create a federated procedure for the second overloaded procedure with an IN parameter of CHAR data type, issue the following CREATE PROCEDURE statement:

```
CREATE PROCEDURE HJZTEST1 SOURCE J15USER1.HJZ_PACK1.HJZTEST1
 NUMBER OF PARAMETERS 1 UNIQUE ID '2'
 FOR SERVER ORA_SERVER WITH RETURN TO CLIENT ALL;
```

Important: In the above example, the NUMBER OF PARAMETERS clause does not uniquely identify the procedure. There are two procedures in the table with the name HJZTEST1 and each procedure has one parameter. You must specify the UNIQUE ID clause to indicate the overloaded procedure that you want to use. Use the value from the OVERLOAD column as the value for the UNIQUE_ID clause. When the UNIQUE ID clause is specified, the NUMBER OF PARAMETERS clause is optional. Use the NUMBER OF PARAMETERS clause to validate that the data source procedure has the number of parameters that you expect.

Federated procedures and Sybase

Before you create a federated procedure, review which parameters are supported, how result sets are returned, and what limitations exist.

Parameters

Sybase procedures use INPUT and OUTPUT parameters. The Sybase wrapper maps a Sybase INPUT parameter to a federated IN parameter and maps a Sybase OUTPUT parameter to a federated INOUT parameter. Although you can use optional parameters in a Sybase procedure, you cannot use optional parameters in a federated procedure. Therefore, when you issue a CALL statement, you must specify all of the parameters.

For Sybase Version 12.0, all of the parameters are input parameters. Federated procedures cannot return an output parameter value. This is a Sybase catalog limitation and does not apply to later versions of Sybase.

Result sets

For Sybase procedures that return an output parameter and a result set, the result set is discarded. If the Sybase procedure returns a result set and return value, a return value of 0 is provided, regardless of the actual return value from the data source procedure. You will not receive a warning message in either of these situations.

However, the result sets are discarded and the SQL0464W message is returned when both of the following conditions are true:

- The federated procedure is defined for a Sybase procedure that returns result sets.
- The federated procedure is invoked inside a trigger or a user-defined function.

Limitations

The Sybase wrapper cannot call a procedure in these situations:

- When another procedure is already called
- When another statement is executed during a single connection

To work around these limitations, you can define the federated procedure on another server. In this Sybase example, the following statements succeed if the nickname *syb_nick* and the procedure *syb_proc* are defined on the different servers. If the nickname and the procedure are defined on the same server, the statements fail.

```
DECLARE clientcur CURSOR FOR SELECT colsml,coldec,colvch,coltsp
FROM syb_nick OPEN clientcur; CALL syb_proc();
```

Example

This example shows how to use the CREATE PROCEDURE statement to create a federated procedure for a data source procedure on Sybase. Note that Sybase does not support the UNIQUE ID clause of the CREATE PROCEDURE statement.

```
CREATE PROCEDURE PROC1 SOURCE BHATIA.PROC1_SYBASE
     NUMBER OF PARAMETERS 3 FOR SERVER SYBASE_SERVER
     SPECIFIC MYPROC1 WITH RETURN TO CLIENT ALL
     MODIFIES SQL DATA DETERMINISTIC EXTERNAL ACTION;
```

PROC1
Required. Specifies the name of the federated procedure.

SOURCE BHATIA.PROC1_SYBASE
Required. Specifies the schema and name for the Sybase procedure. For Sybase procedures, you specify a two-part name in the CREATE PROCEDURE statement. The format for this two-part name is *source_schema_name.source_procedure_name*.

NUMBER OF PARAMETERS 3
Specifies the total number of IN, OUT, and INOUT parameters that the Sybase procedure uses. Use this parameter when you have more than one procedure with the same schema name and procedure name. For example, if your schema is BHATIA and you have a PROC1 procedure with three parameters and another PROC1 procedure with one parameter, the name

for both of these procedures is BHATIA.PROC1. The value for the NUMBER OF PARAMETERS in the data source procedure indicates which procedure you refer to in the CREATE PROCEDURE statement.

FOR SERVER SYBASE_SERVER
Required. Specifies a server definition where the federated procedure is created.

SPECIFIC MYPROC1
Specifies a unique name for the federated procedure that you are creating. This parameter is used only for federated procedures and is not associated with data source procedures. If you do not specify a unique name, a name is generated by the federated database manager.

WITH RETURN TO CLIENT ALL
Specifies that the result set is returned to the client application. Federation returns a maximum of one result set. If this parameter is not specified, the default is WITH RETURN TO CALLER ALL.

MODIFIES SQL DATA
Indicates the level of data access for SQL statements that are included in the federated procedure. If the clause specified does not match the Sybase procedure, an error message is returned. If you do not specify this clause, the clause for the Sybase procedure is used.

DETERMINISTIC
Specifies if the federated procedure always returns the same results for a given set of argument values. This parameter can improve the performance of the interaction between the federated server and the data source.

EXTERNAL ACTION
Specifies if the federated procedure takes an action that changes the state of an object that is not managed by the database manager.

Granting or revoking authorizations to call federated procedures

The administrator of the federated database must grant other users the required authorizations to call the federated procedures.

Before you begin

The user that calls the federated procedure must have a valid user mapping from the federated server to the data source. The remote user ID must have the authorization on the data source that is equivalent to the EXECUTE authorization on the federated server. A user can be granted EXECUTE authorization on the federated procedure. But if the user authorization on the data source is not equivalent to the EXECUTE authorization on the federated server, calls to the data source procedure fail.

The authorization ID for the GRANT statement must have at least one of the following authorities:

- The WITH GRANT OPTION for EXECUTE on the federated procedure
- SYSADM or DBADM authority

Procedure

Issue the GRANT statement from the command line to specify the authorization privileges.

Example 1:

To grant the EXECUTE privilege on all procedures in the BHATIA schema, including any procedures that are created in the future, to users in the HR_DEPT group, use the following statement:

```
GRANT EXECUTE ON PROCEDURE
 BHATIA.* TO HR_DEPT
```

Example 2:

To grant the EXECUTE privilege on the PROC1 procedure to user ZELLER and give the this person the ability to grant the EXECUTE privilege on this procedure to others, use the following statement:

```
GRANT EXECUTE ON PROCEDURE PROC1
 TO ZELLER
 WITH GRANT OPTION
```

Example 3:

To grant the EXECUTE privilege to user ERFAN on the PROC2 procedure that was created with a specific name of MY_PROC2, use the following statement:

```
GRANT EXECUTE ON SPECIFIC
 PROCEDURE MY_PROC2 TO ERFAN
```

Displaying parameter information and calling federated procedures

Query the SYSCAT.ROUTINESFEDERATED catalog view to display parameter information. Then use the CALL statement to call a federated procedure.

Before you begin

The user who calls a federated procedure must have EXECUTE privilege on the data source and a valid user mapping to a remote data source user ID that has the privilege to access the data source tables.

Procedure

To display parameter information and call a federated procedure, use one of these methods:

1. Use the SELECT statement to query the SYSCAT.ROUTINEPARMS view in the federated database catalog.
2. Issue a CALL statement.

This example uses a SELECT statement to obtain information about the federated procedure. For example, if the federated procedure FEDPROC1 is in the federated schema BOB, issue this SELECT statement:

```
SELECT ordinal, char(parmname,30)
 AS name,
   rowtype, char(typename,30) AS type
   FROM syscat.routineparms
   WHERE routinename='FEDPROC1' AND
    routineschema = 'BOB'
   ORDER BY ordinal;
```

The result of the query lists the parameters:

```
ORDINAL NAME ROWTYPE TYPE
------- ---- ------- --------
1       P1   P       INTEGER
2       P2   O       VARCHAR
```

The P row type indicates an input parameter. The O row type indicates an output parameter. Not shown in this example is a B row type that indicates an INOUT parameter.

Altering or dropping federated procedures

You can modify an existing federated procedure by changing the data type of one or more parameters of the federated procedure.

Before you begin

The authorization ID for the DROP PROCEDURE statement must have one of the following authorities:

- SYSADM or DBADM authority
- DROPIN privilege on the schema for the federated procedure
- Definer of the procedure as recorded in the DEFINER column of the catalog view for the federated procedure
- CONTROL privilege on the federated procedure

About this task

In addition to changing the data type of a parameter, you might need to make other changes to a federated procedure. For example, you might need to change a federated procedure if the remote procedure changes. In this case, you cannot directly modify the federated procedure. You must first drop the procedure and then recreate the procedure with the new settings. Otherwise, you must replace the existing procedure by using the CREATE OR REPLACE PROCEDURE statement.

When you drop a federated procedure, the procedure is deleted from the system catalog on the federated database, but the data source procedure is not affected. Keep in mind that when you drop a federated procedure, applications that are dependent on the dropped procedures may be adversely affected.

You can use the command line to drop a federated procedure.

Procedure

Issue the DROP statement to drop a federated procedure.
For example:

```
DROP PROCEDURE federated_procedure_name
```

Joining the result sets of federated procedures

You can join the result sets returned by a federated procedure with the **DB2FEDGENTF** command.

Before you begin

- To use the **-create** parameter of the **DB2FEDGENTF** command, you must have either the DBADM authority on the federated database or a combination of the following authorities or privileges on the federated database:
  1. You must have one of the following authorities:
     - CREATE_EXTERNAL_ROUTINE
     - CREATETAB

2. You also must have one of the following authorities or privileges:
   - Write privilege for the *install_dir*/sqllib/function directory, where *install_dir* is the directory where the federated server is installed
   - IMPLICIT_SCHEMA authority if the implicit or explicit schema name of the function does not exist
   - CREATEIN privilege on the schema if the schema name of the function exists

- To use the **-drop** parameter of the **DB2FEDGENTF** command you must have at least one of the following authorities or privileges on the federated database:
  - DROPIN privilege on the schema for the object
  - OWNER of the object
  - DBADM authority

About this task

You join the result sets that are returned by federated procedures to join data from local and remote tables, particularly in systems that only allow access through federated procedures.

Procedure

1. Create and register a table function with the **-create** parameter of **DB2FEDGENTF** command, for example:

```
DB2FEDGENTF –db sample –u user1 –p password1
  -create
    -stpn S1_INVENTORY
    -tfn S1_INVTRY_TF
    -c 'PART_NUM CHAR(10), S1_QTY INT'
```

2. Use a join to combine the data in the federated procedure result sets. You can join the result set of a federated procedure with a local table or you can join the result sets from two federated procedures.

Examples

In the following examples, the stored procedures named ProINVENTORY and ProINVENTORY2 exist and each return the inventory of two suppliers as result sets. The PARTS table also exists and contains the manufacturer's inventory with the following column names and types:

Table 5. Columns of the PARTS table

Column name	Column type
PART_NUM	CHAR(10)
PART_DESC	VARCHAR(400)
INVENTORY	INT

Joining result sets with local tables example

In this example, the inventory between one supplier and the manufacturer is combined to determine the total inventory available:

1. Use the CREATE PROCEDURE statement to create the S1_INVENTORY federated procedure from the ProINVENTORY stored procedure:

```
CREATE PROCEDURE S1_INVENTORY
 SOURCE ProINVENTORY
 FOR SERVER ORA1;
```

2. Use the **DB2FEDGENTF** command to create the S1_INVTRY_TF table function that includes the PART_NUM and S1_QTY result set:

```
DB2FEDGENTF –db sample –u user1 –p password1
  -create
    -stpn S1_INVENTORY
    -tfn S1_INVTRY_TF
    -c 'PART_NUM CHAR(10), S1_QTY INT'
```

3. Use a join to combine the data of the PARTS table with the result set of the S1_INVTRY_TF table function:

```
SELECT a.PART_NUM, a.PART_DESC, a.INVENTORY + b.S1_QTY as MAX_QTY
 FROM PARTS a, TABLE(S1_INVTRY_TF()) b
 WHERE a. PART_NUM = b. PART_NUM
```

Joining two result sets example

In this example, the manufacturer combines the inventory from both suppliers to determine their total available inventory:

1. Create the S1_INVTRY_TF table function for the first supplier:
   a. Use the CREATE PROCEDURE statement to create the S1_INVENTORY federated procedure from the ProINVENTORY stored procedure:

```
CREATE PROCEDURE S1_INVENTORY
 SOURCE ProINVENTORY
 FOR SERVER ORA1;
```

   b. Use the **DB2FEDGENTF** command to create the S1_INVTRY_TF table function that includes the PART_NUM and S1_QTY result set:

```
DB2FEDGENTF –db sample –u user1 –p password1
  -create
    -stpn S1_INVENTORY
    -tfn S1_INVTRY_TF
    -c 'PART_NUM CHAR(10), S1_QTY INT'
```

2. Create the S2_INVTRY_TF table function for the second supplier:
   a. Use the CREATE PROCEDURE statement to create the S2_INVENTORY federated procedure from the ProINVENTORY2 stored procedure:

```
CREATE PROCEDURE S2_INVENTORY
 SOURCE ProINVENTORY2
 FOR SERVER SYBA1;
```

   b. Use the **DB2FEDGENTF** command to create the S2_INVTRY_TF table function that includes the PART_NUM and S2_QTY result set:

```
DB2FEDGENTF –db sample –u user1 –p password1
  -create
    -stpn S2_INVENTORY
    -tfn S2_INVTRY_TF
    -c 'PART_NUM CHAR(10), S2_QTY INT'
```

3. Use a join to combine the result sets of the S1_INVTRY_TF and S2_INVTRY_TF table functions:

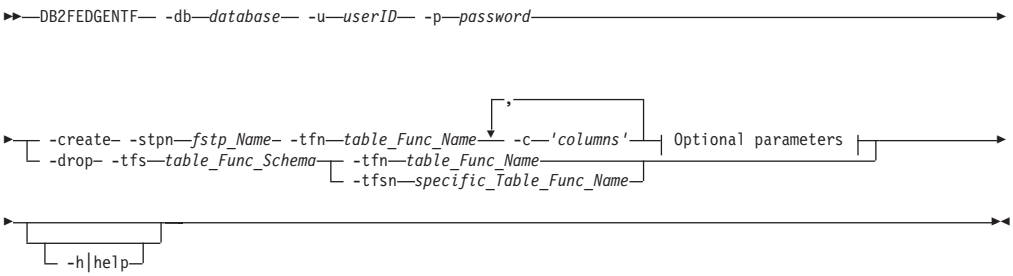
```
SELECT a.PART_NUM, a.PART_DESC, b.S1_QTY + c.S2_QTY as MAX_SUPP_QTY
 FROM PARTS a, TABLE(S1_INVTRY_TF()) b, TABLE(S2_INVTRY_TF()) c
 WHERE a. PART_NUM = b. PART_NUM
 AND a. PART_NUM = c. PART_NUM
```

DB2FEDGENTF command

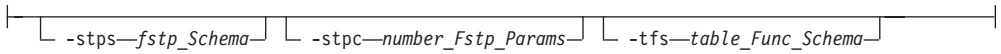
Creates and registers table functions that return result sets from federated procedures.

The **db2fedgentf** command retrieves the JDK path from the DBM configuration parameter JDK_PATH.

Syntax



Optional parameters:



Parameters

-db *database*
Specifies the name of the database that you want to connect to.

-u *userID*
Specifies the federated database user ID.

-p *password*
Specifies the password of the user ID.

-create
Creates and registers a table function in the current schema if the **-tfs** parameter is not specified. The table function returns the specified columns of the result set from the federated procedure.

-stpn *fstp_Name*
Specifies the name of the federated procedure.

-tfn *table_Func_Name*
Specifies the name of the table function. If the name of the table function is not specified, the federated procedure name is used.

-c '*columns*'
Specifies a comma-delimited list that includes the column name and column-type pairs in the signature of the result set returned by the federated procedure.

Example

The column names are PID, PRICE, and QTY, and the column types are CHAR(10), DOUBLE, and INT:

```
'PID CHAR(10), PRICE DOUBLE, QTY INT'
```

-stps *fstp_Schema*
Specifies the schema of the federated procedure. This parameter is optional. If a name is not specified, the default SQL schema as defined in the CURRENT SCHEMA special register is used.

-stpc *number_Fstp_Params*
Specifies the number of the inputs for the federated procedure. This parameter is optional. If the number of the inputs is not specified, the federated procedure is determined by the specified federated procedure name. If the federated procedures are overloaded, you receive an error.

-tfs *table_Func_Schema*
Specifies the schema of the table function. This parameter is optional. If the schema of the table function is not specified, the default SQL schema as defined in the CURRENT SCHEMA special register is used.

- drop
Drops the table function that you specify. The description from the catalog is also deleted and all packages that reference the specified table function become invalid.

-tfs *table_Func_Schema*
Specifies the schema of the table function that you want to drop. If schema of the table function is not specified, the default SQL schema as defined in the CURRENT SCHEMA special register is used.

-tfn *table_Func_Name*
Specifies the name of the table function that you want to drop.

-tfsn *specific_Table_Func_Name*
For overloaded functions, specifies the specific name of the table function that you want to drop. This parameter is mutually exclusive with the `-tfn` *table_Func_Name* parameter. You do not need to specify this option if the table function is uniquely identified by its name and schema. This parameter is optional.

-h|help
Provides usage information for the **DB2FEDGENTF** command.

Examples

In this example, the **DB2FEDGENTF** command runs on the EAST_INVTRY federated procedure to create the table function S1_INVTRY_TF that returns the PRODID, PRICE, and QTY columns:

```
DB2FEDGENTF  -db sample  -u user1  -p password1
   -create
     -stpn EAST_INVTRY
     -tfn E_INVTRY_TF
     -c 'PRODID INT, PRICE DOUBLE, QTY INT'
```

Federated procedure troubleshooting

If you encounter problems with federated procedures, there are several ways that you can troubleshoot the problems.

The following queries and diagnostic tools help you to view information about the federated procedures. This information will assist you in resolving problems with the federated procedures.

Verify data source procedure information

If the SQL1253N error is returned when you issue a CREATE PROCEDURE statement, you can issue the following queries against the catalog tables on the data source to verify information about the data source procedure. The SQL1253N

error indicates that the source procedure specified in the CREATE PROCEDURE (Sourced) statement was not found at the data source. You can query the Oracle server directly or use a pass-through session to query the Oracle server.

For procedures in DB2 for Linux, UNIX, and Windows:

```
SELECT parm_count, result_sets,
 sql_data_access, deterministic, external_action
 FROM syscat.routines
WHERE routineschema = ''
AND routinename = ''
AND routinetype = 'P'
AND parm_count = '' <-- optional
```

For procedures in DB2 for iSeries®:

```
SELECT in_parms+out_parms+inout_parms,
number_of_results, sql_data_access, deterministic,
external_action
FROM qsys2.sysroutines
WHERE routine_schema = ''
and routine_name = ''
and routine_type = 'PROCEDURE'
and in_parms+out_parms+inout_parms = ''; <-- optional
```

For procedures in DB2 for z/OS:

```
SELECT parm_count, result_sets,
 sql_data_access, deterministic, external_action
 FROM sysibm.sysroutines
WHERE schema = ''
AND name = ''
AND routinetype = 'P'
```

For Microsoft SQL Server

```
SELECT id
 FROM dbo.sysobjects
  WHERE id = object_id AND
   (TYPE = 'P' or TYPE = 'X')
```

For Oracle procedures that are in a package:

```
SELECT owner, package_name, object_name, overload, parm_count
   FROM (
      SELECT owner, package_name, object_name, overload,
      SUM(case
         WHEN data_type IS NULL
         THEN 0
         ELSE 1
         END)
      AS parm_count
      FROM sys.all_arguments
      WHERE data_level = 0
      GROUP BY owner, package_name, object_name, overload
      ) aa
   WHERE object_name = '' AND
      package_name = '' AND
      owner = '' AND
      overload = '' AND   <-- optional
      parm_count =;       <-- optional
```

For Oracle procedures that are not in a package:

```
SELECT object_name, object_type, status
   FROM sys.all_objects
   WHERE owner = '' AND
     object_name = '' AND
     object_type IN ('PROCEDURE', 'FUNCTION')
```

For Sybase procedures:

```
SELECT id
   FROM dbo.sysobjects
   WHERE id = object_id('.') AND
     (TYPE = 'P' OR TYPE ='XP')
```

Diagnostic tools

Use the Explain utility, the DESCRIBE command, or the db2audit command to diagnose problems with federated procedures.

For example, the FED_PROC1 procedure has three OUTPUT parameters. To use the DESCRIBE command on the FED_PROC1 procedure, issue the following command:

```
DESCRIBE CALL FED_PROC1(?,?,?);
```

System monitor

The system monitor elements in the federated database contain information about federated procedures. The monitor elements are as follows:

- The Stored Procedure Time monitor element, stored_proc_time, contains the time it has taken the data source to respond to federated procedure statements.
- The Rows Returned by Stored Procedures monitor element, sp_rows_selected, contains the number of rows that are sent from the data source to the federated server. You can use this element to calculate the average number of rows sent to the federated server from the data source for each federated procedure or to calculate the average time to return a row to the federated server from the data source.
- The Stored Procedures monitor element, stored_procs, contains a count of the total number of procedures that the federated server has called from this data source.

SQL error SQL30090N with return code 21

There are several situations in which the SQL30090N error with return code 21 is returned. One of the most common situations is when a federated procedure is being created using a fenced wrapper. Federated procedures can be created only on trusted wrappers.

Result set not returned

A result set might not be returned to the client or caller for one of the following reasons:

- The clause for returning result sets is not specified correctly in the federated procedure.
- Some data sources do not return sets in the same order each time a procedure is called. Because federated procedures return only the first result set, a different result set might be returned from the data source when the federated procedure is called.

For example, there are two procedures on the data source, PROCEDURE A and PROCEDURE B. PROCEDURE B calls PROCEDURE A. The statements to create these procedures are:

```
CREATE PROCEDURE A ()
BEGIN
   DECLARE cur1 CURSOR WITH RETURN TO CLIENT
   FOR SELECT * FROM t;
   OPEN cur1
END
```

```
CREATE PROCEDURE B (arg1 INT)
BEGIN
   DECLARE cur2 CURSOR WITH RETURN TO CLIENT
   FOR SELECT * FROM t;
   IF arg1<10) THEN
      CALL A();
   END IF;
   OPEN cur2
END;
```

The federated procedure FEDPROC1 references the data source PROCEDURE B. The statement for the FEDPROC1 procedure is:

```
CREATE PROCEDURE FEDPROC1
SOURCE newton.B
FOR SERVER s1
NUMBER OF PARAMETERS 1
WITH RETURN TO CLIENT 1;
```

A local procedure calls the federated procedure FEDPROC1. The statement for the local procedure is:

```
CREATE PROCEDURE local (arg1 INT)
   BEGIN
   CALL FEDPROC1 (arg1)
END;
```

When you issue the `CALL LOCAL(1)` statement, the cur1 result set from PROCEDURE A is returned. The result set cur2 is not returned.

However, if you issue the `CALL LOCAL(20)` statement, the cur2 result set from PROCEDURE B is returned.

Pass-through session (Oracle only)

If you create a data source procedure, function, or package in a pass-through session, a successful message is returned even if the object definition has an error. The object is created on the Oracle server, but it is marked INVALID. You cannot create federated procedures on INVALID objects. When you attempt to create a federated procedure that references an INVALID Oracle object, the CREATE PROCEDURE (Sourced) statement fails.

Use one of the following methods to determine why an object is not valid:

- Use the SHOW ERRORS command in the SQL*Plus utility from Oracle.
- Query the Oracle sys.all_errors catalog table.

Chapter 7. Creating and modifying remote tables by using transparent DDL

With transparent DDL, you can use procedures that you are familiar with to create and modify remote tables by using the federated database without using a pass-through session.

What is transparent DDL

Transparent DDL provides the ability to create and modify remote tables through the federated database without using pass-through sessions.

The SQL statements you use with transparent DDL are CREATE TABLE, ALTER TABLE, and DROP TABLE.

A transparent DDL CREATE TABLE statement creates a remote table at the data source and a nickname for that table at the federated server. It will map the DB2 data types you specify to the remote data types using the default reverse type mappings. In general, the wrappers provide type mappings. You can also create user-defined reverse type mappings to override the default mappings.

The advantage of using transparent DDL is that database administrators can use procedures that they are familiar with to create both local and remote tables. Transparent DDL centralizes table administration and facilitates granting authorizations.

Transparent DDL is supported with the following data sources:

- DB2 for z/OS
- DB2 for System i
- DB2 Database for Linux, UNIX, and Windows
- DB2 Server for VM and VSE
- Informix
- JDBC
- Microsoft SQL Server
- ODBC
- Oracle
- Sybase
- Teradata

The database administrator can use the DB2 command line processor (CLP) to create the tables. Using transparent DDL avoids the need to learn the different DDL syntax required for each data source.

Before you can create remote tables on a data source through the federated database, you need to configure access to the data source:

- The wrapper for that data source needs to be registered in the global catalog
- The server definition needs to be created for the server where the remote table will be located

- The user mappings, if required, need to be created between the federated server and the data source server

The privileges held by the authorization ID of the transparent DDL statements must include at least one of the following:

- SYSADM or DBADM authority
- CREATETAB authority on the database and USE privilege on the table space as well as one of:
  - IMPLICIT_SCHEMA authority on the database, if the implicit or explicit schema name of the table does not exist
  - CREATEIN privilege on the schema, if the schema name of the table refers to an existing schema

To issue transparent DDL statements, your authorization ID must have the necessary privileges on the nickname (for the federated server to accept the request), and the comparable privileges on the remote data source server (for the data source to accept the request).

Remote LOB columns and transparent DDL

You specify the length of a LOB column when using transparent DDL.

Some data sources, such as Oracle and Informix, do not store the lengths of LOB columns in their system catalogs. When you create a nickname on a table, information from the data source system catalog is retrieved including column length. Since no length exists for the LOB columns, the federated database assumes that the length is the maximum length of a LOB column in DB2 Database for Linux, UNIX, and Windows. The federated database stores the maximum length in the federated database catalog as the length of the nickname column.

However, when you create a remote table using transparent DDL you must specify the length of the LOB column. When the federated server creates a nickname on the remote table, it stores the length you specify in the federated database catalog as the length of the nickname column. The maximum length of a LOB column is 2 gigabytes.

Creating remote tables and transparent DDL

When a remote table is created through the federated database using transparent DDL, several other actions occur.

When you create the remote table:

- A nickname is automatically created for the remote table. The nickname has the same name as the table name specified in the CREATE TABLE statement. The remote table has the same name as the table name unless you specify another name using the REMOTE_TABNAME option.
- The schema of the remote table is the nickname schema unless you specify another schema using the REMOTE_SCHEMA option.
- The nickname created using transparent DDL can be used like any other nickname. In addition, you can ALTER and DROP the remote table (something you cannot do with a nickname created using CREATE NICKNAME).
- A row is added in the SYSCAT.TABOPTIONS catalog view with an option name of TRANSPARENT and a value of Y.

Creating new remote tables using transparent DDL

To create a remote table using transparent DDL, you specify the CREATE TABLE statement.

Before you begin

Before you create a remote table, you must configure the federated server to access that data source. This configuration includes:

- Creating the wrapper for that data source type
- Supplying the server definition for the server where the remote table will be located
- Creating the user mappings between the federated server and the data source server

To issue transparent DDL statements, your authorization ID must have the necessary privileges on the nickname (for the federated server to accept the request), and the comparable privileges on the remote data source server (for the data source to accept the request).

The privileges held by the authorization ID issuing the transparent DDL statements must include at least one of the following:

- SYSADM or DBADM authority
- CREATETAB authority on the database and USE privilege on the table space as well as one of:
  - IMPLICIT_SCHEMA authority on the database, if the implicit or explicit schema name of the table does not exist
  - CREATEIN privilege on the schema, if the schema name of the table refers to an existing schema

About this task

Restrictions

The following restrictions apply to creating a remote table using transparent DDL:

- You cannot modify or drop tables that were natively created at the remote data source.
- Materialized query tables cannot be created on remote data sources.
- You can specify basic column information in the table definition, but you will not be able to specify table options or column options. For example, the LOB options (LOGGED and COMPACT) are not supported.
- You cannot specify a comment on a column.
- You cannot generate column contents.
- You can specify a primary key, but you cannot specify a foreign key or check constraints. The columns used for a primary key must be NOT NULL, and cannot include columns containing LOBs.
- You cannot modify the parameters of existing columns, such as the data type or length.
- The DEFAULT clause in the CREATE TABLE statement is not supported.

Procedure

To create a remote table from the command line prompt, issue the CREATE TABLE statement with the appropriate parameters set.

Creating new remote tables using transparent DDL - examples

The following examples illustrate what to specify to create remote tables using transparent DDL and the use of data type mappings.

When you create remote tables using transparent DDL:

- The remote data source must support the column data types and primary key option in the CREATE TABLE statement.

  Example: The remote data source does not support primary keys. Depending on how the data source responds to requests it does not support, an error might be returned or the request might be ignored.
- The remote server must be specified in the OPTIONS clause. The OPTIONS clause can be used to override the remote name or the remote schema of the table being created. The SQL_SUFFIX option is allowed at the end of the CREATE TABLE statement. You can specify this option for any relational data source to add data source-specific options to the CREATE TABLE statement that is issued at the data source.

Example: You want to create the table EMPLOY on an Oracle server. In the CREATE TABLE statement, use the DB2 data types when you specify each column. Using the CLP, the syntax to create the table is:

```
CREATE TABLE EMPLOY
   ( EMP_NO       CHAR(6) NOT NULL,
     FIRSTNAME    VARCHAR(12) NOT NULL,
     MIDINT       CHAR(1) NOT NULL,
     LASTNAME     VARCHAR(15) NOT NULL,
     HIREDATE     DATE,
     JOB          CHAR(8),
     SALARY       DECIMAL(9,2),
     PRIMARY KEY (EMP_NO) )
     OPTIONS (REMOTE_SERVER 'ORASERVER',
     REMOTE_SCHEMA 'J15USER1', REMOTE_TABNAME 'EMPLOY' )
```

EMPLOY
: The name of the nickname associated with the table.

REMOTE_SERVER *'ORASERVER'*
: The name that you supplied for the server in the CREATE SERVER statement. This value is case-sensitive.

REMOTE_SCHEMA *'J15USER1'*
: The remote schema name. Although this parameter is optional, it is recommended that you specify a schema name. If this parameter is not specified, the nickname schema is used for the remote schema name. This value is case-sensitive.

REMOTE_TABNAME *'EMPLOY'*
: The remote table name. This parameter is optional. If this parameter is not specified, the local table name is used for the remote table name. This value must be a valid name on the remote data source and cannot be an existing table name. This value is case-sensitive.

In the example above, the federated database uses reverse data type mappings to map the DB2 data types to Oracle data types. On the remote Oracle server, the

EMPLOY table is created using Oracle data types. The following table shows the mappings from the DB2 data types to the Oracle data types for the columns specified in the example.

Table 6. An example of reverse data type mappings from the federated database to Oracle

Column	DB2 data type specified in the CREATE TABLE statement	Oracle data type used in the remote table
EMP_NO	CHAR(6) NOT NULL	CHAR(6) NOT NULL
FIRST_NAME	VARCHAR(12) NOT NULL	VARCHAR2(12) NOT NULL
MID_INT	CHAR(1) NOT NULL	CHAR(1) NOT NULL
LAST_NAME	VARCHAR(15) NOT NULL	VARCHAR2(15) NOT NULL
HIRE_DATE	DATE	DATE
JOB	CHAR(8)	CHAR(8)
SALARY	DECIMAL(9,2)	NUMBER(9,2)

Altering remote tables using transparent DDL

You can alter remote data source tables that were created through the federated database using transparent DDL. You cannot alter tables that were created directly at the remote data source.

Before you begin

The privileges held by the authorization ID of the transparent DDL statements must include at least one of the following:

- SYSADM or DBADM authority
- CREATETAB authority on the database and USE privilege on the table space as well as one of:
  - IMPLICIT_SCHEMA authority on the database, if the implicit or explicit schema name of the table does not exist
  - CREATEIN privilege on the schema, if the schema name of the table refers to an existing schema

To issue transparent DDL statements, your authorization ID must have the necessary privileges on the nickname (for the federated server to accept the request), and the comparable privileges on the remote data source server (for the data source to accept the request).

About this task

You can use the ALTER TABLE statement to modify tables created through IBM InfoSphere Federation Server using transparent DDL. Using the ALTER TABLE statement you can:

- Add new columns
- Add the table primary key

Do not use the ALTER TABLE statement to add or modify column options. Use the ALTER NICKNAME statement instead.

Restrictions

The following restrictions apply to altering a remote table using transparent DDL:

- You cannot modify tables that were natively created at the remote data source.
- An existing primary key cannot be altered or dropped in a remote table.
- Altering a remote table invalidates any packages dependent on the nickname associated with the remote table.
- The remote data source must support the changes in the ALTER TABLE statement. For example, suppose that the remote data source does not support primary keys. Depending on how the data source responds to requests it does not support, an error might be returned or the request might be ignored.
- You cannot specify a comment on a column.
- You cannot generate column contents.
- You can specify a primary key, but you cannot specify a foreign key or check constraints. The columns used for a primary key must be NOT NULL, and cannot include columns containing LOBs.
- You cannot modify the parameters of existing columns, such as the data type or length.
- The DEFAULT clause in the ALTER TABLE statement is not supported.

Procedure

To alter a remote table using transparent DDL, issue the ALTER TABLE statement.
Example: You want to add a primary key on a remote table EMPLOYEE that you created using transparent DDL. Using the following ALTER TABLE statement to modify the table:

```
ALTER TABLE EMPLOYEE
   ADD PRIMARY KEY (EMP_NO, WORK_DEPT)
```

The columns used for a primary key must be NOT NULL, and cannot be columns that contain LOBs.
Example: You want to add the columns ORDER_DATE and SHIP_DATE to the remote table SPALTEN that was created using transparent DDL. Using the following ALTER TABLE statement to create the table:

```
ALTER TABLE SPALTEN
   ADD COLUMN ORDER_DATE  DATE
   ADD COLUMN SHIP_DATE  DATE
```

Dropping remote tables using transparent DDL

You can drop remote data source tables that were created through the federated database using transparent DDL. You cannot drop tables that were created directly at the remote data source.

Before you begin

The privileges held by the authorization ID of the transparent DDL statements must include at least one of the following:

- SYSADM or DBADM authority
- CREATETAB authority on the database and USE privilege on the table space as well as one of:
  - IMPLICIT_SCHEMA authority on the database, if the implicit or explicit schema name of the table does not exist
  - CREATEIN privilege on the schema, if the schema name of the table refers to an existing schema

To issue transparent DDL statements, your authorization ID must have the necessary privileges on the nickname (for the federated server to accept the request), and the comparable privileges on the remote data source server (for the data source to accept the request).

About this task

To drop a remote table that was created through the federated database using transparent DDL, you can use the DROP statement.

Dropping a nickname for a remote table created using transparent DDL merely drops the local nickname for that table. The DROP NICKNAME statement does not drop the remote table. You must use the DROP TABLE statement to drop the remote table.

Dropping a remote table first deletes the table on the data source, then deletes the corresponding nickname for the remote table in the federated database. Deleting the nickname invalidates any packages based on that nickname.

Restrictions

You cannot drop tables that were natively created at the remote data source.

Procedure

To drop a remote table, issue the DROP TABLE statement.
Example: To drop a table named SPALTEN, issue the following DROP statement:

```
DROP TABLE SPALTEN
```

where *SPALTEN* is the local name for the remote table.

Chapter 8. Managing transactions in a federated system

Federated system transaction processing allows you to read and update databases in a single transaction while maintaining the consistency of the data. Federated systems support single-phase and two-phase commit protocols. When you administer your federated system, you must set up the appropriate protocol.

Understanding federated system transaction support

Knowledge of transaction processing concepts in a DB2 Database for Linux, UNIX, and Windows distributed environment will help you understand federated system transactions.

To understand federated system transaction processing, you should be familiar with the following distributed transaction processing concepts:

- Unit of work (UOW)
- Remote unit of work (RUOW)
- Distributed unit of work (DUOW)
- Multisite update
- Transaction manager (TM)
- Resource manager (RM)
- Type 1 connection
- Type 2 connection
- One-phase commit
- Two-phase commit

These concepts work identically in both federated and non-federated database systems. However, the scope of each concept changes in a federated system.

For example, a unit of work implicitly begins when any data in a database is read or written. For a unit of work in a federated system, the database can be a federated database or a data source database. For a distributed unit of work in a federated system, you can access both a federated database and a data source database.

An application must end a unit of work by issuing either a COMMIT or a ROLLBACK statement, regardless of the number of databases that are accessed. The COMMIT statement makes all changes within a unit of work permanent. The ROLLBACK statement removes these changes from a database. Changes made by a unit of work become visible to other applications after a successful commit.

Recommendation: Always explicitly commit or roll back units of work in your applications.

In a distributed unit of work that involves updates of multiple databases on multiple sites, data must be consistent. The multisite update or two-phase commit protocol is commonly used to ensure data consistency across multiple databases within a distributed unit of work.

Federated transactions support both one-phase commit protocol and two-phase commit protocol. The DB2_TWO_PHASE_COMMIT server option enables two-phase commit support for the following data sources:

- DB2 family data sources, in fenced mode and trusted mode
- Informix, in trusted mode
- Oracle, in fenced mode and trusted mode
- Sybase, in fenced mode
- MS SQL Server, in trusted mode

When a data source is declared as a federated two-phase commit data source, that is, the DB2_TWO_PHASE_COMMIT server option is set to "Y", a commit against this data source uses two-phase commit protocol, even if it is a single site update transaction or a multi site update transaction.

When a data source is declared as a federated one-phase commit data source (the default), and it is a single site update transaction, a commit against this data source uses one-phase commit protocol.

In the following example of a one-phase commit operation, Oracle is defined as a one-phase commit data source:

```
SELECT * FROM oracle_nickname
UPDATE oracle_nickname
COMMIT
```

In the following example of a two-phase commit operation, Oracle and DRDA are defined as two-phase commit data sources:

```
SELECT * FROM oracle_nickname
UPDATE oracle_nickname

SELECT * FROM drda_nickname
UPDATE drda_nickname
COMMIT
```

What is an update in a federated system?

In a federated system, an update is not just a transaction that includes an INSERT, UPDATE, or DELETE statement. There are certain operations that are considered updates in a federated system and certain types of updates that are allowed in federated system.

In a federated system, updates can be performed locally or remotely.

- Local site updates are updates to DB2 tables or views that do not reference nicknames
- Remote site updates are updates to objects on a remote data source. Remote data sources include:
  - Another DB2 Database for Linux, UNIX, and Windows database or instance on the federated server
  - Another DB2 Database for Linux, UNIX, and Windows database or instance on another server
  - Data sources other than DB2 Database for Linux, UNIX, and Windows, such as DB2 for System i, Informix, Oracle, and Teradata

There are four types of actions that the federated server considers to be update transactions. The following table shows the updates that you can perform on a federated system.

Table 7. Types of updates and the site where the updates are performed

Type of action	Local site	Remote site	Explanation
Local update (DDL and DML)	Y	N	An update on an object in the federated database.
Remote update (nickname)	N	Y	An update on a remote data source object that you created a nickname for.
Dynamic SQL in pass-through sessions	N	Y	An update on a remote data source object. You cannot use a pass-through session to update local objects. Even SELECT queries sent in pass-through sessions are considered to be an update action.
Transparent DDL	Y	Y	A pair of transactions that create, alter, or drop remote tables and their corresponding nicknames in a federated database. For example, a pair of transactions that create a remote table on a data source and a nickname on the federated server.

What is an update transaction in a pass-through session?

A federated server treats all dynamic SQL statements sent through pass-through sessions as updates. This behavior ensures data integrity.

If a dynamic SQL statement that is sent through a pass-through session is successful, the transaction is recorded as an update. The SQL can be any type of statement, including SELECT statements.

Data sources that automatically commit DDL statements

Some data sources, such as Oracle, automatically commit the current transaction at their data source sites as part of a DDL statement execution.

If you create a remote table using transparent DDL or in a pass-through session, these data sources cannot rollback the remote table after the table is created. You must delete the remote table manually.

User-defined functions that are pushed down to the data source for processing

If a remote user-defined function performs an update on a data source, the federated server is unaware of the update.

Because the federated server does not treat these user-defined functions as update statements, all the statement level protection that the federated system applies to the update operations is not applicable. As a result, data integrity might be compromised in some situations.

Important: Data integrity cannot be guaranteed when a user-defined function that is pushed down to a data source performs an update.

Chapter 9. Performing two-phase commit transactions

You can use the two-phase commit capability of a federated system to update data for one or more data sources in a single transaction.

Two-phase commit for federated transactions

A federated system can use two-phase commit for transactions that access one or more data sources. Two-phase commit uses the industry standard X/Open XA protocol to coordinate the processing of distributed unit of work transactions.

In a two-phase commit operation, commit processing occurs in two phases: the prepare phase and the commit phase. During the prepare phase in a federated system, a federated server polls all of the federated two-phase commit data sources that are involved in a transaction. This polling activity verifies whether each data source is ready to commit or roll back the data. During the commit phase, the federated server instructs each two-phase commit data source to either commit the data or to roll back the transaction.

In a one-phase commit environment, multiple data sources are updated one data source at a time using separate commit operations. This can cause data synchronization problems if some data sources are successfully updated and others are not.

For example, if a transaction withdraws funds from one account and deposits them in another account using one-phase commit, the system might successfully commit the withdraw operation and unsuccessfully commit the deposit operation. The deposit operation can be rolled back, but the withdraw operation cannot because it has already been successfully committed. The result is that the funds are virtually "lost".

In a two-phase commit environment, the withdraw and deposit transactions are prepared together and either committed or rolled back together. The result is that the integrity of the fund amounts remains intact.

Planning for federated two-phase commit

Federated two-phase commit does not provide benefits in all business environments. Also, there are several factors to consider before you decide to deploy federated two-phase commit.

To use two-phase commit for federated transactions, you must consider the following issues:

- Whether your operating system and data source environment can support two-phase commit for federated transactions.
- Whether your business environment requires two-phase commit for federated transactions.

  To decide whether or not two-phase commit is right for your business environment, you need to understand how two-phase commit for federated transactions works and the problems that it solves.
- How to configure the federated server and compatible data sources to use two-phase commit for federated transactions.

There are basic federated server and data source requirements to use two-phase commit for federated transactions as well as performance considerations you must consider when deploying two-phase commit.

- To manually resolve indoubt transactions, you need to understand the inner workings of two-phase commit for federated transactions.

  Two-phase commit for federated transactions can resolve problems without intervention. But when there are extended network outages, hardware failures, or an urgent need to free up system resources, you can manually resolve problems through *heuristic processing*.

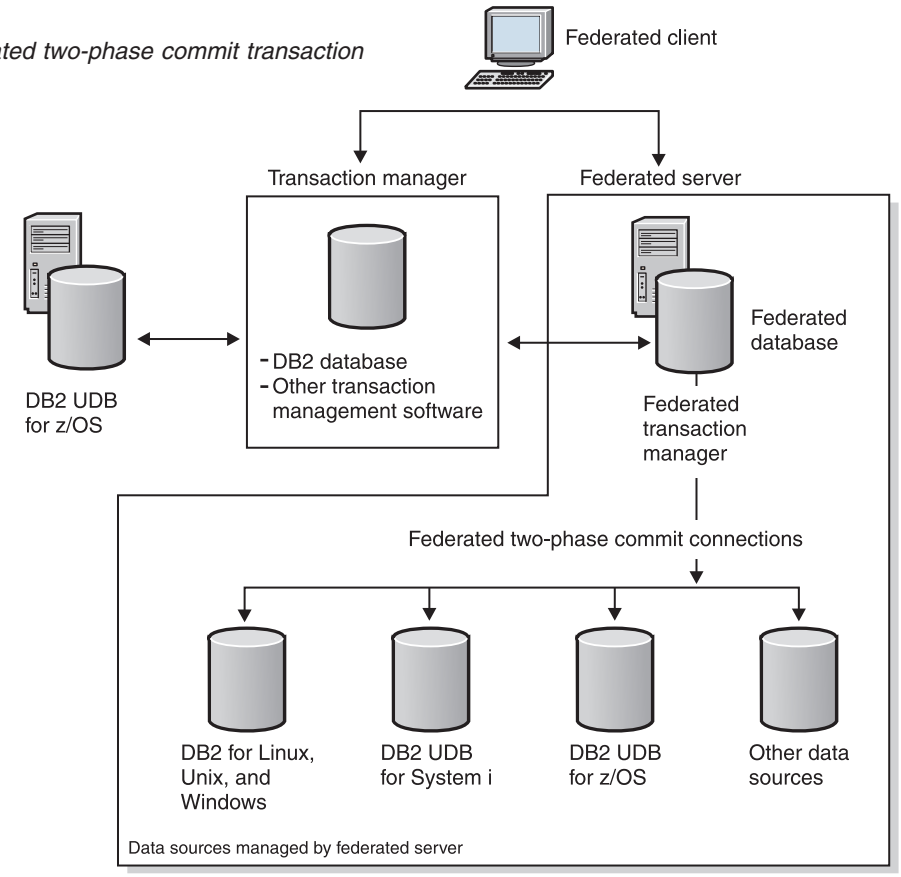
Federated architecture for two-phase commit

Federated two-phase commit is based on the two-phase commit feature available in DB2. In two-phase commit, the X/Open Distributed Transaction Processing (DTP) model has multiple components: transaction identifiers, transaction managers, and resource managers. In federated systems using federated two-phase commit, another component is added, the federated transaction manager.

A federated server becomes a federated transaction manager if the server coordinates activity for one or more remote data sources that use the two-phase commit protocol. A federated transaction manager performs some transaction management functions on behalf of the transaction manager. The client or application that initiates a distributed unit of work transaction and the transaction manager are unaware of the activity that the federated transaction manager coordinates at the remote data sources. The federated transaction manager communicates with DB2 database transaction managers using an XA interface. In addition to any X/Open requirements for two-phase commit, the transaction manager database must be accessible from the federated instance. Resource managers follow the instructions that are provided by a federated transaction manager to commit or roll back a transaction.

The following figure shows an example of a simple two-phase commit transaction in a typical federated system, from client initiation to the data source updates.

Figure 3. Simple federated two-phase commit transaction



In the previous figure, the connection from the client to the transaction manager is a type 2 connection. Each database connection also has its own sync point setting. A sync point is a point in time when all the recoverable data that a program accesses is consistent. Sync point two-phase connections support distributed unit of work transactions with updates to multiple data sources.

When the client connects to the DB2 database, the transaction manager is aware of the transaction, but no additional coordination is required from the federated server. When the federated server connects to the data sources by using the two-phase commit protocol, the federated server becomes the federated transaction manager. The federated server monitors and coordinates the two-phase commits. At this point, the transaction manager is unaware of the two-phase commit transactions with the data sources. The transaction manager only knows that a single transaction is being processed with the federated server.

Data sources are not capable of initiating resynchronization if a failure occurs in a federated system. The federated server initiates the resynchronization process.

Results can be unpredictable if you attempt to access a data source by using multiple paths in the same transaction with federated two-phase commit. For example, if the federated server is a resource manager to an external transaction manager, the data source might be accessed indirectly from the federated server and directly as a resource manager to the transaction manager. In this case, the data source might not be able to tell whether these two paths are from the same global transaction. The data source might create two transaction entries for the same global transaction and treat each transaction as separate from the other,

possibly leading to unpredictable results. The data source might also detect that the two paths are from the same global transaction, and reject the second path.

Two-phase commit for federated transactions - examples

A federated system that uses two-phase commit can be configured in several different ways. Configuration choices depend on the required solution.

Configurations can use Type 1 or Type 2 connections.

Type 1 connections are connections in which an application process is connected to an application server according to the rules for remote unit of work.

Type 2 connections are connections in which an application process is connected to an application server and establishes the rules for application-directed distributed unit of work. The application server is then the current server for the process.

The following figure shows a DB2 Type 1 connection with a federated server functioning as the transaction manager.

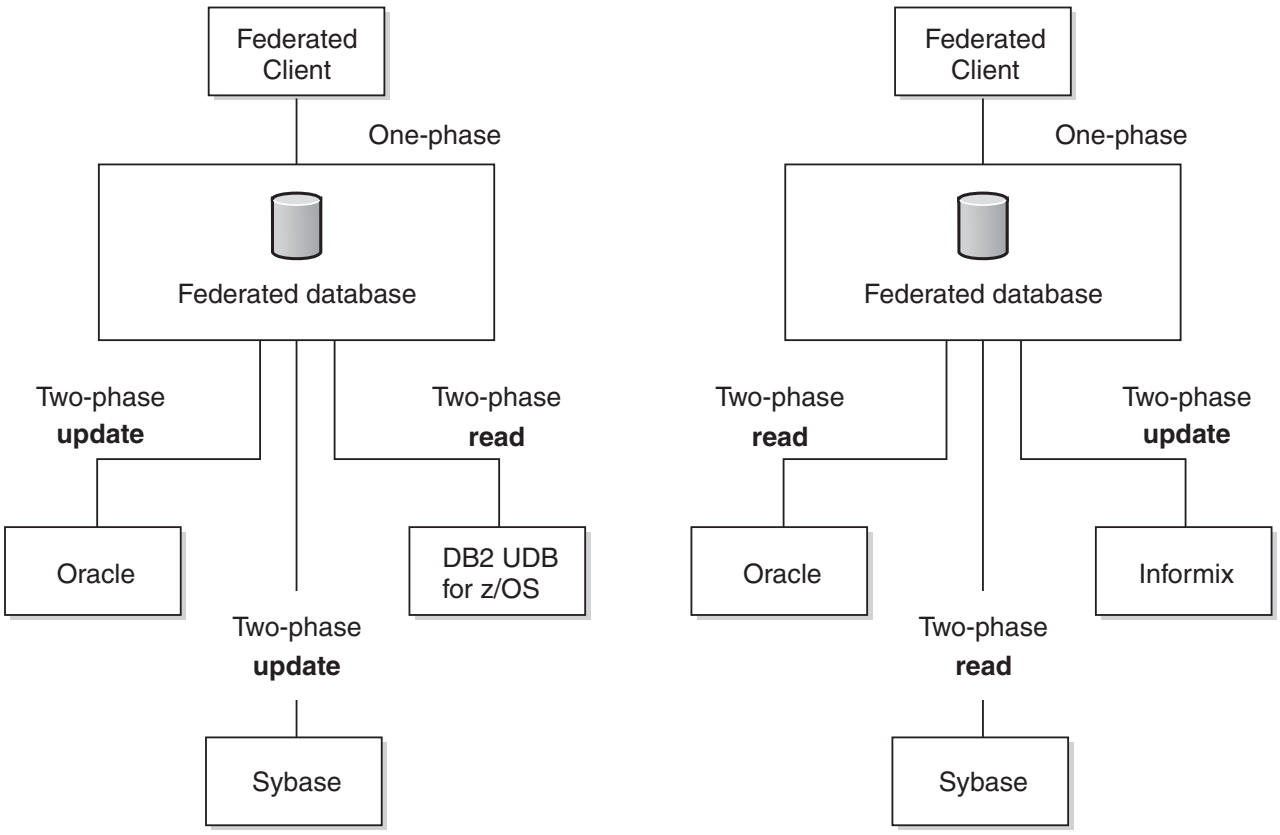


Figure 4. DB2 Type 1 connection with a federated server as the transaction manager

The following figure shows a DB2 Type 2 connection with a federated server functioning as the resource manager. In this configuration, all federated data sources must be supported for federated two-phase commit and enabled for federated two-phase commit.

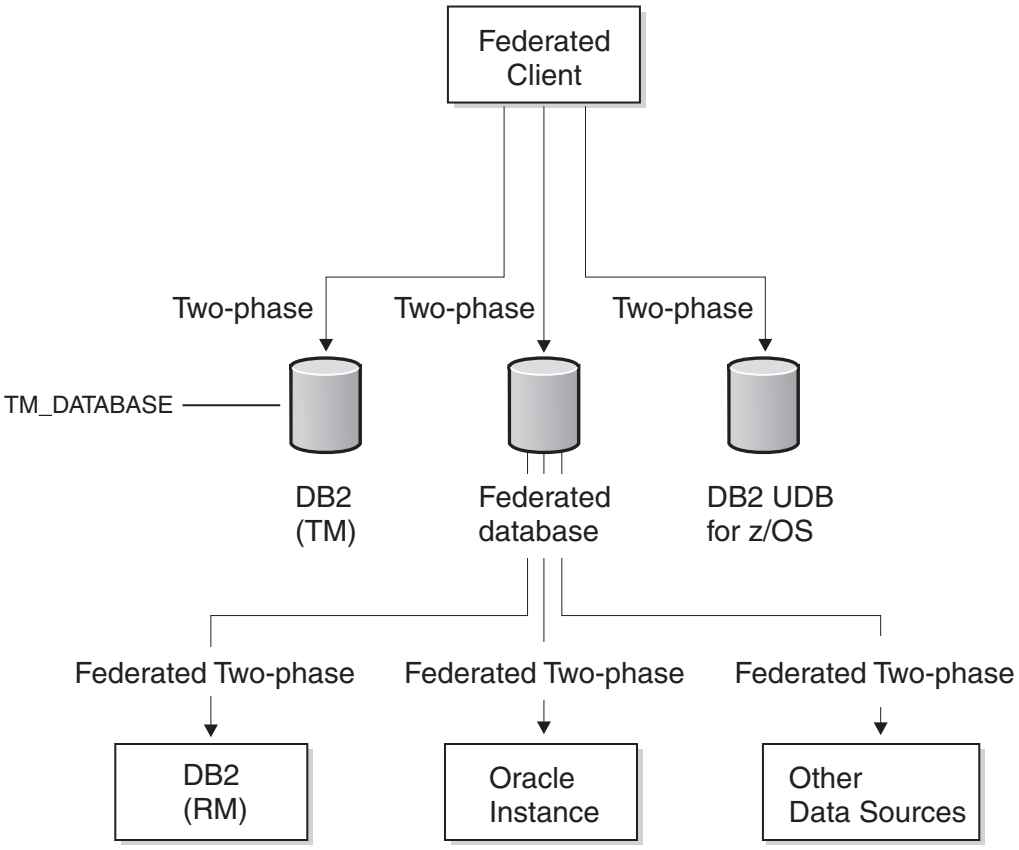


Figure 5. DB2 Type 2 connection with federated server as the resource manager

The following figure shows a DB2 Type 2 connection with a federated server functioning as the transaction manager.

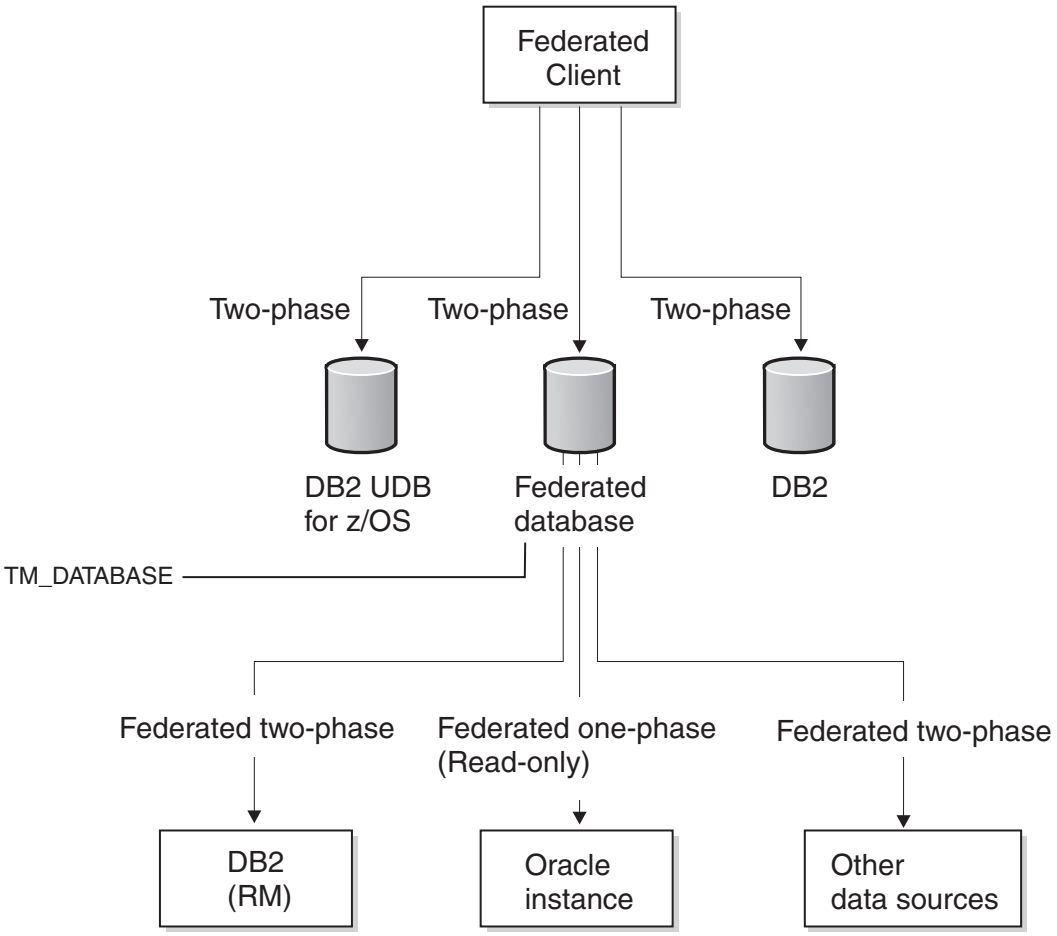


Figure 6. DB2 Type 2 connection with a federated server as the transaction manager

The following figure shows an XA connection with a federated server functioning as the resource manager.

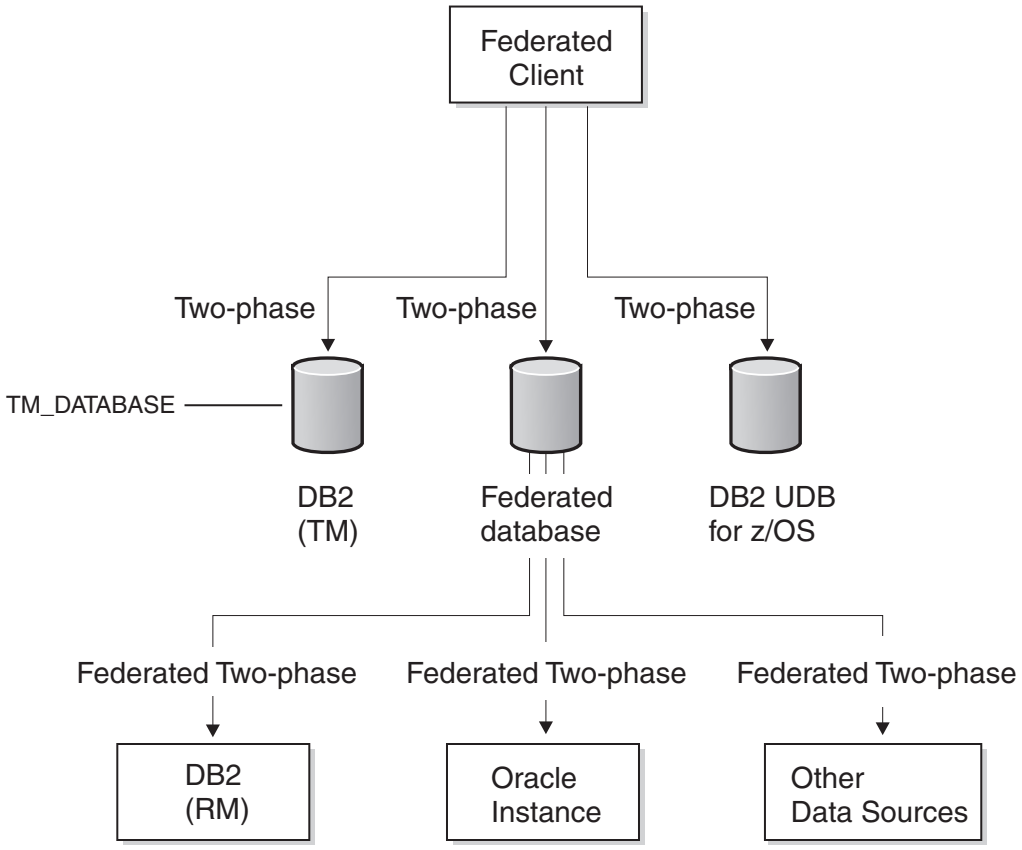


Figure 7. XA connection with federated server as the resource manager

How federated two-phase commit transactions are processed

The federated server maintains data consistency and atomicity of the data sources that it manages. The range of possible transactions depends on the type of connection and whether the federated server is the transaction manager or resource manager for the connection.

Atomicity is a database principal in which sets of operations are defined within indivisible transactions. This principal ensures that the database is consistent at all times because if a single operation within the indivisible transaction fails, the whole transaction fails rather than compromising data integrity due to a partial change.

For example, a transaction to transfer funds from one account to another involves withdrawing funds from the first account and adding funds to the second account. If only the withdrawal succeeds, the funds essentially cease to exist in the first account.

The federated server processes federated update requests under strict rules. A federated update is one of the following actions:

- A federated insert, update, or delete operation where the corresponding data source supports insert, update, or delete operations. For example, some data sources do not support update operations. Some data sources are read-only, in which case federated insert, update, or delete operations are not allowed.

- A successful pass-through operation inside a pass-through session.
- A transparent DDL operation, which is considered to be both a local update and a federated update because it performs database update both locally and remotely.
- A federated stored procedure with MODIFY SQL ACCESS.

Note: Do not use multiple paths to access the same data source within in a single transaction. Such a transaction can become deadlocked. That is, the transaction can hang. For example, do not use multiple federated servers to refer to the same data sources in the same transaction.

The following table lists what happens in a single distributed unit of work transaction including the type of connection, the type of commit, the federated server role in the transaction, what operations are allowed, and how transparent DDL can be used.

Table 8. What happens in a single distributed unit of work transaction

Type of commit	Type of connection	Federated server role	Operations	Transparent DDL
One-phase	DB2 Type 1 or XA local transaction	Sub-transaction manager. Also acts as the transaction coordinator, determines the transaction outcome, and delivers it to each participating resource manager.	One-phase commit and two-phase commit read operations are permitted. One one-phase data source can be updated as long as it is the only update in the transaction.	Allowed and managed according to one-phase commit data source rules. Each statement that is issued must be the only update in a one-phase commit transaction. Cannot coexist with other federated two-phase commit data source updates in the same transaction. It is strongly recommended that COMMIT or ROLLBACK statements be issued before and after transparent DDL transactions occur.
Two-phase	DB2 Type 1 or XA local transaction	Transaction manager. Also acts as the transaction coordinator, determines the transaction outcome, and delivers it to each participating resource manager.	One-phase commit and two-phase commit read operations are permitted. Multiple two-phase data sources can be updated.	Allowed and managed according to two-phase commit data source rules. Can coexist with other federated two-phase or one-phase commit data source updates in the same transaction.

Table 8. What happens in a single distributed unit of work transaction (continued)

Type of commit	Type of connection	Federated server role	Operations	Transparent DDL
One-phase	DB2 Type 2 or XA global transaction	Can be the transaction manager. If not the transaction manager, only relays the outcome from the external transaction coordinator to each participating resource manager.	One-phase commit and two-phase commit read operations are permitted. One-phase updates are not allowed except for DB2-coordinated transactions which can perform federated one-phase updates over a Distributed Relational Database Architecture™ (DRDA) two-phase inbound connection.	Allowed and managed according to one-phase commit data source rules. Each statement that is issued must be the only update in a one-phase commit transaction. Cannot coexist with other federated two-phase commit data source updates in the same transaction. It is strongly recommended that COMMIT or ROLLBACK statements be issued before and after transparent DDL transactions occur.
Two-phase	DB2 Type 2 or XA global transaction	Can be the transaction manager. If not the transaction manager, only relays the outcome from the external transaction coordinator to each participating resource manager.	One-phase commit and two-phase commit read operations are permitted. Multiple two-phase data sources can be updated.	Allowed and managed according to two-phase commit data source rules. Can coexist with other federated two-phase or one-phase commit data source updates in the same transaction.

How data consistency and atomicity are maintained

Federated servers attempt to ensure data consistency and maintain transaction atomicity of data sources.

Any conflict between an application synchronization point setting and the update capability of a target data source results in an error (SQL30090, reason code 18).

Local updates that include DDL made to the federated database cannot be mixed within the same transaction as an update to a federated one-phase data source. Transparent DDL

Using DDL and transparent DDL

Local updates that include DDL made to the federated database cannot be mixed within the same transaction as an update to a federated one-phase data source. Transparent DDL is an exception. For transparent DDL, both local updates and data source updates are allowed regardless of the type of connection and whether the data source is configured for one-phase or two-phase commit.

Transparent DDL creates a table on a remote data source and a nickname in the local federated database for the remote table. A federated server treats transparent DDL transactions as updates.

Transparent DDL provides the ability to create and modify remote tables through the DB2 database system, without the need to use pass-through sessions. The SQL statements for transparent DDL are CREATE TABLE, ALTER TABLE, and DROP TABLE. For example, a transparent DDL CREATE TABLE statement creates a remote table at the data source and a nickname for that table at the federated server. The statement contains a local update operation and a remote update operation.

Some data sources, such as Oracle, do not permit transparent DDL on a federated two-phase commit connection.

Enabling two-phase commit for federated transactions

To use federated two-phase commit for specific data sources, you must enable the associated federated servers. The enablement process involves preparing the federated server and modifying the data source server definition.

Before you begin

- When you enable federated two-phase commit for a data source, you increase the number of records that are written to both the federated server database log and the data source database log. Consider the impact this has on the administration and maintenance of these log files in order to ensure that they comply with your local policies.
- The data source that you want to use must be a supported federated two-phase commit data source.
- Federated two-phase commit is not supported in the massively parallel processing (MPP) environment.
- Federated two-phase commit is supported in the fenced environment for the following data sources:
  - DB2 family data sources support fenced mode and trusted mode.
  - Informix supports trusted mode.
  - Oracle supports fenced mode and trusted mode.
  - Sybase supports fenced mode.
  - MS SQL Server supports trusted mode.
- Federated two-phase commit is not supported in the DB2 pureScale environment.
- For DB2 for System i,version 5.3, and earlier and DB2 for z/OS data sources, ensure that the configuration parameter SPM_NAME is set to the default value, the server host name. SPM_NAME defaults to a variant of the first seven characters of the TCP/IP host name. DB2 for System i, version 5.4, and later does not require that you set SPM_NAME.

About this task

About this task

The DB2_TWO_PHASE_COMMIT server option enables two-phase commit for data sources. You register a data source server definition by using the CREATE SERVER statement. The value you set for DB2_TWO_PHASE_COMMIT persists for

all connections that are established under that server definition. You can change the value at any time by using the ALTER SERVER statement. After the CREATE SERVER or ALTER SERVER statement is successfully committed, the new setting is available for use on subsequent outbound connection requests.

Clients and application programs can use the SET SERVER OPTION to temporarily override the current value of the DB2_TWO_PHASE_COMMIT server option. The SET SERVER OPTION statement must be run immediately after the connection to the federated server database and before any connections are established to the remote data sources. The command is in effect only for the duration of the connection to the federated database. You cannot change the DB2_TWO_PHASE_COMMIT server option once the federated server has established a connection to the remote data source.

When you include the XA_OPEN_STRING_OPTIONS option in a CREATE SERVER statement, you can embed specialized information in the default XA_OPEN string. This embedded information can be any of the following kinds of information:

- Unique IDs for transactions in addition to what IBM InfoSphere Federation Server provides
- User-defined parameters about how transactions are handled
- A user-defined string to append to the XA_OPEN request

When an XA_OPEN call is made, usually at the beginning of the first transaction to a remote data source that uses two-phase commit, the wrapper appends the value of the user-defined string onto the default XA_OPEN string for the XA_OPEN call.

You can include both DB2_TWO_PHASE_COMMIT and XA_OPEN_STRING_OPTIONS in a CREATE SERVER, SET SERVER, or ALTER SERVER statement.

Procedure

1. Run the CREATE SERVER, ALTER SERVER, or SET SERVER statement with the DB2_TWO_PHASE_COMMIT option set to Y.
2. Optional: Run the CREATE SERVER, ALTER SERVER, or SET SERVER statement with the XA_OPEN_STRING_OPTIONS option.

Server option examples

This example shows how to set two-phase commit by using the CREATE SERVER statement:

```
CREATE  SERVER  Net8_Server  TYPE  ORACLE VERSION  8.1.7 WRAPPER  NET8
OPTIONS (DB2_TWO_PHASE_COMMIT  'Y');
```

This example shows how to disable two-phase commit by using the ALTER SERVER statement:

```
ALTER  SERVER  Net8_Server OPTIONS  (SET  DB2_TWO_PHASE_COMMIT  'N');
```

This example shows how to set an XA trace file to D:\Temp\sybase_xa.log for the Sybase wrapper using the ALTER SERVER statement and the XA_OPEN_STRING_OPTIONS server option:

```
 ALTER SERVER Ctlib_Server OPTIONS (ADD XA_OPEN_STRING_OPTIONS
'-LD:\Temp\sybase_xa.log');
```

This example shows how to temporarily disable two-phase commit by using the SET SERVER OPTION statement:

```
SET  SERVER  OPTION  DB2_TWO_PHASE_COMMIT TO 'N'  FOR SERVER  Net8_Server;
```

Data source requirements and configuration for federated two-phase commit transactions

Before enabling federated two-phase commit for a data source, you must ensure it is a supported data source.

Federated systems support two-phase commit operations with the following data sources:

- DB2 family data sources through the Distributed Relational Database Architecture (DRDA) protocol:
  - DB2 Universal Database for Linux, UNIX, and Windows, version 8.1 or later
  - DB2 Universal Database for z/OS, version 7.1 or later
  - DB2 Universal Database for System i, version 5.3 or later
- Informix IDS, version 7.31 or later, version 9.40 or later, version 10.0 or later
- Informix XPS, version 8.40 or later
- Microsoft SQL Server 2000 and Microsoft SQL Server 2005 for a federated server only on Windows
- Oracle, version 8.1.7 or later, with the XA library
- Sybase Adaptive Server Enterprise, version 12 or later, with the XA library for a federated server only on Windows

If you attempt to enable federated two-phase commit for an unsupported data source, you receive an SQL1881N error.

Configuring DRDA data sources

The federated server provides connectivity to DB2 data sources by using the open DRDA protocol. This support is equivalent to that provided by the DB2 Connect™ server.

Before you begin

In addition, for two-phase commit, the federated server interacts with each data source using the industry-standard XA model.

Before you begin

Restrictions:

- Not all DB2 data sources support XA over DRDA natively. For those that do not, such as DB2 for z/OS and DB2 for System i, the federated server uses the syncpoint manager (SPM). The syncpoint manager performs a mapping between the XA and non-XA two-phase-commit flows that all DB2 servers support. When federated two-phase commit access is provided by using the syncpoint manager, not all XA semantics are supported because of incompatibilities between federated support and the syncpoint manager. For example, transactions cannot be nested. All transactions must be committed or rolled back before starting a new transaction.

- Federated two-phase commit supports DB2 for z/OS, but DB2 for z/OS does not allow a SAVEPOINT statement to be issued in a federated two-phase commit transaction.
- In a DB2-coordinated transaction, a DB2 for z/OS client can perform a federated one-phase update over a DRDA two-phase inbound connection. However, such an update cannot be completed over an XA DRDA two-phase inbound connection. Nor can a mix of one-phase and two-phase updates over a DRDA two-phase inbound be completed.
- The XA_OPEN_STRING_OPTIONS server option is not supported for DRDA data sources. If you use the option, an SQL1881 error is returned.

Requirements:

- For those DB2 data sources that support XA by using SPM rather than by supporting XA natively, ensure that the SPM_NAME and SVCENAME parameters in the database manager configuration are properly set to their defaults.

About this task

Procedure

To configure a DRDA data source:

Procedure

Run the CREATE SERVER, ALTER SERVER, or SET SERVER statement with the DB2_TWO_PHASE_COMMIT option set to Y.
The DRDA wrapper automatically generates the following XA OPEN string for DRDA data sources:

```
DB=dbname,UID=uid,PWD=password,TPM=FDB2,HOLD_CURSOR=T
```

Configuring Oracle data sources

There are several requirements and restrictions for using Oracle data sources for federated two-phase commit.

Before you begin

Before you begin

Restrictions:

- Pass-through DDL and transparent DDL directed to Oracle both fail with SQL30090 reason code 21 (ORA-2089) Regular SQL submitted in pass-through sessions works.

Requirements:

- You must give the following privileges to all users that run two-phase commit transactions from the federated server:
  - grant select on dba_pending_transactions to USERID;
  - grant select on dba_2pc_pending to USERID;
  - grant force transaction to USERID;
- Optionally, you can also give the following privilege to users that run two-phase commit transactions from the federated server:
  - grant force any transaction to USERID;

- If you intend to run more than 10 two-phase commit transactions simultaneously, consider increasing the distributed_transactions parameter of your Oracle server found in the init.ora file.

About this task

Procedure

To configure an Oracle data source:

Procedure

1. Run the CREATE SERVER, ALTER SERVER, or SET SERVER statement with the DB2_TWO_PHASE_COMMIT option set to Y.

   The Oracle wrapper automatically creates the following XA OPEN string for Oracle data sources:

   ```
   Oracle_XA=Acc=Puid/password+SesTm=0+DB=dbname+SqlNet=dblink+Threads=true
   ```

   For example:

   ```
   XA_OPEN_STRING_OPTIONS '+LogDir=/home/user/directory+DbgFl=0x7'
   ```

2. Optional: Specify additional XA options by using the XA_OPEN_STRING_OPTIONS server option.

Configuring Informix data sources

There are several requirements and restrictions for using Informix data sources for federated two-phase commit.

Before you begin

Before you begin

Restrictions:

- You cannot access Informix nicknames with a mix of a two-phase commit server and a one-phase commit server in a single connection to a federated server.
- The WITH HOLD cursor option is not supported.
- The XA_OPEN_STRING_OPTIONS server option is not supported for Informix data sources.

Requirements:

- The Informix database must have logging enabled.
- The Informix XA library only allows one connection per thread. As a result, the federated server cannot access Informix data sources with multiple servers that are enabled for federated two-phase commit in a single connection. If an application needs to use multiple servers that are enabled for federated two-phase commit, complete the optional steps in the following procedure.

About this task

Procedure

To configure an Informix data source:

Procedure

1. Run the CREATE SERVER, ALTER SERVER, or SET SERVER statement with the DB2_TWO_PHASE_COMMIT option set to Y.

The Informix wrapper generates the following XA OPEN string automatically for Informix data sources:

```
DB=dbname;RM=rmname;CON=con;USER=user;PASSWD=password
```

2. Optional: If an application needs to use multiple servers that are enabled for federated two-phase commit, complete the following steps:
   a. Copy the Informix wrapper libraries: `libdb2informix.a`, `libdb2informixF.a`, and `libdb2informixU.a`.
   b. Define multiple instances of the Informix wrapper by specifying a different copy of the Informix wrapper libraries in the LIBRARY clause within the CREATE SERVER statement.
   c. Define each federated two-phase commit server for the different wrapper instances.

   For example:

```
CREATE WRAPPER wrapper1 library 'libdb2informix.a'
CREATE SERVER server1 type informix version 9.4 wrapper wrapper1 options
  (node 'inf1', dbname 'firstdb', db2_two_phase_commit 'Y');
CREATE WRAPPER wrapper2 library 'libdb2informix2.a'
CREATE SERVER server2 type informix version 9.4 wrapper wrapper2 options
  (node 'inf2', dbname 'seconddb', db2_two_phase_commit 'Y');
```

Configuring Microsoft SQL Server data sources

There are several requirements and restrictions for using Microsoft SQL Server data sources for federated two-phase commit.

Before you begin

Before you begin

Restrictions:

- For Microsoft SQL Server data sources, federated two-phase commit is only supported by IBM InfoSphere Federation Server installed on Windows.
- The DB2 isolation level is not propagated to the Microsoft SQL server.

Requirements:

- For federated two-phase commit to work with Microsoft SQL, an extra server option, XA_OPEN_STRING_OPTIONS, must be added to the server:

```
alter server S1 options(add xa_open_string_options
'RMRecoveryGuid=c200e360-38c5-11ce-ae62-08002b2b79ef');
```

  where `RMRecoveryGuid` = resource manager ID.

  The resource manager ID is available in the following location of the Microsoft SQL Server Registry:

```
[HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\MSSQLServer]
 "ResourceMgrID" = "{resource manager ID}"
```

About this task

Procedure

To configure a Microsoft SQL Server data source:

Procedure

1. Run the CREATE SERVER, ALTER SERVER, or SET SERVER statement with the DB2_TWO_PHASE_COMMIT option set to Y.

The Microsoft SQL Server wrapper automatically generates the following XA OPEN string for Microsoft SQL Server data sources:

```
TM=tmname
```

2. Optional: Specify additional XA options in addition to the required RMRRecovery Guid value by using the XA_OPEN_STRING_OPTIONS server option.

Configuring Sybase data sources

There are several requirements and restrictions for using Sybase data sources for federated two-phase commit.

Before you begin

Before you begin

Restrictions:

- For Sybase data sources, federated two-phase commit is only supported by IBM InfoSphere Federation Server installed on Windows.
- Pass-through DDL and transparent DDL directed to Sybase both fail with an SQL910N error. Regular SQL submitted in pass-through sessions works.

Requirements:

- The Sybase database administrator must have a license for distributed transaction management of Sybase Adaptive Server Enterprise (ASE) and must enable the feature with the following command in the isql tool:

  ```
  sp_configure 'enable dtm', 1
  ```

  Sybase ASE must be restarted for this parameter to take effect.
- The user name that is specified in an open string must have the dtm_tm_role in the corresponding Sybase ASE. The administrative user can assign this role with the following command in the isql tool:

  ```
  sp_role "grant", dtm_tm_role, user_name
  ```
- For a data source for Sybase Adaptive Server Enterprise (ASE) to act as a resource manager to a federated database, a logical resource manager (LRM) entry must exist in the `xa_config` file in the `$SYBASE/$SYBASE_OCS/config` directory (version 12 or later) that maps the resource manager name to the Sybase ASE name. Consult the Sybase ASE XA documentation for more information.

  The LRM name is used by the federated server in the XA OPEN string. The federated server uses the Sybase ASE node name for the LRM name.
- Ensure that the server name specified in the XA configuration file `xa_config` is present in the initialization file `sql.ini` in the `$SYBASE/ini` directory.

About this task

Procedure

To configure a Sybase data source:

Procedure

1. Install the Sybase XA library file libxadtm.dll on the federated server.
2. Before you use federated two-phase commit functions, create the following LRM entries in the `$SYBASE/$SYBASE_OCS/config/xa_config` file. If you do not

have write permissions for the xa_config file, create an xa_config file in another directory and set its absolute path in the XACONFIGFILE environment variable in the db2dj.ini file:

```
;one comment line is required
lrm=lrm_name
server=server_name
```

where *server_name* is an entry name in the $SYBASE/ini/sql.ini file.

3. The Sybase wrapper automatically creates the following default XA_OPEN string for Sybase data sources:

```
-Nrmname -Uuserid -Ppassword
```

If you need to specify other options for the XA_OPEN string, use the XA_OPEN_STRING_OPTIONS server option.

Recovering from federated two-phase commit problems

A federated system can recover from problems during two-phase commit with automatic resynchronization or manual recovery of indoubt transactions.

Resynchronization for federated systems

Federated two-phase commit includes an automatic process that attempts to handle errors during commit transactions.

Besides errors on the federated server, a federated environment increases the potential for errors that result from network, communications, or data source failures.

To ensure data integrity, the federated server handles these errors during the federated two-phase commit process:

First phase error
If a database communicates that it has failed to prepare to commit a unit of work, the federated server rolls back the unit of work during the second phase of the commit process. During the second phase, the federated server sends a rollback message to all participating data sources that are waiting for the transaction outcome.

Second phase error
Error handling at this phase depends upon whether the second phase commits or rolls back the transaction. The second phase only rolls back the transaction if the first phase encountered an error.

If one of the participating data sources fails to commit or rollback the unit of work, possibly due to a communications failure, the federated server retries the commit or rollback through a process called resynchronization. Resynchronization is initiated and managed by the federated server automatically. The calling application is informed that the commit was successful through the SQLCA (SQL communications area) if the application connects to the federated server through a two-phase commit connection. The calling application is disconnected from the federated server if the application connects to the federated server through a one-phase commit connection.

Most data sources are not capable of initiating resynchronization if a failure occurs in a federated system. The federated server initiates the resynchronization process.

In certain circumstances, a federated server might fail during transaction processing, for example, due to a power failure. Resynchronization usually resolves any distributed unit of work transactions without intervention.

Resynchronization attempts to complete all indoubt transactions. As part of normal resynchronization, the resynchronization agent connects to the resource manager database for a transaction and issues a commit or rollback decision. The federated transaction manager then propagates that decision to the data sources that participated in the distributed unit of work transaction.

Manually recovering indoubt transactions

If you cannot wait for resynchronization to automatically resolve indoubt transactions, you can resolve the indoubt transactions manually. This process is sometimes referred to as heuristic processing.

For example, the communications link between the external transaction manager and a federated transaction manager fails in the middle of a transaction. If you have enough information about the transaction, you can free resources on the federated server and on remote data sources by rolling back the transaction from the federated server.

Use heuristic processing only when you know the reason for the transaction failure, and you must free locked resources immediately. In most situations, let the automated resynchronization recover transactions. There are multiple layers of transaction management in a federated system. Recovering transactions heuristically is a complex and potentially risky process.

There are three basic ways to perform heuristic processing:

- The LIST INDOUBT TRANSACTIONS command

  You can use this command line command to perform heuristic processing.
- The Indoubt Transaction Manager window

  You can use this graphical user interface tool to perform heuristic processing.
- Heuristic APIs

  You can use these APIs within your applications to perform heuristic processing.

The specific operations and tasks that you use to perform heuristic processing varies, depending on the circumstances of the error.

In federated systems, when a heuristic processing request is sent to a federated transaction manager, the resulting decision to commit or roll back must be compatible with the actual status of the indoubt transaction on the federated server. Otherwise, an error message is returned.

The status of federated two-phase commit in doubt transactions is slightly different than basic DB2 two-phase commit in doubt transactions:

- A status of (d) means that the transaction is missing commit acknowledgement from one or more federated data sources.
- A status of (b) means that the transaction is missing rollback acknowledgement from one or more federated data sources.

If you cannot successfully commit or rollback a transaction with status (d) or (b), you can specify that the transactions be forgotten by using option (f). However, when you use the (f) option, all records of the transaction are erased from the

federated server and you must manually clean up any remaining synchronization problems on the involved data sources. Only use the (f) option when it is absolutely necessary, such as when a remote server crashes or connections to remote servers are dropped and there is an urgent requirement to free up resources and use it with caution.

Note: Because the (d) and (b) statuses are new for WebSphere® Federation Server v9.1, down-level federated clients cannot support them. If you use a down-level client to manually recover indoubt transactions, the (d) and (b) statuses are mapped to status (m), instead, which is not an accurate value. To prevent inaccurate information when you are manually recovering indoubt transactions, be sure to use a v9.1 federated client. By default, the computer that runs WebSphere Federated Server v9.1 includes the v9.1 federated client.

Tracing distributed unit of work transaction states across data sources

If you decide to resolve indoubt transactions manually instead of letting synchronization resolve them automatically, tracing transactions in the federated system is essential. When you are tracing an indoubt distributed unit of work transaction, the only way to determine the data source or data sources that failed is to capture the XID for the failed transaction.

You must look for that XID in the data source database managers for all of the data sources that a federated server might have accessed as a part of the distributed unit of work transaction.

To determine the identifier and state of each transaction that is involved in the distributed unit of work that you want to trace, issue the LIST INDOUBT TRANSACTIONS command in your application database, the federated database, and any data sources in the distributed unit of work transaction.

Note: Each data source that supports the two-phase commit protocol might use a different command. Search for the string XID in the command documentation for your specific data source.

The federated transaction manager generates an XID in hexadecimal format during transaction processing. This XID begins with the format identifier of F2PC, which is the number 46325243 in hexadecimal format. The federated transaction manager sends the XID to the data sources. Before a federated server sends an XID to a data source, however, the federated server changes the XID so that it conforms to the XID format of that data source. These modifications include updating the branch qualifier length section of the XID and adding the branch qualifier section of the XID.

You might need to compare XIDs across several data sources, so you must know which part of the XID that you can accurately compare across your environment when you trace an XID.

For example, the transaction manager is a DB2 database. When you issue the LIST INDOUBT TRANSACTIONS command at the database, a string in hexadecimal representation is returned for the transaction XID that is similar to the following one:

```
463250430000019 00000000473931453346313 5 2E47453934000000 000000000000E80000
```

This string is actually composed of several distinct parts:

Format ID	Transaction identifier length	Branch qualifier length	Transaction identifier
46325043	00000019	00000000	4739314533463135 2E47453934000000 000000000000E800 00

The values listed in the string are hexadecimal. For example, the transaction identifier length (hexadecimal 19) represents the decimal value 25.

If that same XID is passed from a federated server that acts as a federated transaction manager to a data source that acts as a resource manager, the XID string is appended. For example, the XID that is passed to a resource manager for a DB2 database system for Windows data source changes to the following format:

```
463252430000019 00000001473931453346313 5 2E47453934000000  000000000000E8000001
```

Here is that same changed XID string broken down into the standard parts:

Format ID	TID length	Branch qualifier length	Transaction identifier	Branch qualifier
46325043	00000019	00000001	4739314533463135 2E47453934000000 000000000000E800 00	01

The branch qualifier length now has a 1, and the branch qualifier section is added. However, the transaction identifier field has not changed. You can still trace the XID across different data sources by limiting your search string input to the transaction identifier section.

Troubleshooting federated two-phase commit issues

Troubleshooting federated two-phase commit issues is often specific to the data source that is causing the issue.

Applications must handle error codes that indicate that transactions have timed out, specifically -913 and -918 error codes in addition to checking for -911 error codes.

Oracle data source troubleshooting

Try the following methods to troubleshoot issues with Oracle data sources.

- To log information:

  ```
  db2 "alter server ora1 options (add XA_OPEN_STRING_OPTIONS
  '+LogDir=C:\temp+DbgFl=0x7')
  ```

  Where *C:\temp* is the full path to the location where you want the log file created. Logging information can significantly slow performance, so log information only when you are troubleshooting issues.
- To display trace information, add the following lines to the Oracle client sqlnet.ora file:

  ```
  TRACE_LEVEL_CLIENT=16
  TRACE_DIRECTORY_CLIENT=C:\temp
  ```

You can also set TRACE_LEVEL_CLIENT to a lower number, such as 4 or 8. Displaying trace information can significantly slow performance, so enable trace level information only when you are troubleshooting issues.

- To list pending transactions exist on the Oracle data source:

```
select * from dba_pending_transactions where formatid=1177702467;
select * from dba_2pc_pending;
```

  To list the state of a transaction and the ID of a transaction:

```
select A.STATE, A.LOCAL_TRAN_ID, A.FAIL_TIME, A.GLOBAL_TRAN_ID,
B.FORMATID || '.' || B.GLOBALID || '.' || b.BRANCHID as fmt_xid
from dba_2pc_pending A, dba_pending_transactions B
where A.GLOBAL_TRAN_ID = B.FORMATID || '.' ||
B.GLOBALID and STATE='prepared' and B.FORMATID=1177702467;
```

- To resolve transactions manually:

```
rollback force '4.31.157818';
commit force '10.24.154537'
```

  where '4.31.157818' is the Oracle field A.LOCAL_TRAN_ID that corresponds to the XID in GLOBAL_TRAN_ID.

Sybase data source troubleshooting

Try the following methods to troubleshoot issues with Sybase data sources.

- Check the Sybase XA log file `syb_xa_log` located in the `$SYBASE` directory.
- Use the db2diag tool to check the `db2diag.log` file.
- To check a transaction on the Sybase server:

```
$ isql -Uuser_name -Ppassword -Sserver_name
1> sp_transactions
2> go
```

  If you find an invalid or unnecessary transaction, ask the Sybase administrator to delete the transaction.

Federated two-phase commit performance

Data sources that are configured for two-phase commit transactions incur a performance penalty when you compare them to data sources that are configured for one-phase commit transactions.

When a data source uses two-phase commit, the federated server acts as a coordinator, ensuring that all participants are correctly synchronized. This coordination is achieved by additional logging on the federated server and additional communication between data sources. As such, a federated transaction accessing a data source for two-phase commit requires more processing than a transaction that accesses a data source for one-phase commit. Consequently, a data source must be enabled for two-phase commit only when the federated transactions require two-phase commit.

A federated transaction that only requires one-phase commit, such as an update to a single site, but which is run on a data source for two-phase commit is likely to incur a performance penalty when you compare it to the same transaction that is run on a data source without two-phase commit.

A transaction under two-phase commit control incurs the following additional processing regardless of whether or not the transaction requires two-phase commit:

1. All transactions require additional database log writes to the federated server log file.

   The additional log writes enable the federated server to track the data sources in the transaction in order to coordinate subsequent commit and rollback operations.
2. All transactions require additional database communication between the federated server and the data source.
3. Insert, update, and delete operations require one or more additional database log writes to the remote data source.

Most of the additional processing occurs at the transaction boundaries and is not influenced by the contents within the transaction. As a result, the percentage increase in elapsed time for a short transaction is greater than that for a long transaction.

Concurrent transactions cause the federated server to write multiple log records from the log buffer to the log file in a single write operation. Consequently, applications running federated two-phase commit transactions concurrently incur less overhead than applications running the same transactions serially.

Improving federated two-phase commit performance

You can take steps to improve the performance of transactions in a federated two-phase commit configuration.

Consider the following possible configurations:

- In order to reduce the time spent writing the additional log records to the federated server, place the federated database log files on a device that is capable of fast write transactions, preferably a device that has a write cache. Generally, the federated server log write transactions cause most of the additional processing time. Correct placement of the log files is likely to improve the two-phase commit performance. This improvement is particularly true for read-only transactions. The placement of the federated server log files is controlled by the NEWLOGPATH database configuration parameter.
- For insert, update, and delete transactions, place the remote data source log files on media that are capable of fast writes.
- To reduce the overhead introduced by additional XA messages that are sent between the federated server and data sources:
  - Place the federated server on the same computer as one of the two-phase commit data sources.
  - If you cannot place the federated server on the same computer as a data source, increase the network speed and reduce latency between the federated server and the data sources to help improve performance.

For applications, only enable two-phase commit for a data source when at least one transaction in the application requires two-phase commit. For each server, set the default value for DB2_TWO_PHASE_COMMIT to N and use the SET SERVER OPTION statement to enable two-phase commit within the applications that specifically require two-phase commit.

Chapter 10. Inserting, updating, and deleting data in a federated system

During the development of your federated environment, whether you modify remote information or move data among data sources, you must insert, update, and delete data on your data sources.

Before you modify data on a remote data source, ensure that you have the appropriate authorization privileges to issue the INSERT, UPDATE and DELETE statements on the nickname. You should also have an understanding of referential integrity, preserving statement atomicity, and assignment semantics.

Authorization privileges for INSERT, UPDATE, and DELETE statements

The privileges required to issue INSERT, UPDATE, and DELETE statements on nicknames are similar to the privileges required to issue these same statements on tables. In addition, you must hold adequate privileges on the data source to perform select, insert, update, and delete operations on the underlying object.

You can grant or revoke SELECT, INSERT, UPDATE, and DELETE privileges on a nickname.

However, granting or revoking privileges on a nickname does not grant or revoke privileges at the data source. At the data source, the privileges must be granted or revoked for the REMOTE_AUTHID specified in the user mapping at the federated server.

The privileges held by the authorization ID of the statement must include the necessary privileges on the nickname (for the federated database to accept the request). The user ID at the data source that is mapped to the authorization ID (through a user mapping) must have the necessary privileges on the underlying table object (for the data source to accept the request).

When a query is submitted to the federated database, the authorization privileges on the nickname in the query are checked. The authorization requirements of the data source object referenced by the nickname are only applied when the query is actually processed. If you do not have SELECT privilege on the nickname, then you can not select from the data source object that the nickname refers to.

Likewise, just because you have UPDATE privilege on the nickname does not mean you will automatically be authorized to update the data source object that the nickname represents. Passing the privileges checking at the federated server does not imply that you will pass the privilege checking at the remote data source. Through user mappings, a federated server authorization ID is mapped to the data source user ID. The privilege checking is enforced at the data source.

Federated system INSERT, UPDATE, and DELETE restrictions

Some restrictions apply to using INSERT, UPDATE, or DELETE statements in a federated system.

The following restrictions apply to updates on nicknames:

- A data source object that is read-only, such as a JOIN view, cannot be updated
- You cannot perform insert, update, and delete operations on federated views created with UNION ALL statements. Federated views created with UNION ALL statements are read-only views.
- A federated insert, update, or delete operation, or a call to federated procedure with an SQL data access indication of MODIFIES SQL DATA is invalid in a function, a data-change-table-reference, a compound statement that specifies ATOMIC (with the exception of the DB2 for Linux, UNIX, and Windows data source), a trigger, and an application execution environment where any of the following is true:
  - SAVEPOINT is in effect (with the exception of the DB2 for Linux, UNIX, and Windows data source)
  - Scrollable cursor is used
  - Target view contains multiple tables or nicknames

Unsupported data sources

The federated system does not support insert, update and delete operations against nicknames on nonrelational data sources.

The data sources that federation does not support include:

- BioRS
- Excel
- Table-structured files
- Web services
- XML

Referential integrity in a federated system

In a federated system, the federated database does not enforce referential integrity among data sources.

However, referential integrity constraints at a data source can affect nickname updates. Suppose that you need to insert data that is on the federated server into a nickname. When the federated server sends the insert to the data source, it violates a referential integrity constraint at that data source. The federated server maps the resulting error to a federated error.

Applications are responsible for referential integrity between data sources.

INSERT, UPDATE, and DELETE statements and large objects (LOBs)

You can perform read operations on remote LOBs on federated systems. Write operations on LOBs are supported for some data sources.

With federation you can perform read operations on LOBs that are located in any relational data source. You can perform write operations on LOBs that are located in the following data sources:

- Oracle, using the NET8 wrapper
- DB2 for z/OS, DB2 for System i, and DB2 Database for Linux, UNIX, and Windows, using the DRDA wrapper
- Teradata, using the Teradata wrapper

Under certain conditions, you can perform write operations on LOBs in other data sources by altering the nickname column type to a VARCHAR.

Preserving statement atomicity in a federated system

During update operations, federated systems always attempt to keep data in an atomic state at the completion of a DML statement. When data is in an atomic state, it is guaranteed to either be successfully processed or to remain unchanged.

When a client or application issues an INSERT, UPDATE, or DELETE statement on a nickname, a federated server internally processes that statement either as a single DML statement, or as a series of multiple DML statements. If a federated server must send multiple DML statements to a target data source for processing, it is possible that data atomicity might be compromised. To prevent compromising data atomicity, federated systems use data source savepoint APIs to monitor the series of multiple DML statements.

Some data sources do not externalize savepoint APIs that the federated system can use. Under these circumstances, federated INSERT, UPDATE, or DELETE statements are run without the protection of savepoint APIs.

When an error occurs during federated insert, update, delete transactions, partial update results can occur at the data sources. To correct inconsistency problems, a federated system automatically performs an internal transaction rollback before it returns an SQLCODE error to the applications.

The following data sources do not externalize savepoint APIs that the federated server can use:

- DB2 for System i
- DB2 for VM and VSE
- Informix
- JDBC
- Microsoft SQL Server
- ODBC
- Teradata

When an entire insert, update, or delete transaction is pushed down to the data source for processing, the federated server assumes that the data source will preserve the statement atomicity if an error occurs. When only part of the insert, update, or delete transaction is pushed down to the data source for processing, the entire transaction is rolled back if an error occurs.

Modifying data in a federated system

You can modify data in a federated system by inserting, updating, and deleting data in data source objects.

Inserting data into data source objects

To insert data into data sources, use the nicknames for the data source objects in the INSERT statement.

Before you begin

To insert data using a nickname, all of the following privileges must be true:

- The privileges held by the authorization ID of the statement must include the INSERT privilege on the nickname (for the federated database to accept the request).
- The user ID at the data source must have the INSERT privilege on the underlying table object (for the data source to accept the request).
- The user ID at the data source must be mapped to the authorization ID at the federated server through a user mapping.

About this task

Restrictions

Federation does not support INSERT operations with nonrelational data sources.

Procedure

Issue the INSERT statement to insert data into data source objects.

Example

An Informix table consists of two columns. The first column contains INTEGER data and the second column contains VARCHAR data (up to 20 characters). The nickname *infx_table_nn* is registered with the federated server for the Informix table.

You can issue INSERT, UPDATE, and DELETE statements on the Informix table using the *infx_table_nn* nickname. The following statement inserts a new row of information into the Informix table:

```
INSERT INTO db2user1.infx_table_nn VALUES(1,'Walter')
```

Updating data in data source objects

To update data into data sources, use the nicknames for the data source objects in the UPDATE statement.

Before you begin

To update data using a nickname, all of the following privileges must be true:

- The privileges held by the authorization ID of the statement must include the UPDATE privilege on the nickname (for the federated database to accept the request)
- The user ID at the data source must have the UPDATE privilege on the underlying table object (for the data source to accept the request)
- The user ID at the data source must be mapped to the authorization ID at the federated server through a user mapping.

About this task

Restrictions

Federation does not support UPDATE operations with some data sources, see "Federated system INSERT, UPDATE, and DELETE restrictions" on page 120.

To update data into data source objects, issue the UPDATE statement.

Example: An Informix table consists of two columns. The first column contains INTEGER data and the second column contains VARCHAR data (up to 20 characters). The nickname *infx_table_nn* is registered with the federated server for the Informix table.

You can issue INSERT, UPDATE, and DELETE statements on the Informix table using the *infx_table_nn* nickname. The following statement updates a row of information in the Informix table:

```
UPDATE db2user1.infx_table_nn SET c2='Bill' WHERE c1=2
```

Deleting data from data source objects

To delete data from data sources, use the nicknames for the data source objects in the DELETE statement.

Before you begin

To delete data using a nickname, all of the following privileges must be true:

- The privileges held by the authorization ID of the statement must include the DELETE privilege on the nickname (for the federated database to accept the request)
- The user ID at the data source must have the DELETE privilege on the underlying table object (for the data source to accept the request)
- The user ID at the data source must be mapped to the authorization ID at the federated server through a user mapping.

About this task

Restrictions

Federation does not support DELETE operations with some data sources, see "Federated system INSERT, UPDATE, and DELETE restrictions" on page 120.

Procedure

Issue the DELETE statement to delete data from data source objects.

Example

An Informix table consists of two columns. The first column contains INTEGER data and the second column contains VARCHAR data (up to 20 characters). The nickname *infx_table_nn* is registered with the federated server for the Informix table.

You can issue INSERT, UPDATE, and DELETE statements on the Informix table using the *infx_table_nn* nickname. The following statement deletes a row of information in the Informix table:

```
DELETE FROM infx_table_nn WHERE c1=3
```

Assignment semantics in a federated system

When you assign data to a nickname column, the data type might change based on the assignment rules that the federated system uses. You should understand the assignment rules so you get the results that you expect.

The rules for determining the target data type of an assignment to a nickname column are:

- Determine the local source type: The local source type is determined by the local column type or the local result type of the expressions. If the source is constant, the local source type is the same as the type of the constant.
- Determine the target type:
  - If the assignment source has no type, such as parameter markers and NULLs, then the target type is MIN(local_target_type, remote_target_type), where local_target_type is the updated column local data type and remote_target_type is the updated column data source data type. The remote_target_type refers to the default forward type mapping type of the remote target column's data type.
  - If the assignment source is not NULL or parameter markers, then the target type is MIN(local_target_type, remote_target_type, local_source_type).

The definition of MIN(type1, type2)

- Type1 and type2 are not exactly the same.
- MIN(type1,type2) = MIN(type2, type1)
- MIN(type1, type2) = remote_target_type(local_target_type), when MIN(type1, type2) = DECIMAL(0,0)
- BLOB is only compatible with BLOB, so MIN(BLOB(x), BLOB(y))=BLOB(z) where z=min(x,y)
- TIME and DATE data types are not compatible.
- Datetime types and character strings are compatible.
- In Unicode databases, character strings and graphic strings are compatible.

The following tables list the minimum of two data types for numeric, character string, graphic string, and date and time data types.

Table 9. Numeric data types

type1	type2	MIN(type1, type2)
SMALLINT	SMALLINT or INTEGER or BIGINT or REAL or DOUBLE	SMALLINT
INTEGER	BIGINT or REAL or DOUBLE	INTEGER
BIGINT	REAL or DOUBLE	BIGINT
REAL	DOUBLE	REAL
DECIMAL(w,x)	SMALLINT	DECIMAL(p,0) where p=w-x, if p<5; SMALLINT, otherwise
DECIMAL(w,x)	INTEGER	DECIMAL(p,0) where p=w-x, if p<11; INTEGER, otherwise
DECIMAL(w,x)	BIGINT	DECIMAL(p,0) where p=w-x, if p<19; BIGINT, otherwise

Table 9. Numeric data types (continued)

type1	type2	MIN(type1, type2)
DECIMAL(w,x)	DECIMAL(y,z)	DECIMAL(p,s) where p=min(w,y)+min(w-x,y-z), s=min(x,z)
DECIMAL(w,x)	DOUBLE or REAL	DECIMAL(w,x)

The following table lists the minimum of two data types for character string data types.

Table 10. Character string data types

type1	type2	MIN(type1, type2)
CHAR(x)	CHAR(y) or VARCHAR(y) or LONG VARCHAR or CLOB(y)	CHAR(z) where z=min(x,y)
VARCHAR(x)	VARCHAR(y) or LONG VARCHAR or CLOB(y)	VARCHAR(z) where z=min(x,y)
LONG VARCHAR	CLOB(y)	LONG VARCHAR where x>32700, CLOB(x) where x<=32700
CLOB(x)	CLOB(y)	CLOB(z) where z=min(x,y)

The following table lists the minimum of two data types for graphic string data types.

Table 11. Graphic string data types

type1	type2	MIN(type1, type2)
GRAPHIC(x)	GRAPHIC(y) or VARGRAPHIC(y) or LONG VARGRAPHIC or DBCLOB(y)	GRAPHIC(z) where z=min(x,y)
VARGRAPHIC(x)	VARGRAPHIC(y) or LONG VARGRAPHIC or DBCLOB(y)	VARGRAPHIC(z) where z=min(x,y)
LONG VARGRAPHIC	DBCLOB(y)	LONG VARGRAPHIC where x>32700, DBCLOB(x) where x<=32700
DBCLOB(x)	DBCLOB(y)	DBCLOB(z) where z=min(x,y)

The following table lists the minimum of two data types for data and time data types.

Table 12. Date and time data types

type1	type2	MIN(type1, type2)
DATE	TIMESTAMP	DATE
TIME	TIMESTAMP	TIME

If the data of data type CHAR you are inserting is shorter than the target length, the data source pads the rest of the column.

If you are inserting data of DATE or TIME data type into a remote column of TIMESTAMP data type, the data source pads the rest of the column.

Assignment semantics in a federated system - examples

The following table shows several examples of the application of federated assignment semantics in queries given a local type and remote type.

Table 13. Examples of assignment semantics

Local type	Remote type	Your query	Generated remote query
FLOAT	INTEGER	set c1=123.23	set c1=INTEGER(123.23)
INTEGER	FLOAT	set c1=123.23	set c1=INTEGER(123.23)
FLOAT	INTEGER	set c1=123	set c1=123
CHAR(10)	CHAR(20)	set c1='123'	set c1='123' ('123' has a type VARCHAR(3) and it's the shortest of all)
CHAR(10)	CHAR(20)	set c1=char23col	set c1=CHAR(char23col, 10)
CHAR(10)	CHAR(20)	set c1=expr1	• set c1=expr1 -- if expr1 returns char(n) n<= 10 • set c1=CHAR(expr1, 10) if expr1 returns char(n) n>10
TIMESTAMP	DATE	set c1= current timestamp	set c1=DATE(current timestamp)

Data source restrictions on data type values

For some data types, the federated server supports a wider range of values than the data source.

When you use these data types, use the values that the only data source supports. If you insert a value that is outside the supported range, the data source either converts the value or returns an error.

If you use a predicate that contains a value that is outside the supported range and the predicate is evaluated at the data source, either of the following results can occur:

- The predicate can return a different result than the same predicate on the federated server.
- The predicate can result in an error.

Times and timestamps with 24 in the hours field

If you use a predicate with a time or timestamp that contains 24 in the hours field, you might receive an error. To prevent problems, you can convert these times or timestamps. You can use a statement similar to the following UPDATE statement:

```
UPDATE my_table SET timestamp_col = timestamp_col + 0 SECONDS;
```

This update changes the time field to 00:00:00 and increments the date by one day.

The federated server allows times and timestamps to contain 24 in the hours field. The distinction between a timestamp with 24 hours and a timestamp with 00 hours is that they are unequal in comparisons but equal in arithmetic.

Example

- The following predicate returns false:

```
2008-05-14-24:00:00.000000 = 2008-05-15-00:00:00.000000
```

- The following predicate returns true:

```
(2008-05-14-24:00:00.000000 + 0 SECONDS) =
 (2008-05-15-00:00:00.000000 + 0 SECONDS)
```

Some data sources such as Oracle and Sybase do not allow times or timestamps with 24 in the hours field.

Empty strings on Oracle

You can avoid empty string problems by not using empty strings in applications that use Oracle nicknames. Instead you can use the NULL value or a string consisting of a single blank (' ').

In VARCHAR columns on a federated server that is not VARCHAR2 compatible, a distinction is made between the empty string and NULL values, which results in different behaviors between the local tables and nicknames. Whereas, in VARCHAR columns on a remote data source that is VARCHAR2 compatible, the empty string and NULL values are considered equivalent. If an empty string ('') is inserted into a VARCHAR column, it is converted into a NULL value.

In the following examples, MY_TABLE is a local table in the federated database and MY_NICK is a nickname on a remote table in an Oracle database:

Example 1

In this example, an empty string is inserted into MY_TABLE and MY_NICK, which both contain a NOT NULL column named NOT_NULL_COL. The INSERT statement on MY_TABLE succeeds but the INSERT on MY_NICK fails with an error because the empty string is converted to a NULL value, which conflicts with the NOT NULL constraint.

```
INSERT INTO my_table(not_null_col) VALUES('');
INSERT INTO my_nick(not_null_col) VALUES('');
```

Example 2

In this example, both the federated and remote tables are initially empty. The first SELECT statement returns no rows because the federated server distinguishes between the empty string and NULL values but the second SELECT statement returns one row because during the insert, the Oracle database converts the empty string to a NULL value.

```
INSERT INTO my_table(my_col) VALUES('');
SELECT * FROM my_table WHERE my_col IS NULL;

INSERT INTO my_nick(my_col) VALUES('');
SELECT * FROM my_nick WHERE my_col IS NULL;
```

Federated updates with application savepoints

Application savepoints in a federated system help you control the work that is performed by a subset of SQL statements in a transaction.

You can use multiple savepoints within a single transaction. A subset of SQL statements that is grouped under a savepoint can be treated as a unit. You can logically divide a transaction into a single level or into nested levels of savepoint units.

After a savepoint is set within your application, you can later release the savepoint or roll back the work that is performed since you set the savepoint. For example, if an individual SQL statement within a savepoint fails, the unit as a whole remains intact. You can either:

- Undo all of the SQL statements that ran within the savepoint by rolling back to the savepoint.
- Release the savepoint when you no longer need the system to maintain the savepoint.

You can perform federated updates with application savepoints operations on DB2 Database for Linux, UNIX, and Windows.

Federated updates with application savepoints - examples

This example of application savepoints illustrates the effect of rollback operations within nested savepoints.

Example: Create a nickname named NN_DEPT for table DEPARTMENT in the schema DEPTDATA on the server DB2_SERVER. The nickname NN_DEPT contains the columns DEPT_NO, DEPT_NAME, and MANAGER_NO.

```
CREATE NICKNAME NN_DEPT FOR DB2_SERVER.DEPTDATA.DEPARTMENT;
```

Insert a row before you create the savepoint SP1. Insert another row before you create the savepoint SP2. Insert two more rows before you create the savepoint SP3. Insert a fifth row.

```
INSERT INTO NN_DEPT VALUES ('A10', 'SALES', 'ADAM');
SAVEPOINT SP1 ON ROLLBACK RETAIN CURSORS;

INSERT INTO NN_DEPT VALUES ('B10', 'MARKETING', 'BRIAN');
SAVEPOINT SP2 ON ROLLBACK RETAIN CURSORS;

INSERT INTO NN_DEPT VALUES ('C10', 'ENGINEERING', 'CINDY');
INSERT INTO NN_DEPT VALUES ('C20', 'TESTING', 'DOUG');
SAVEPOINT SP3 ON ROLLBACK RETAIN CURSORS;

INSERT INTO NN_DEPT VALUES ('D10', 'SUPPORT', 'EMILY');
```

The NN_DEPT nickname now contains five rows with departments A10, B10, C10, C20, and D10. You want to roll back some of these rows. The following examples describe the results of rolling back to each savepoint:

- `ROLLBACK TO SAVEPOINT SP3;`

  The last row with department D10 is no longer in the NN_DEPT nickname. The rows that you inserted before you created the savepoint SP3 (A10, B10, C10, C20) still exist in the NN_DEPT nickname.
- `ROLLBACK TO SAVEPOINT SP2;`

  The rows with departments C10, C20, D10 are no longer in the NN_DEPT nickname. The rows that you inserted before you created the savepoint SP2 (A10, B10) still exist in the NN_DEPT nickname.
- `ROLLBACK TO SAVEPOINT SP1;`

  The rows with departments B10, C10, C20, D10 are no longer in the NN_DEPT nickname. The row that you inserted before you created the savepoint SP1 (A10) still exists in the NN_DEPT nickname.

Restrictions on federated updates with application savepoints

Federated systems impose some restrictions on savepoint operations.

Savepoint support for write operations on remote data in federated applications is limited to DB2 Database for Linux, UNIX, and Windows. The following restrictions apply to the data source object that the nickname is defined for (as the target of a write operation):

- For Version 9.5, the data source object in a savepoint operation can be a nickname itself.
- For previous versions, the data source object in a savepoint operation cannot be a nickname.

You can activate savepoints in serial mode, the massively parallel processing (MPP) environment, and the federated one-phase commit or federated two-phase commit environments. Any restrictions that exist in these environments also apply to savepoints operations.

Chapter 11. Importing and exporting data for nicknames

You can use the IMPORT command to import data into a nickname and the EXPORT command to export data from a query that references a nickname.

You can only use the IMPORT commands with the following data sources:

- DB2 family
- Informix
- Microsoft SQL Server
- Oracle
- Sybase
- Teradata

Restrictions for importing data into nicknames

There are restrictions on using the IMPORT command to import data into a nickname.

The following restrictions apply when you use the IMPORT command to import data into a nickname:

- The remote object on which the nickname is defined must be a table. You cannot import data into a nickname that is defined on a view or synonym.
- The supported file types are IXF, ASC and DEL.
- The ALLOW WRITE ACCESS clause must be specified. This clause invokes the online import mode. The ALLOW WRITE ACCESS clause allows concurrent applications both read and write access to the import target table.
- You cannot use the COMMITCOUNT AUTOMATIC mode with nicknames.
- The COMMITCOUNT *n* must be specified with *n* being a valid, nonzero number.
- Only the INSERT and INSERT_UPDATE operations are supported with nicknames.
- The column types that are not supported with nicknames are LOBs and generated columns. To import LOB data to a remote table, the corresponding nickname column must be a VARCHAR data type.
- The following `filetype` modifiers are not supported with nicknames:

  ```
  dldelfiletype
  generatedignore
  generatedmissing
  identityignore
  identitymissing
  indexixf
  indexschema
  lobsinfile
  nodefaults
  no_type_idfiletype
  usedefaults
  ```

- Hierarchy (typed table) is not supported with nicknames.

If you submit an IMPORT command that does not adhere to these restrictions, the SQL error code -27999N is returned. For example:

```
SQL27999N The requested IMPORT operation into a remote target (nickname) cannot be
 performed.  Reason code = "reason_code"
```

IMPORT command with nicknames - examples

The examples show you how to import data into nicknames from different file types.

DEL file type

This example uses the INSERT_UPDATE option to import data from a DEL file type:

```
IMPORT FROM import_file_1.del OF DEL
   ALLOW WRITE ACCESS
   COMMITCOUNT 50
   INSERT_UPDATE INTO NICKNAME_1;
```

IXF file type

This example uses the INSERT option to import data from an IXF file type:

```
IMPORT FROM import_file_1.ixf OF IXF
   ALLOW WRITE ACCESS
   COMMITCOUNT 20
   INSERT INTO NICKNAME_1;
```

ASC file type

This example uses the INSERT option to import data from an ASC file type. The example includes the STRIPTBLANKS file modifier to truncate any trailing blank spaces in the data. The METHOD L parameter specifies the start and end of the column numbers.

```
IMPORT FROM import_file_1.asc OF ASC MODIFIED BY STRIPTBLANKS
   METHOD L(1 6, 8 32, 34 44, 46 48)
   ALLOW WRITE ACCESS
   COMMITCOUNT 20
   INSERT INTO NICKNAME_1;
```

Restrictions for exporting data using nicknames

There are restrictions on using the EXPORT command to export data from a query that references a nickname.

The following restrictions apply when you use the EXPORT command to export data using a nickname:

- The description of the target table that is necessary to perform the import CREATE operation is not saved in the IXF file format. Use the `db2look` utility to collect the information that you need to recreate the table.
- You can export data into the IXF and DEL file types. The ASC file type is not supported for exporting data from nicknames.

Chapter 12. Working with nicknames

A nickname is an identifier that an application uses to reference a data source object, such as a table or view. In a federated system, you use can nicknames to access data source objects and improve the performance of queries on remote data sources.

Nicknames in a federated system

When you want to select or modify data source data, you query the nicknames by using the SELECT, INSERT, UPDATE, and DELETE statements. You submit queries in DB2 SQL to the federated database.

You can join data from local tables and remote data sources with a single SQL statement, as if all the data is local. For example, you can join data that is in the following objects:

- A local DB2 table in the federated database, an Oracle table, and a Sybase view
- A DB2 UDB for z/OS table on one server, a DB2 UDB for z/OS table on another server, and an Excel spreadsheet

By processing SQL statements as if the data sources were ordinary relational tables or views within the federated database, the federated system can join relational data with data in nonrelational formats.

Tables and views in the federated database are *local objects*. Do not create nicknames for these objects. Instead use the actual object name in your SQL statements.

Remote objects are objects not located in the federated database. You need to create nicknames for these objects. For example:

- Tables and views in another database or instance on the federated system
- Tables and views in another database or instance on another system
- Tables and views in data sources such as Oracle, Sybase, and ODBC

Cursors declared WITH HOLD

You can use the WITH HOLD option on a cursor that is defined on a nickname.

For the DRDA wrapper and the DB2 Database for Linux, UNIX, and Windows data source, cursors that you declare using the WITH HOLD option remain open across multiple units of work.

If you declare a cursor WITH HOLD and the data source does not support this option, the attribute is ignored.

Triggers

A nickname cannot be an update target in a trigger.

You can include SELECT statements on nicknames in the trigger body. You cannot include INSERT, UPDATE, or DELETE statements on nicknames in the trigger body.

Accessing data with nicknames

With a federated system, you can easily access data, regardless of where it is stored. To access data, you create nicknames for your data source objects, such as tables and views.

For example, if the nickname DEPT represents the remote table EUROPE.PERSON.DEPT, you can use the statement SELECT * FROM DEPT to query information in the remote table. When you query a nickname, you do not have to remember the connection details about the data source. Thus, you do not need to be concerned with these issues:

- The name of the object at the data source
- The server on which the data source object resides
- The data source type on which the object resides, such as Informix and Oracle
- The query language or SQL dialect that the data source uses
- The data type mappings between the data source and federated server
- The function mappings between the data source and federated server

The underlying metadata in the federated database catalog provides the federated server with the information that it needs to process your queries. This metadata is gathered from the data sources when the federated server and database are set up and configured to access the data sources.

After the federated system is set up, you use the nicknames to query the data sources or to further enhance the federated system configuration.

The SQL statements you can use with nicknames

SQL statements support the use of nicknames.

Table 14. Common SQL statements for use with nicknames

SQL statement	Description	Authorization required
ALTER NICKNAME	Modifies an existing nickname by changing the local column name, the local data type, the federated column options, or the informational constraints. The table or view at the data source is not affected.	• SYSADM or DBADM • ALTER or CONTROL privilege on the nickname • ALTERIN privilege on the schema, if the schema name of the nickname exists • Definer of the nickname, as recorded in the DEFINER column of the catalog view for the nickname
ALTER TABLE	Changes a remote table that was created through the federated database by using transparent DDL. You cannot alter tables that were created natively on the data source. Can use informational constraints to add a referential integrity constraint to a nickname.	• SYSADM or DBADM • ALTER or CONTROL privilege on the nickname • ALTERIN privilege on the schema, if the schema name of the nickname exists

Table 14. Common SQL statements for use with nicknames (continued)

SQL statement	Description	Authorization required
COMMENT ON	Adds or replaces comments in the catalog descriptions of various objects, including functions, function mappings, indexes, nicknames, servers, server options, type mappings, and wrappers.	• SYSADM or DBADM • ALTER or CONTROL privilege on the object • ALTERIN privilege on the schema • Definer of the object, as recorded in the DEFINER column of the catalog view for the object
CREATE ALIAS	Defines an alias for a nickname.	• SYSADM or DBADM • IMPLICIT_SCHEMA authority on the database, if the implicit or explicit schema name of the alias does not exist • CREATEIN privilege on the schema, if the schema name of the alias refers to an existing schema
CREATE INDEX with SPECIFICATION ONLY clause	Creates an index specification (metadata) that indicates to the query optimizer that a data source object has an index. No actual index is created, only the specification is created.	• SYSADM or DBADM • CONTROL or INDEX privilege on the underlying data source object — *and either* IMPLICIT_SCHEMA authority on the database, or CREATEIN privilege on the schema
CREATE TABLE with the OPTIONS clause	Creates a remote table through the federated database using transparent DDL.	• SYSADM or DBADM • CREATETAB privilege on the database and USE privilege on the table space —*and either* IMPLICIT_SCHEMA authority on the database, or CREATEIN privilege on the schema
CREATE TABLE with the AS fullselect and DATA INITIALLY DEFERRED REFRESH clauses	Creates a materialized query table using a fullselect that references a nickname.	• SYSADM or DBADM • CREATETAB privilege on the database and USE privilege on the table space —*and either* IMPLICIT_SCHEMA authority on the database, or CREATEIN privilege on the schema • CONTROL privilege on the table or view • SELECT privilege on the table or view and ALTER privilege if REFRESH DEFERRED is specified
CREATE VIEW	Creates a view that references one or more nicknames.	• SYSADM or DBADM • CONTROL or SELECT privilege on the nickname—*and either* IMPLICIT_SCHEMA authority on the database, or CREATEIN privilege on the schema

Table 14. Common SQL statements for use with nicknames (continued)

SQL statement	Description	Authorization required
DELETE	Deletes rows from the data source object, such as a table or view that has a nickname.	• SYSADM or DBADM • DELETE privilege on the nickname and DELETE privilege on the underlying data source object • CONTROL privilege on the underlying data source object
DROP	Deletes an object, such as a nickname, federated view, or index specification. The table, view, or index at the data source is not affected. When the tables that are created by using the transparent DDL are dropped, the corresponding nickname for that table is also dropped.	• SYSADM or DBADM • DROPIN privilege on the schema for the object • CONTROL privilege on the object
GRANT	Grants privileges on nicknames and federated views, such as ALTER, DELETE, INDEX, INSERT, SELECT, or UPDATE. Privileges at the data source must be granted separately.	• SYSADM or DBADM • WITH GRANT OPTION for each identified privilege • CONTROL privilege on the object
INSERT	Inserts rows into the data source object, such as a table or view that has a nickname.	• SYSADM or DBADM • INSERT privilege on the nickname and INSERT privilege on the underlying data source object • CONTROL privilege on the underlying data source object
LOCK TABLE	Causes the remote object at the data source to be locked. Prevents concurrent application processes from changing a data source table that has a nickname.	• SYSADM or DBADM • SELECT privilege on the underlying table • CONTROL privilege on the underlying table.
REVOKE	Revokes privileges on nicknames and federated views, such as ALTER, DELETE, INDEX, INSERT, SELECT, or UPDATE. Privileges at the data source must be revoked separately.	• SYSADM or DBADM • CONTROL privilege on the object
SELECT	Selects rows from the data source object, such as a table or view that has a nickname.	• SYSADM or DBADM • SELECT privilege on the nickname and SELECT privilege on the underlying data source object • CONTROL privilege on the underlying data source object

Table 14. Common SQL statements for use with nicknames (continued)

SQL statement	Description	Authorization required
UPDATE	Updates the values in specified columns in rows in the data source object, such as a table or view that has a nickname.	• SYSADM or DBADM • UPDATE privilege on the nickname and UPDATE privilege on the underlying data source object • CONTROL privilege on the underlying data source object

When you submit a query to the federated database, the authorization privileges for the nickname in the query are checked. The authorization requirements of the data source object that the nickname refers to are only applied when the query is processed at the data source.

To select, insert, update, or delete data with a nickname, the authorization ID of the statement must include these privileges:

- The appropriate privilege on the nickname for the federated database to accept the request
- The appropriate privilege on the underlying table object for the data source to accept the request

For example to update a data source using a nickname, you need UPDATE privilege on the nickname and UPDATE privilege on the underlying data source object.

Creating nicknames over temporal tables

Federation support of temporal tables gives you the ability to create a nickname over a remote temporal table.

You can define a table with temporal attributes to create the following types of temporal tables:

- System-period temporal tables that maintain historical versions of its rows
- Application-period temporal tables that store data based on time criteria
- Bitemporal tables that are a combination of system-period temporal tables and application-period temporal tables

When you create a nickname over a temporal table, you can perform insert, update, and delete operations against the nickname.

Example

- Define a table named *policy_info_bus*:

```
CREATE TABLE policy_info_bus ( policy_id CHAR(4) NOT NULL,
 bus_start DATE NOT NULL,  bus_end DATE NOT NULL,
 PERIOD BUSINESS_TIME(bus_start, bus_end) );
```

- Issue a query against a temporal table nickname:

```
SELECT * from policy_info_bus_nickname;
```

However, temporal operations over nicknames are not supported. You cannot perform temporal DDL operations, alter operations, or query operations against nickname. For example, you cannot query the system time as of a specific date as shown in the following example:

```
SELECT * FROM policy_info_bus_nickname FOR BUSINESS_TIME AS OF '2012-12-02';
```

Accessing new data source objects

To access new data source objects, you must create nicknames for them. Use the CREATE NICKNAME statement for data sources that do not have nicknames.

Before you begin

The federated system must be configured to access the data source.

A server definition for the data source server on which the object resides must exist in the federated database. You create a server definition with the CREATE SERVER statement.

To insert, update, or delete data with a nickname, all of the following privileges must be true:

- The privileges held by the authorization ID of the statement must include the necessary SELECT, INSERT, UPDATE, and DELETE privileges on the nickname for the federated database to accept the request.
- The user ID at the data source must have the necessary SELECT, INSERT, UPDATE, and DELETE privileges on the underlying table object for the data source to accept the request.
- The user ID at the data source must be mapped to the authorization ID at the federated server through a user mapping.

You must have the required authorizations for the CREATE NICKNAME statement.

- SYSADM or DBADM
- IMPLICIT_SCHEMA authority on the federated database, if the implicit or explicit schema name of the nickname does not exist
- CREATEIN privilege on the schema, if the schema name of the nickname exists

About this task

Periodically, you need to access data source objects that lack nicknames. These might be new objects added to a data source, such as a newly created view. These might be existing objects that were not registered with the federated server when it was initially setup. In either case, you must create a nickname for the object.

Procedure

Issue the CREATE NICKNAME statement to access new data source objects.

Creating nicknames for relational and nonrelational data sources

The CREATE NICKNAME statement differs slightly for relational and nonrelational data sources.

For relational data sources the CREATE NICKNAME statement syntax is:

```
CREATE NICKNAME nickname_name FOR server_name."remote_schema"."object_name"
```

nickname_name
A unique nickname for the data source object. Nicknames can be up to 128 characters in length.

The nickname is a two-part name—the schema and the nickname. If you omit the schema when creating the nickname, the schema of the nickname will be the authentication ID of the user creating the nickname. The default values for the schema name are chosen based on the following hierarchy:

1. CURRENT SCHEMA special register
2. SESSION_USER special register
3. SYSTEM USER special register
4. Authorization ID connected to the database

FOR *server_name."remote_schema"."object_name"*
A three-part identifier for the remote data source object. If your data source does not support schemas, omit the schema from the CREATE NICKNAME statement.

- *server_name* is the name assigned to the data source server in the CREATE SERVER statement.
- *remote_schema* is the name of the remote schema to which the object belongs.
- *object_name* is the name of the remote object that you want to access.

OPTIONS (*options_list***)**
Information about the nickname that enables the SQL Query Compiler and the wrapper to efficiently execute queries.

For some nonrelational data sources the CREATE NICKNAME statement syntax is:

```
CREATE NICKNAME nickname_name column_definition_list
     FOR SERVER server_name
     OPTIONS (options_list)
```

nickname_name
A unique nickname for the data source object, as described above for relational data sources.

column_definition_list
A list of nickname columns and data types.

FOR SERVER *server_name*
The local name that you created for the remote server in the server definition information CREATE SERVER statement.

OPTIONS (*options_list***)**
Information about the nickname that enables the SQL Query Compiler and the wrapper to efficiently execute queries.

Nickname column and index names

When you create a nickname, the federated server uses an algorithm to generate nickname column and index names.

This algorithm is enhanced in Version 9.5 to match the nickname column and index names and the original names as closely as possible.

The algorithm applies only to relational data sources. For nonrelational data sources, the name specified in the CREATE NICKNAME statement remains unchanged.

The following table shows examples of nickname column names that result from remote column names. In these examples, the remote columns are in separate remote tables. The nickname column names that differ from the remote column name are shown in bold.

Table 15. Nickname column names generated from remote column names.

	Name that results from current algorithm for data sources that ..			**Name that resulted before Version 9.5 (algorithm folds and converts non-alphanumeric characters)**
Remote column name	**Fold identifiers to upper case**	**Fold identifiers to lower case**	**Leave identifiers unchanged**	
column1	column1	**COLUMN1**	**COLUMN1**	**COLUMN1**
Column1	Column1	Column1	**COLUMN1**	**COLUMN1**
COLUMN1	COLUMN1	COLUMN1	COLUMN1	COLUMN1
column-1	column-1	column-1	column-1	**COLUMN_1**
Column-1	Column-1	Column-1	Column-1	**COLUMN_1**
COLUMN-1	COLUMN-1	COLUMN-1	COLUMN-1	**COLUMN_1**

If the remote columns are in the same table, the federated server uses the existing algorithm that generates unique names. For example, for the nickname column names for the data sources that leaves identifiers unchanged, the remote column names result in the following names:

- column1 results in COLUMN1
- Column1 results in COLUMN10
- COLUMN1 results in COLUMN11

Accessing data sources using pass-through sessions

You can submit SQL statements directly to data sources by using a special mode that is called pass-through. The pass-through session allows you to submit SQL statements in the SQL dialect for that data source.

Use a pass-through session when you want to perform an operation that is impossible with the SQL. For example, use a pass-through session to create a procedure, create an index, or perform queries in the native dialect of the data source.

The data sources that support pass-through only accept SQL statements in a pass-through session.

Similarly, you can use a pass-through session to perform actions that are not supported by SQL, such as certain administrative tasks. The administrative tasks that you can perform depend on the data source. For example, for DB2 Database for Linux, UNIX, and Windows, you can run the statistics utility for the data source, but you cannot start or stop the remote database.

You can query only one data source at a time in a pass-through session. Use the SET PASSTHRU command to open a session. When you use the SET PASSTHRU

RESET command it closes the pass-through session. If you use the SET PASSTHRU command instead of the SET PASSTHRU RESET command, the current pass-through session is closed and a new pass-through session is opened.

You can declare a cursor WITH HOLD in a pass-through session. If you use this option with the DRDA wrapper and the DB2 Database for Linux, UNIX, and Windows data source, the cursors that you declare remain open across multiple units of work. If you declare a cursor WITH HOLD and the data source does not support this option, the attribute is ignored.

You cannot use pass-through sessions with nonrelational data sources.

Accessing heterogeneous data through federated views

A *federated view* is a view in the federated database whose base tables are located at remote data sources. The federated view references base tables with nicknames, instead of the data source table names.

Before you begin

You must have one of the following authorizations to issue the CREATE VIEW statement:

- SYSADM or DBADM
- For each nickname in any fullselect, both:
  - CONTROL or SELECT privilege on the underlying table or view
  - One of the following authorities or privileges:
    - IMPLICIT_SCHEMA authority on the federated database, if the implicit or explicit schema name of the view does not exist
    - CREATEIN privilege on the schema, if the schema name of the view refers to an existing schema

Privileges for the underlying objects are not considered when defining a view on a federated database nickname.

About this task

When you query from a federated view, data is retrieved from the remote data source. The action of creating a federated database view of data source data is sometimes called “creating a view on a nickname.” This is because you reference the nicknames instead of the data sources when you create the view.

These views offer a high degree of data independence for a globally integrated database, just as views defined on multiple local tables do for centralized relational database managers.

Restrictions:

- Federated views with UNION ALL statements are read-only views.
- Federated views that include more than one nickname in the FROM clause are read-only views.
- Federated views that include only one nickname in the FROM clause might be read-only views.
  - If the nickname in the FROM clause is to a nonrelational data source, the federated view is read-only.

– If you include other nicknames as predicates or as subqueries when you create the view, the federated view can be updated.

Procedure

Issue the CREATE VIEW statement to create a federated view.
Authorization requirements of the data source for the table or view referenced by the nickname are applied when the query is processed. The authorization ID of the statement can be mapped to a different remote authorization ID by a user mapping.

Creating federated views - examples

These examples show how to create federated views to access data from several data sources. The examples show the syntax for the CREATE VIEW statement for federation.

Example: Creating a federated view that merges similar data from several data source objects

You are working with customer data on separate servers: one in Europe, one in Asia, and one in South America. The European customer data is in an Oracle table. The nickname for that table is ORA_EU_CUST. The Asian customer data is in a Sybase table. The nickname for that table is SYB_AS_CUST. The South American customer data is in an Informix table. The nickname for that table is INFMX_SA_CUST. Each table has columns containing the customer number (CUST_NO), the customer name (CUST_NAME), the product number (PROD_NO), and the quantity ordered (QUANTITY). To create a view from these nicknames that merge this customer data, issue the following statement:

```
CREATE VIEW FV1
   AS SELECT * FROM ORA_EU_CUST
   UNION
   SELECT * FROM SYB_AS_CUST
   UNION
   SELECT * FROM INFMX_SA_CUST
```

Example: Joining data to create a federated view

You are working with customer data on one server and sales data on another server. The customer data is in an Oracle table. The nickname for that table is ORA_EU_CUST. The sales data is in a Sybase table. The nickname for that table is SYB_SALES. You want to match up the customer information with the purchases that are made by those customers. Each table has a column containing the customer number (CUST_NO). To create a federated view from these nicknames that joins this data, issue the following statement:

```
CREATE VIEW FV4
   AS SELECT A.CUST_NO, A.CUST_NAME, B.PROD_NO, B.QUANTITY
   FROM ORA_EU_CUST A, SYB_SALES B
   WHERE A.CUST_NO=B.CUST_NO
```

Creating a nickname on a nickname

You can create a nickname on a nickname.

Procedure

1. On the Windows federated server, install IBM InfoSphere Federation Server.
2. Configure the Windows federated server to access Excel data sources.

3. On the Windows federated server, create a nickname for the Excel spreadsheet.
4. On the AIX® federated server, install IBM InfoSphere Federation Server.
5. Configure the AIX federated server to access DB2 family data sources.
6. On the AIX federated server, create a nickname for the Excel nickname on the Windows federated server.

Example

Access a Microsoft Excel spreadsheet from an AIX federated server

You have a federated server on AIX and a federated server on Windows. You want to access an Excel spreadsheet from both federated servers. However, the Excel wrapper is only supported on federated servers on Windows. To access the Excel spreadsheet from the AIX federated server:

Selecting data in a federated system

How you select data in a federated systems depends on the type of data source that you select from.

Some of the types of distributed requests used with a federated system are requests that query:

- A single remote data source
- A local data source and a remote data source
- Multiple remote data sources
- A combination of remote and local data sources

To select data from the data sources, use the nicknames for the data source objects in the SELECT statement.

The federated database is a local data source. Tables and views in the federated database are local objects. You do not create nicknames for these objects, you use the actual object name in the SELECT statements.

Remote data sources include: another DB2 Database for Linux, UNIX, and Windows database instance on the federated server, another DB2 Database for Linux, UNIX, and Windows database instance on another server, and other data sources. Objects that reside on remote data sources are remote objects.

Selecting data in a federated system - examples

This topic illustrates a scenario in which a federated server access multiple data sources and provides examples of SELECT statements.

Example: A federated server is configured to access a DB2 for z/OS data source, a DB2 for System i data source, and an Oracle data source. Stored in each data source is a table that contains sales information. This configuration is depicted in the following figure.

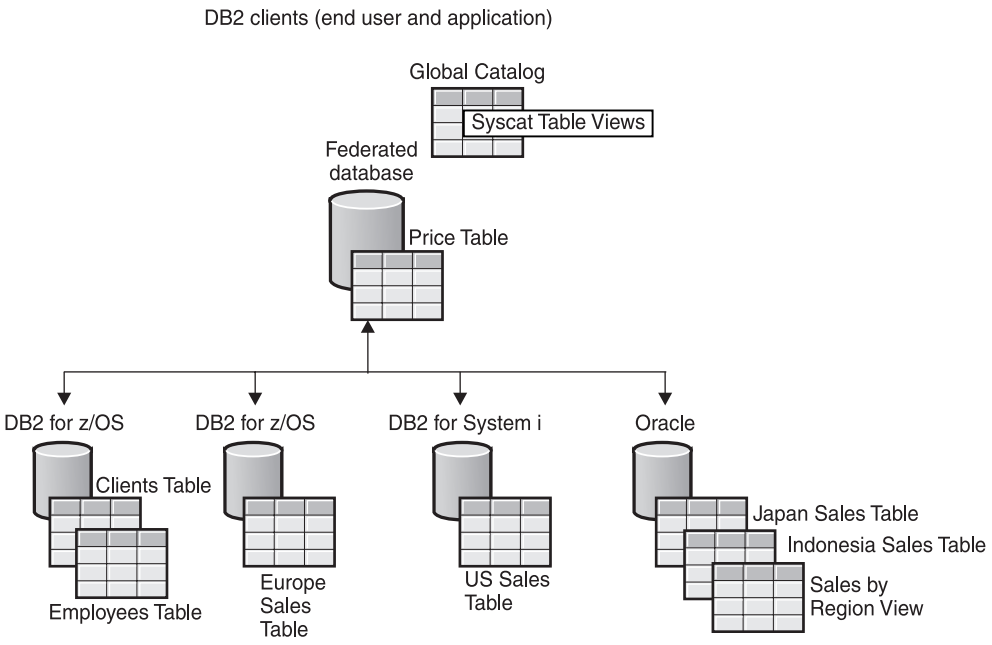


Figure 8. Sample federated system with DB2 and Oracle data sources

The sales tables include columns that record the customer number (CUST_NO), the quantity ordered (QUANTITY), and the product number ordered (PROD_NO). Also, a local table in the federated database contains price information. The price table includes columns that record the product number (PROD_NO) and the current price (PRICE).

The nicknames for the remote data source objects are stored in the SYSCAT.TABLES tables, as shown in the following tables. The TYPE column indicates the type of object, such as nickname (N), local table (T), or view (V).

Table 16. Data source information

Data source object name	Type of object	Location
PRICES	Local table	Federated database
EUROPE_SALES	Remote table	DB2 for z/OS database
US_SALES	Remote table	DB2 for System i database
JAPAN_SALES	Remote table	Oracle database
SALES_BY_REGION	Remote view	Oracle database

Table 17. SYSCAT tables

TABNAME	TYPE
PRICES	T
FED_PRICES	N
Z_EU_SALES	N
Si_US_SALES	N
ORA_JAPANSALES	N
ORA_REGIONSALES	N
...	

To select data using a nickname, all of the following privileges must be true:

- The privileges held by the authorization ID of the statement must include the SELECT privilege on the nickname (for the federated database to accept the request).
- The user ID at the data source must have the SELECT privilege on the underlying table object (for the data source to accept the request).
- The user ID at the data source must be mapped to the authorization ID at the federated server through a user mapping.

The following SELECT statement examples use the sample federated system described above.

Example: Querying a single data source:

Z_EU_SALES contains the products ordered by your European customers. It also includes the quantity ordered at each sale. This query uses a SELECT statement with an ORDER BY clause to list the sales in Europe and sorts the list by customer number:

```
SELECT CUST_NO, PROD_NO, QUANTITY
     FROM Z_EU_SALES
     ORDER BY CUST_NO
```

Example: Joining a local data source and a remote data source:

PRICES is a table that resides in the federated database. It contains the price list for the products that you sell. You want to select the prices from this local table that correspond to the products listed in Z_EU_SALES. You also want to sort the result set by the customer number.

```
SELECT sales.CUST_NO, sales.PROD_NO, sales.QUANTITY
     FROM Z_EU_SALES sales, PRICES
     WHERE sales.PROD_NO=PRICES.PROD_NO
     ORDER BY sales.CUST_NO
```

Example: Querying multiple remote data sources:

You want to gather all the sales information from each region and order the result set by product number.

```
WITH GLOBAL_SALES (Customer, Product, Quantity) AS
     (SELECT CUST_NO, PROD_NO, QUANTITY FROM Z_EU_SALES
       UNION ALL
     SELECT CUST.NO,PROD.NO, QUANTITY FROM iS_US_SALES
       UNION ALL
     SELECT CUST.NO,PROD.NO, QUANTITY FROM ORA_JAPANSALES)
     SELECT Customer, Product, Quantity
     FROM GLOBAL_SALES
     ORDER BY Product
```

A view at the Oracle data source lists the sales for Japan and Indonesia. The nickname for this view is ORA_SALESREGION. You want to combine this information with the sales from the United States and display the product prices next to each sale.

```
SELECT us_jpn_ind.CUST_NO, us_jpn_ind.PROD_NO,
        us_jpn_ind.QUANTITY, us_jpn_ind.QUANTITY*PRICES.PRICE
     AS SALEPRICE FROM
     (SELECT CUST_NO, PROD_NO, QUANTITY
     FROM ORA_SALESREGION
     UNION ALL
```

```
SELECT CUST_NO, PROD_NO, QUANTITY
FROM iS_US_SALES us ) us_jpn_ind,PRICES
WHERE us_jpn_ind.PROD_NO = PRICES.PROD_NO
ORDER BY SALEPRICE DESC
```

Informational constraints on nicknames

You can use informational constraints on nicknames to improve the performance of queries. Informational constraints are rules that the optimizer uses to improve performance but that the database manager does not enforce.

You can specify the following for nicknames:

- Referential constraints
- Check constraints
- Functional dependency constraints
- Primary key constraints
- Unique constraints

Specifying informational constraints on nicknames

To improve the performance of queries on remote data sources, you can add informational constraints to nicknames. However, the federated server does not enforce or check the constraints.

About this task

Restrictions

After you define informational constraints on a nickname, you can only alter the column names for that nickname after you remove the informational constraints.

About this task

For relational data sources, you can specify informational constraints when you alter a nickname.

For nonrelational data sources, you can specify informational constraints when you create a nickname or alter a nickname.

Procedure

To specify informational constraints on a nickname from the DB2 command line, issue the CREATE NICKNAME statement or the ALTER NICKNAME statement with the appropriate constraint attributes.

Specifying informational constraints on nicknames - examples

These examples illustrate the use of informational constraints on nicknames. You use the CREATE or ALTER NICKNAME statements for check constraints, referential constraints, and other data structures.

Example: Informational check constraint

In the following remote table, the data in the salary column is always greater than 10000.

```
CREATE TABLE account.salary (
    empno INTEGER NOT NULL PRIMARY KEY,
    salary  INTEGER NOT NULL
);
```

Create a nickname for this table:

```
CREATE NICKNAME account.salary FOR myserv.account.salary;
```

Then add informational check constraints for the nickname by issuing the following statement:

```
ALTER NICKNAME account.salary ADD CONSTRAINT cons1 CHECK( salary > 10000 )
NOT ENFORCED
ENABLE QUERY OPTIMIZATION;
```

Example: Informational referential constraint: nickname to nickname

In this example, there are two nicknames N1 and N2. Column F1 of nickname N2 contains the key value in column P1 of nickname N1. You can define the referential constraint on nickname N2 by issuing the following statement:

```
ALTER NICKNAME SCHEMA1.N2 ADD CONSTRAINT ref1
                          FOREIGN KEY (F1) REFERENCES SCHEMA1.N1 (P1)
                          NOT ENFORCED;
```

Example: Informational referential constraint: nickname to table

In this example, nickname N3 with column F1 contains the key value in column P1 of table T1. You can define the referential constraint on nickname N3 by issuing the following statement:

```
ALTER NICKNAME SCHEMA1.N3 ADD CONSTRAINT ref1
                          FOREIGN KEY (F1) REFERENCES SCHEMA1.T1 (P1)
                          NOT ENFORCED;
```

Example: Informational referential constraint: table to nickname

In this example, table T2 with column F1 contains the key value in column P1 of nickname N4. You can define the referential constraint on table T2 by issuing the following statement:

```
ALTER TABLE SCHEMA1.T2 ADD CONSTRAINT ref1
                          FOREIGN KEY (F1) REFERENCES SCHEMA1.N4 (P1)
                          NOT ENFORCED;
```

Example: Functional dependency

In this example, the column pair C1 and C2 uniquely determine the value in the column P1. You can define the functional dependency by issuing the following statement:

```
ALTER NICKNAME SCHEMA1.NICK1 ADD CONSTRAINT FD1 CHECK( P1 DETERMINED BY (C1,C2) )
                         NOT ENFORCED ENABLE QUERY OPTIMIZATION;
```

Example: Table-structured file

This statement defines a primary key for a table-structured file:

```
CREATE NICKNAME MY_FILE (
       X INTEGER NOT NULL,
       Y INTEGER,
       PRIMARY KEY (X) NOT ENFORCED
)  FOR SERVER MY_SERVER OPTIONS(FILE_PATH '/usr/pat/DRUGDATA1.TXT');
```

Star schema

The statement creates four dimension tables and one fact table:

```
CREATE TABLE SCHEMA.FACT (
       LOCATION_CODE   INTEGER NOT NULL,
       PRODUCT_CODE    INTEGER NOT NULL,
       CUSTOMER_CODE   INTEGER NOT NULL,
       SDATE           DATE NOT NULL,
       SALES           INTEGER NOT NULL
);

CREATE TABLE SCHEMA.LOCATION (
       LOCATION_CODE   INTEGER NOT NULL PRIMARY KEY,
       STATE           CHAR(2) NOT NULL,
       SHOP_ID         INTEGER NOT NULL,
       ...
);

CREATE TABLE SCHEMA.PRODUCT (
       PRODUCT_CODE    INTEGER NOT NULL PRIMARY KEY,
       PRODUCT_CAT     INTEGER NOT NULL,
       PRODUCT_NAME    VARCHAR(20) NOT NULL,
       ...
);

CREATE TABLE SCHEMA.CUSTOMER (
       CUSTOMER_CODE   INTEGER NOT NULL PRIMARY KEY,
       NAME            VARCHAR(20) NOT NULL,
       TEL             VARCHAR(10) NOT NULL,
       ...
);

CREATE TABLE SCHEMA.TIMEDIM (
       SDATE           DATE NOT NULL UNIQUE,
       YEAR            INTEGER NOT NULL,
       QUARTER         INTEGER NOT NULL,
       ...
);
```

The federated server creates the following nicknames for the fact table and the dimension tables:

```
CREATE NICKNAME SCHEMA.FACT FOR SERVER.SCHEMA.FACT;
CREATE NICKNAME SCHEMA.LOCATION FOR SERVER.SCHEMA.LOCATION;
CREATE NICKNAME SCHEMA.PRODUCT FOR SERVER.SCHEMA.PRODUCT;
CREATE NICKNAME SCHEMA.CUSTOMER FOR SERVER.SCHEMA.CUSTOMER;
CREATE NICKNAME SCHEMA.TIMEDIM FOR SERVER.SCHEMA.TIMEDIM;
```

You can define the following foreign key relationship by issuing the following statements:

```
ALTER NICKNAME SCHEMA.FACT ADD CONSTRAINT L1 FOREIGN KEY (LOCATION_CODE)
                               REFERENCES SCHEMA.LOCATION(LOCATION_CODE)
                               NOT ENFORCED ENABLE QUERY OPTIMIZATION;

ALTER NICKNAME SCHEMA.FACT ADD CONSTRAINT P1 FOREIGN KEY (PRODUCT_CODE)
                               REFERENCES SCHEMA.PRODUCT(PRODUCT_CODE)
                               NOT ENFORCED ENABLE QUERY OPTIMIZATION;

ALTER NICKNAME SCHEMA.FACT ADD CONSTRAINT C1 FOREIGN KEY (CUSTOMER_CODE)
                               REFERENCES SCHEMA.CUSTOMER(CUSTOMER_CODE)
                               NOT ENFORCED ENABLE QUERY OPTIMIZATION;

ALTER NICKNAME SCHEMA.FACT ADD CONSTRAINT S1 FOREIGN KEY (SDATE)
                               REFERENCES SCHEMA.TIMEDIM(SDATE)
                               NOT ENFORCED ENABLE QUERY OPTIMIZATION;
```

When the value of the TEL column in the CUSTOMER nickname is unique, you can add the following informational unique constraint with this statement:

```
ALTER NICKNAME SCHEMA.CUSTOMER ADD CONSTRAINT U1 UNIQUE( TEL )
                                    NOT ENFORCED ENABLE QUERY OPTIMIZATION;
```

When the value of the SHOP_ID column in the LOCATION nickname uniquely determines the value of the LOCATION_ID column, you can define the following functional dependency with this statement:

```
ALTER NICKNAME SCHEMA.LOCATION
ADD CONSTRAINT F1 CHECK( LOCATION_ID DETERMINED BY SHOP_ID )
                         NOT ENFORCED ENABLE QUERY OPTIMIZATION;
```

Because the value of the QUARTER column in the TIMEDIM nickname is between 1 and 4, you can define the following informational check constraint with this statement:

```
ALTER NICKNAME SCHEMA.TIMEDIM
ADD CONSTRAINT Q1 CHECK( QUARTER BETWEEN 1 AND 4 )
                         NOT ENFORCED ENABLE QUERY OPTIMIZATION;
```

The statements in this example create nicknames for remote tables. Nicknames have primary keys when remote tables have primary keys. When you create nickname on views, nicknames lack primary keys. In this case, you can change the nickname to add an informational primary key constraint. For example:

```
CREATE NICKNAME SCHEMA.LOCATION FOR SERVER.SH.V_LOCATION;
ALTER NICKNAME SCHEMA.LOCATION
      ADD CONSTRAINT P1 PRIMARY KEY ( LOCATION_CODE ) NOT ENFORCED;
```

Updating nickname statistics

Retrieving the statistics for a nickname ensures that the query optimizer is using the most recent information about the nickname when it generates the query access plans.

Nickname statistics update facility - overview

You retrieve the statistics for a nickname to provide the query optimizer with accurate and current information about the nickname. The optimizer uses this information to determine an optimal access plan for a query.

When you register a nickname for a data source object, the federated server adds information about that data source object to the system catalog on the federated database. The query optimizer uses this information to plan how to retrieve data from the data source object. The federated database does not automatically detect changes to the data source objects, so the information in the system catalog can become outdated.

You can retrieve the currently available statistics on database nicknames, columns, and indexes from the remote data source. Typically, you might use the DB2 command line processor to manually update nickname statistics when new statistics are collected on a remote table.

You can retrieve nickname statistics for the following relational data sources:

- DB2 family (DRDA)
- Informix
- JDBC
- Microsoft SQL Server

- ODBC
- Oracle (NET8)
- Sybase (CTLIB)
- Teradata

You can retrieve nickname statistics for the following nonrelational data sources:
- BioRS
- Excel
- Table-structured files
- XML on the root nickname

The statistics collected for each data source varies.

You can update the following statistics using the catalog-based method for statistics collection:
- CARD
- FPAGES
- NPAGES
- OVERFLOW
- COLCARD
- HIGH2KEY
- LOW2KEY
- NLEAF
- NLEVELS
- CLUSTERFACTOR
- CLUSTERRATIO
- FULLKEYCARD
- FIRSTKEYCARD

You can update the following statistics using the data-based method for statistics collection:
- CARD
- COLCARD
- HIGH2KEY
- LOW2KEY
- FULLKEYCARD
- FIRSTKEYCARD

You can retrieve the statistics of a single nickname or all nicknames in a schema on a specific server definition. You can also retrieve statistics for nicknames on data source objects with Oracle Label Security. If the database is restricted, only users with the appropriate access can view the HIGH2KEY and LOW2KEY statistics. For unrestricted databases, the nickname statistics on objects with Oracle Label Security are exposed and might pose a security risk. In those instances, you can restrict access to the system catalogs that contain sensitive information. If any part of the retrieval fails, the database rolls back the changes.

You can retrieve nickname statistics on any operating system that supports the federated server.

Methods of retrieving nickname statistics

You can choose the method to use for statistics collection, and you can limit your choices to specific columns and indexes. The catalog-based method has better performance. By contrast, the data-based method provides more up-to-date statistics, but takes longer to run.

You can choose one of the following methods for statistics collection:

- Catalog-based method

  The catalog-based method copies statistics from data source catalog tables to the federated catalog table. Only those statistics that can be semantically mapped to federated statistics are copied. However, the nickname statistics are only as accurate and up-to-date as the information currently in the catalog at the remote source. If statistics information is out-of-date, the nickname statistics collected are also out-of-date. When you use the catalog-based method, ensure that statistics on the remote source are current.

  Because statistics are copied from the remote source catalog to the catalog on the federated server, the catalog-based method of statistics collection is generally very fast.

- Data-based method

  The data-based method does not depend on the statistics at the remote source. This method generates its own statistics empirically through the results of the queries that it issues against the nicknames. With this method, the statistics that are collected accurately reflect the remote data.

  The data-based method can be slow if the row size of the nicknames involved is large. The queries typically involve large sorts and aggregates. For this reason, choose data-based statistics collection only if satisfactory statistics cannot be obtained by the catalog-based method.

If you want to increase the performance of the data-based method at the expense of the quality of the statistics gathered, limit statistics collection to the types of columns and indexes for which the benefit is greatest. Those types of columns include columns that are involved in predicates, join keys, grouping operations, or columns that are part of one or more indexes.

With the catalog-based method, you generally do not need to limit statistics collection to specific columns or indexes, because the overhead of this collection method is very low.

Retrieving nickname statistics

You can retrieve the statistics for a nickname to ensure that the query optimizer uses the information about the nickname that is available at the data source. You can update nickname statistics for a single nickname, multiple nicknames, or all nicknames.

About this task

The query optimizer gathers statistics, including the HIGH2KEY and LOW2KEY nickname statistics, for data source objects that use label-based access control or Oracle Label Security. For databases that are use these types of security, only users with the appropriate access level can see the statistics. For unrestricted databases, you can restrict access or set the HIGH2KEY and LOW2KEY nickname statistics to an empty or meaningless value.

By default, statistics collection is attempted for all columns and all indexes of a nickname. You can limit statistics to specific columns and indexes and specify a log file.

Before you begin

The following prerequisites apply when you use the command line prompt to update statistics:

- The federated server creates the log file on the server. The directories that you list in the path must exist.
- The privileges for the fenced user ID of the federated instance must include the privilege to create the log file in the specified location.

Restrictions

The user ID that you use to connect to the federated database must be mapped to the remote data source table.

If the local column name or type does not represent the default type mapping from the remote column name or type, the nickname statistics update utility does not retrieve column statistics.

Procedure

Call the stored procedure SYSPROC.NNSTATS to update nickname statistics from the DB2 command line or within an application.

Retrieving nickname statistics - examples

These examples show how to retrieve nickname statistics from the command line by using the SYSPROC.NNSTAT stored procedure.

Example: Retrieving all statistics

The federated server retrieves the statistics for the nicknames on the DB2SERV server and does not create a log.

```
CALL SYSPROC.NNSTAT('DB2SERV',NULL,NULL,NULL,NULL,0,NULL,?)
```

Retrieving all statistics for a schema and returning a log

Example: Retrieving statics with the catalog-based method

The federated server retrieves the statistics for the nickname STAFF in the ADMIN schema. Statistics are gathered for columns 1 through 5 and indexes 1 and 2. The catalog-based method is used to collect statistics. The federated server writes the log to the /home/iiuser/reportlogs/log1.txt file.

```
CALL SYSPROC.NNSTAT(
 NULL, 'ADMIN','STAFF','COL1, COL2, COL3, COL4, COL5','IND1, IND2',1,
  '/home/iiuser/reportlogs/log1.txt',?)
```

In this example, the federated server retrieves the statistics for all the nicknames on the DB2Serv server in the admin schema. The federated server writes the log to the /home/iiuser/stats/recent.log file.

```
CALL SYSPROC.NNSTAT(
 'DB2Serv', 'admin', NULL, NULL, NULL, 0, '/home/iiuser/stats/recent.log', ?)
```

Restrictions on HIGH2KEY and LOW2KEY statistics

When you collect the HIGH2KEY and LOW2KEY statistics for a DB2 nickname, the information for some columns is not collected.

About this task

Restrictions

When a wrapper creates a nickname on a remote table or nickname, the wrapper collects the HIGH2KEY and LOW2KEY statistics only for numeric columns, and only when the column cardinality is greater than 3.

Procedure

To collect HIGH2KEY and LOW2KEY statistics, use either of the following methods:

Procedure

- Use SYSPROC.NNSTAT with the METHOD parameter set to 2. This setting specifies data-based statistics collection. The data-based method queries data from the remote table to calculate the values for local statistics. This method, however, can use significant resources at the remote server and the federated server.
- Issue the SQL UPDATE statement to update the HIGH2KEY and LOW2KEY columns in the SYSSTAT.COLUMNS view. In this case, you must manually determine the correct values for HIGH2KEY and LOW2KEY.

Creating a DB2 tools catalog

When you update the statistics for a nickname, you can use a DB2 tools catalog to schedule subsequent updates to nickname statistics. If you lack a DB2 tools catalog, you are prompted to create a catalog. You can create a DB2 tools catalog from the command line prompt.

Before you begin

The DB2 administration server must be installed.

Procedure

Issue the CREATE TOOLS catalog command.
Specify the system that you want to create a database on for the DB2 tools catalog.

What to do next

The database must be on a cataloged system that has no metadata storage. If the system you want is not cataloged, you must catalog the system before you create the database for the DB2 tools catalog.

Viewing the status of the updates to nickname statistics

After you request an update to the statistics for a nickname, you can view the status of the update.

About this task

Procedure

To view the status of updates to nickname statistics from the DB2 command line, look in the SYSPROC.FED_STATS table.

The following example shows a describe of the table: SYSPROC.FED_STATS (The actual length of the columns is reduced to simplify the example.):

```
db2 describe table sysproc.fed_stats

Column Type Type
name                           schema    name                Length   Scale Nulls
------------------------------ --------- ------------------ -------- ----- ------
SERVER                         SYSIBM    VARCHAR              128     0     Yes
SCHEMA                         SYSIBM    VARCHAR              128     0     Yes
NICKNAME                       SYSIBM    VARCHAR              128     0     Yes
STATS_UPDATE_TIME              SYSIBM    TIMESTAMP             10     0     No
LOG_FILE_PATH                  SYSIBM    VARCHAR             1000     0     Yes
SQLCODE                        SYSIBM    INTEGER                4     0     Yes
SQLSTATE                       SYSIBM    CHARACTER              5     0     Yes
STATUS                         SYSIBM    VARCHAR             1000     0     Yes
8 record(s) selected.


db2 "select * from sysproc.fed_stats"

SERVER  SCHEMA  NICKNAME STATS_UPDATE_TIME           LOG_FILE_PATH  SQLCODE
------  ------- -------- -------------------------  -------------  ----------
ORA8    HAROLDL NICK1    2006-05-02-12.03.24.927112  -             1791 42704

SQLSTATE  STATUS
--------  ------------------------------------------------------------------
SQL1791N  The specified server definition, schema, or nickname does not exist.

1 record(s) selected.

.
```

SYSPROC.NNSTAT stored procedure

Retrieve currently available statistics on one or more nicknames. The statistics are saved in the system catalog on the federated database.

Authorization

SYSPROC.NNSTAT is a fenced procedure. The privileges for the fenced user ID of the federated instance must include the privilege to create the log file in the specified location.

Syntax

```
CALL SYSPROC.NNSTAT(
                    SERVER          VARCHAR(128)
                    SCHEMA          VARCHAR(128)
                    NICKNAME        VARCHAR(128)
                    COLNAMES        CLOB(2M)
                    INDEXNAMES      CLOB(2M)
                    METHOD          SMALLINT
                    LOG_FILE_PATH   VARCHAR(1000)
                    OUT_TRACE       VARCHAR(2000)
                    UPDATE_METHOD   SMALLINT
                    )
```

Parameters

Server The server on which the federated server gathers the nickname statistics. This server is what the user registers to define a data source in the federated database. If you specify one nickname, you can specify NULL for this parameter.

Schema

If NULL is specified, the federated server retrieves all the nicknames under the given server. If the Server parameter is NULL, the federated server retrieves the statistics of the nickname under the given schema. If the Schema parameter and Nickname parameter are NULL and you specify a server, the federated server retrieves statistics on the given server.

Nickname

The name of the nickname. If you specify a nickname, you must also specify a schema.

Colnames

The names of the columns that are specified as column-name identifiers.

You can specify this parameter for a single nickname only. If you specify column names, you must also specify a schema and a nickname.

If NULL is specified, statistics are collected for all columns. NULL is the default.

Only the specified columns are processed. If an empty string ('') is specified, columns are not processed.

Indexnames

The names of the indexes that are specified as index-name identifiers.

You can specify this parameter for a single nickname only. If you specify index names, you must also specify a schema and a nickname. Only the specified indexes are processed.

If NULL is specified , statistics are collected for all indexes. NULL is the default.

If an empty string ('') is specified, indexes are not processed.

Method

The method for collecting statistics information from the data source.

0 or NULL

The catalog-based method is used first. If this method fails, then the data-based method is used. This is the default.

1 Catalog-based statistics collection. The catalog-based method maps information from remote catalogs to local statistics for the nickname. This method is valid for relational nicknames only.

2 Data-based statistics collection. The data-based method queries data from the remote table to calculate the values for local statistics. This method is valid for both relational and nonrelational nicknames.

This method is the default for relational nicknames if the catalog-based method fails for a given nickname. Typically, the reason that statistics cannot be collected is because nicknames are defined for remote views. In this case, statistics are not available at the remote source.

Collection of nickname statistics is not supported against graphic columns in non-Unicode databases for columns statistics HIGH2KEY and LOW2KEY. DB2 does not support conversion from graphic to character data types in a non-Unicode database. Therefore, you cannot use method 2 to store graphic values into HIGH2KEY and LOW2KEY VARCHAR2 catalog columns.

Log_File_Path
The path name and file name for the log file. The federated server creates the log file on the server. The directories that you list in the path must exist. On Windows, use two backslashes to specify the log path. For example: c:\\temp\\nnstat.log. If you specify NULL, the federated server does not create a log.

Out_Trace
The trace.

Update_Method
The method for updating catalog statistics.

0 or NULL
All statistics are updated together. This is the faster method. However, if the statistics for one nickname are invalid, all nickname statistics are affected and catalog statistics will not be updated successfully.

1 Statistics are updated by nickname. If the statistics for one nickname are invalid, the statistics for that nickname remain in the catalog as default statistics. Statistics for all other nicknames are updated independently. This is the slower method.

Example 1: In this example, the federated server retrieves the statistics for the nickname STAFF in the ADMIN schema. Statistics are gathered for columns 1, 3, 4, 6, 7, and 10 and indexes 1 through 3. The data-based method is used to collect statistics. The federated server writes the log to the /home/iiuser/reportlogs/log1.txt file. Statistics are updated by nickname.

```
CALL SYSPROC.NNSTAT(
    NULL, 'ADMIN','STAFF','COL1, COL3, COL4, COL6, COL7, COL10',
    'IND1, IND2, IND3',2,'/home/iiuser/reportlogs/log1.txt',?,1)
```

Example 2: In this example, the federated server retrieves the statistics for all the nicknames on the DB2SERV server in the ADMIN schema. The federated server writes the log to the c:\\reports\\log1.txt file.

```
CALL SYSPROC.NNSTAT(
    'DB2SERV','ADMIN',NULL,NULL,NULL,0,'c:\\reports\\log1.txt',?)
```

Example 3: In this example, the federated server retrieves the statistics for all the nicknames on the DB2SERV server and does not create a log.

```
CALL SYSPROC.NNSTAT(
    'DB2SERV',NULL,NULL,NULL,NULL,0,NULL,?)
```

Automatic update of nickname statistics

Automatic statistics collection is a feature that collects up-to-date table and nickname statistics. This feature is enabled by default.

Before you begin

Before you begin

For automatic statistics collection to run against a data source, ensure that there is a user mapping defined for the instance owner to connect to the data source.

About this task

About this task

Automatic statistics collection is part of the automated table maintenance feature that determines when database statistics need to be updated. For nicknames, automatic statistics collection uses the catalog-based method of the nickname statistics (NNSTAT) stored procedure. The catalog-based method retrieves nickname statistics from catalog information at the remote site.

You can customize which table and nicknames are considered by automatic statistics collection. For example, you can customize which tables are considered by the automatic statistics collection feature and specifically include or exclude all nicknames, or nicknames on specific servers.

Automatic statistics collection is controlled through a maintenance policy. The same policy is used for automatic RUNSTATS and for statistics on nicknames. Sample files and examples are available that you can customize to create or update a policy.

If you do not want nickname statistics automatically collected, you can disable the automatic statistics collection feature or specify a policy to replace the default policy.

Procedure

To change the default behavior of automatic statistics collection for table and nicknames:

Procedure

- Optional: Create a policy for automatic table RUNSTATS operations. Use the system stored procedures SYSPROC.AUTOMAINT_SET_POLICY and SYSPROC.AUTOMAINT_SET_POLICYFILE.
- Optional: Collect automated maintenance policy information about automatic table RUNSTATS operations. Use the system stored procedures SYSPROC.AUTOMAINT_GET_POLICY and SYSPROC.AUTOMAINT_GET_POLICYFILE.
- Optional: Set the value of the AUTO_MAINT, AUTO_TBL_MAINT, and AUTO_RUNSTATS configuration parameters to OFF to disable automatic statistics collection.

Chapter 13. Federated three-part names

With federated three-part names, you can reference remote objects directly. When you reference remote objects directly, you do not need to create nicknames for the remote objects.

Federated three-part names are very useful for environments in which the federated server accesses a large amount of remote tables. In previous releases, in addition to setting up the servers, you had to create nicknames for all of the tables that you want to access. In version 10.1, federated three-part names allow you to issue SQL statements against any data that you are authorized to access in a set of servers. When you use federated three-part names, you no longer need to create nicknames.

The metadata for remote objects referenced in federated three-part names is retrieved at runtime. This feature makes improves synchronization of locally stored metadata with remote servers.

Federated three-part table names are supported for all relational data sources.

Support and considerations for federated three-part names

When working with federated three-part names, you need to be aware of supported function and environments, and when to use federated three-part names or nicknames.

Support for federated three-part names

Federated three-part names are supported for all relational data sources in the following environments:

- DB2 pureScale
- Massively parallel processing (MPP)
- Serial mode

The following SQL statements support federated three-part table names specified for relational data sources:

- SELECT
- INSERT, UPDATE, DELETE
- Positioned updates and deletes
- CREATE VIEW
- MERGE (the remote table cannot be a target table)
- UNION, INTERSECT

These SQL statements are supported on the Export utility

Considerations: When to use federated three-part names or nicknames

For the following functions, you need to use nicknames instead of federated three-part names:

- Referencing remote objects for nonrelational data sources

Federated three-part names are supported for relational data sources only.

- Defining informational constraints and creating indexes.

  These functions are not supported for federated three-part names.
- Controlling authorizations and privileges at the object level.

  You cannot control privileges for federated three-part names.
- Updating statistics.

  Statistics are collected when you use federated three-part names to access the remote tables. However, because the metadata for remote objects is not in the federated database catalog, you cannot update statistics.

In the following situation, you can use nicknames or federated three-part names:

- For altering a column data type to a data type other than the default mapping, you need to use nicknames. As an alternative, you can create views over the federated three-part names or use casting functions in DML statements.

Federated cache for federated three-part names

The federated cache is a catalog cache that stores metadata and statistics for remote objects.

When you issue an SQL statement against a remote table or view that is referenced by a federated three-part name, the metadata and statistics for the remote object are stored in the federated cache on the first reference. Subsequent queries that reference the same three-part name, in the same session or in other sessions, will obtain metadata directly from the federated cache.

The metadata information in a cache entry is associated with an expiration duration. The expiration duration is the interval at which the metadata is considered valid and up-to-date since it was loaded from a remote server. The default value of the expiration duration is zero (0). A zero value means that metadata is always considered validated, unless you choose to explicitly invalidate the data.

You can change the interval value by setting the environment variable FEDCACHE_EXPIRE_INTERVAL. You specify this value in units of seconds.

Recommendation: Set the FEDCACHE_EXPIRE_INTERVAL variable in the db2dj.ini file.

Examples

You can flush the cache to keep cached entries up-to-date by issuing the following commands:

- To invalidate specific three-part name metadata:

  ```
  FLUSH FEDERATED CACHE FOR rudb.rschema.t1
  ```
- To invalidate all three-part name metadata in a specific schema named rschema:

  ```
  FLUSH FEDERATED CACHE FOR rudb.rschema.*
  ```
- To invalidate all three-part name metadata in a specific server named rudb:

  ```
  FLUSH FEDERATED CACHE FOR rudb.*.*
  ```
- To flush a server named rudb:

  ```
  FLUSH FEDERATED CACHE FOR SERVER rudb
  ```

Specifying federated three-part names

A federated three-part name represents a remote object. The three-part name consists of a server name, a remote schema name, and a remote object name.

Description

In version 10.1, a federated three-part name differs from three-part names supported in previous releases. In previous releases, a three-part name referred to a table name consisting of database name, a schema name, and a table identifier in the format *db.schema.table*.

A federated three-part name consists of the following elements:

server_name
A server name that is assigned to the data source server in the CREATE SERVER statement.

remote_schema
The remote schema name to which the object belongs.

object_name
The object name of the remote object that you want to access.

You specify a federated three-part name in the following format:
server_name.remote_schema.remote_object

Alternatively, you can specify:

remote_schema.remote.table@server_name.

To specify this syntax, you need to set the DB2_COMPATIBILITY_VECTOR registry variable to the bit 0x20000 or to ORA. When this variable is set, the "@" in a table, view, or column name is treated as a delimiter.

Authorization

The privileges held by the authorization ID at the data source must include the privilege to select data from the object that the federated three-part name represents.

The remote object metadata for objects referenced in federated three-part names is stored in the federated cache instead of the federated database catalog. You do not need privileges previously required for nicknames when you reference federated three-part names in SQL statements.

Notes®

You can create a server with the same name as the local database name. If these names are equivalent, when the server name is used in federated three-part name, the three-part name is treated as a local name. The local name is used to find the local table.

- If the local table is not found, error SQL0204N is issued.
- If the first part of the three-part name does not match the local database name, the SYSCAT.SERVERS catalog is checked to verify if the name matches an existing server name.
- If a match for the server name is not found, error SQL0204N is issued.

For data sources that do not support schemas, you can specify a remote object as a two-part name, for example *server.table*. Any two-part name, for example P1.TABLE, is first resolved to the local object as *schema_name.object_name*.

- If the attempt to resolve the name fails, the federated server attempts to resolve the name to a federated object as *server_name.object_name*.
- If both attempts fail, error SQL0204N is issued.

Recommendation: Do not create a server with the same name as a local schema name or a local database name.

Examples

Example 1: In this example, a DRDA wrapper was created and a server named SUDB was created for a remote database named REMOTE. You want to run a query against a remote table named EMPLOYEE. The table is in the schema named RSCHEMA in database REMOTE. Instead of creating a nickname over the EMPLOYEE table, you can query the remote table directly by using a federated three-part name..

```
SELECT birthdate FROM sudb.rschema.employee WHERE firstname='SAM'

SELECT sudb.rschema.employee.birthdate FROM sudb.rschema.employee
WHERE sudb.rschema.employee.firstname='SAM'
```

Similarly, you can issue UPDATE, INSERT, and DELETE statements against the remote table:

```
UPDATE sudb.rschema.employee SET firstname='MARY'
INSERT INTO sudb.rschema.employee VALUES  ('Bob')
DELETE FROM sudb.rschema.employee
```

Example 2: In this example, a view is created based on a query that contains nicknames:

```
CREATE VIEW v_tpn AS (SELECT c1, c2 FROM sudb.rschema.employee)
```

When the schema of the remote object changes, for example if column c2 is dropped, the view remains valid. However, this change will result in an error when column c2 is referenced. For example, after c2 is dropped from the remote table employee, the following query is issued:

```
select c1 from v_tpn
```

This query will succeed. However, the following query referencing c2 will fail:

```
select c1, c2 from v_ptn
```

Restrictions on federated three-part names

Federated three-part names are supported for relational data sources. They are not supported for any nonrelational data sources.

The following restrictions also apply to federated three-part names:

- Functions that do not support nicknames do not support federated three-part names
- You can specify federated three-part names on supported SQL statements only. See “Support and considerations for federated three-part names” on page 159.
- Static SQL is not supported
- Statistical views are not supported
- Materialized query tables are not supported

The following statements do not support federated three-part names:

- ALTER TABLE
- COMMENT
- CREATE ALIAS
- CREATE FUNCTION (Both SQL and external)
- CREATE INDEX
- CREATE METHOD
- CREATE PROCEDURE (SQL, external and sourced)
- CREATE TABLE
- CREATE TRIGGER
- CREATE VARIABLE
- GRANT
- IMPORT (as target table) LOAD
- LOCK TABLE
- RENAME
- REVOKE
- Statements with extended indicator variables

Chapter 14. Working with remote XML data

Federation supports the remote XML data type that gives you the ability to access and manipulate XML data in a DB2 Database for Linux, UNIX, and Windows database.

The federated system adheres to the same XML semantics as the DB2 database system. The native XML data store provides the ability to store and access XML documents stored in hierarchical form as a column of a relational table. You define the XML column with the XML data type.

You can create a relational nickname over a remote table or view that contains the XML data type. You can also use the XML wrapper to create a nonrelational nickname over an XML document.

You can then use the nickname in the XQuery and SQL languages. The XQuery language is the primary mechanism for querying XML documents. You can use SQL to perform basic operations such as selecting XML columns and inserting, updating, or deleting XML data. You can also integrate SQL and XQuery to create queries for both existing relational data and XML data by using SQL/XML functions and predicates and XQuery functions.

Creating a nickname for remote XML data - examples

To work with remote XML data, you need to create a nickname for a remote table that contains XML data or for an XML document.

Example: Create a relational nickname that corresponds to a remote DB2 table named XMLTABLE1. Table XMLTABLE1 contains column XMLCOL that is defined with the XML data type:

```
CREATE NICKNAME NNXML1 FOR SERVER1.SCHEMA1.XMLTABLE1;
```

Example: Create a nonrelational nickname for an XML document by using the XML wrapper:

```
CREATE NICKNAME NNXML2
   (file_path VARCHAR(64) OPTIONS(DOCUMENT 'FILE'),
  doc XML)
FOR SERVER XML_SERVER;
```

Manipulating XML data - examples

After you create a nickname, you can query and manipulate XML data in the remote table or XML document on the federated system. The federated system supports all of the data manipulation operations for XML data that the DB2 database system supports.

The following examples show the different methods that you can use to work with XML data by using the nickname NNXML1.

Manipulating XML data with SQL

With SQL you can perform basic select, insert, update, and delete operations.

Example: Use the SELECT and INSERT statements to access and insert XML data from the nickname NNXML1.

In the following SELECT statement, the XML document is selected on the federated server:

```
SELECT xmlcol FROM NNXML1;
```

In the following INSERT statement a string is inserted into a nickname XML column:

```
INSERT INTO NNXML1 (xmlcol) VALUES ('<a><b>My data</b></a>');
```

Manipulating XML data with SQL/XML functions

With SQL/XML functions, you can perform a range of operations that include querying, validating, and publishing XML data.

Example: Use the XMLVALIDATE function to validate XML data by using the XML schema that is obtained from the schema specification in the XML instance document:

```
SELECT XMLVALIDATE(xmlcol)FROM NNXML1;
```

Manipulating XML data with XQuery

You can use XQuery functions to manipulate XML data, such as the xmlcolumn function.

Example: Use XQuery to retrieve element b, which is a child of root a, from the XML column named XMLCOL:

```
xquery db2-fn:xmlcolumn('NNXML1.XMLCOL')/a/b;
```

Validating XML documents against XML schemas

You can verify that an XML document is valid. To validate the XML, you need to register the XML schema and explicitly invoke the XMLVALIDATE function. Validation is optional, but recommended.

Registering XML schemas

Schema registration is required for XML validation. You must also register the schema before you perform annotated schema decomposition.

About this task

You register an XML schema in the XML schema repository (XSR). The registration process creates an XSR object.

XML schema registration occurs on the federated server.

Procedure

1. Register the XML schema document in the XSR by using stored procedures or by using commands through the command line processor.
2. Optional: Specify additional XML schema documents to include with the XSR object if the XML schema consists of more than one schema document.
3. Complete the registration process with the XSR.

Validating XML documents

To validate a particular XML document, you must explicitly invoke the XMLVALIDATE function.

About this task

You typically validate XML data during an insert or update operation by using the XMLVALIDATE function. The XMLVALIDATE function validates an XML value against an XML schema or against the XML schema that is obtained from the schema specification in the instance document.

When you insert or update an XML column in a nickname and use the XMLVALIDATE function, validation occurs on the federated server. The federated server then serializes the data and sends it to the data source. During serialization, any data type annotations added by XMLVALIDATE are not preserved.

Procedure

Invoke the XMLVALIDATE function as part of an SQL statement.

Results

- If the XMLVALIDATE is called without a schema specification, the schema is determined based on the *schemaLocation* attribute in the instance document. If no schema information exists, an error message is issued.
- If XMLVALIDATE is called with a registered schema or with a specific Uniform Resource Identifier (URI), validation is performed against that specific schema. If an external schema is specified, the external schema overwrites the internal schema specification.

Example

Example: Validate an XML document using the XML schema named MYSCHEMA.MYDOCUMENTS:

```
INSERT INTO NNXML1(XMLCOL)
  VALUES (
   XMLVALIDATE(
    ? ACCORDING TO XMLSCHEMA ID MYSCHEMA.MYDOCUMENTS
  )
)
```

If the XML schema that is associated with the SQL identifier MYSCHEMA.MYDOCUMENTS exists in the XML repository, the input XML value is validated.

Decomposing XML documents with annotated XML schemas into nicknames

With annotated XML schema decomposition, you can decompose documents that are based on annotations specified in an XML schema.

About this task

The annotated schema documents must be stored in and registered with the XML schema repository (XSR).

In cases where you need to access XML data as relational data, rather than in its hierarchical form, you can decompose, or shred, the XML data. You can decompose an XML document into a nickname that references a remote table.

Procedure

1. Annotate the schema documents with XML schema decomposition.
2. Create the nickname that you want to use for decomposition. The name of the nickname must match the values in the annotated schema.
3. Register the schema using the REGISTER XMLSCHEMA command.
4. Grant USAGE privileges on the XSR object to allow specific users to use a specific XML schema.
5. Enable the schema for decomposition using the ALTER XSROBJECT statement.
6. Use the DECOMPOSE XML DOCUMENT command to decompose the XML instance document.

Federated processing of remote XML data

XML data is sent and received between the federated system and the remote data source client as a serialized XML string in character or binary format.

The processes that affect how the federated server sends and receives XML data are serialization, parsing, and code page conventions.

Federated parsing of remote XML data

The federated system uses the DB2 XML parser to process remote XML data.

The parser requires well-formed XML data that adheres to data encoding rules. The parser does not perform schema validation.

- During data retrieval, the parser issues an error message if XML data is not well-formed.
- During data insertion, depending on where parsing takes place, the federated server or the target data source issues an error message if XML data is not well-formed.

Whitespace handling for federation

During XML parsing, you can preserve or strip boundary whitespace in XML documents. The whitespace characters that occur between elements is *boundary whitespace*.

The federated server enforces the same whitespace conventions as the DB2 database system.

For application host variables and parameter markers, the special register CURRENT IMPLICIT XMLPARSE OPTION determines if whitespace is stripped during the federated bind operation. If the data source follows different rules for whitespace handling, the federated system attempts to compensate for the difference in semantics.

Code page issues for remote XML data

Certain code page issues can affect federated statements.

The federated server adheres to the DB2 XML parser encoding rules.

For federated statements, the following issues can occur when manipulating serialized remote XML data:

- For remote XML data that is serialized in binary format:
  - When data is sent from the remote client to the federated wrapper, no data loss occurs.
  - If an internal encoding exists that is not a valid IBM encoding scheme, the wrapper either replaces the encoding attribute with a valid IBM encoding scheme or removes the encoding attribute from the XML declaration and the DB2 XML parser decodes the value.
- For remote XML data that is serialized in character format:
  - The code page of the XML data is in the code page of the federated database. When the data source client sends data to the federated wrapper, data loss is possible. If the conversion results in a character substitution, a warning might be generated depending on the data source behavior and wrapper implementation.
  - If an internal encoding exists, the wrapper removes it because the serialized string is in the code page of the federated database.

For nonrelational wrappers, the DB2 XML parser decodes XML data using the internal encoding of the document.

Restrictions on the remote XML data type

Federated systems impose some restrictions on the remote XML data type.

You can only use the remote XML data type with the DB2 for Linux, UNIX, and Windows data source and the XML wrapper.

You cannot perform the following SQL operations:

- Alter the XML data type for a nickname, which results in error SQL0270N
- Create a federated index SPECIFICATION ONLY with the xml-index-specification clause, which results in error SQL0104N
- Define a federated stored procedure with the XML data type, which results in error SQL1254N
- Create a type mapping between the XML data type and another data type, which results in error SQL0604N

The following restrictions apply to XML columns for the XML wrapper:

- You cannot specify any column options for an XML column.
- Only the root nickname can contain a single XML column that references the entire contents of an XML document.
- Each root nickname must correspond to a single XML document.
- Child nicknames, nickname XPath options, and nickname namespaces options are not relevant.

Chapter 15. Error tolerance in nested table expressions

Error tolerance is a mechanism that allows query execution to continue while tolerating certain SQL errors in nested table expressions. With error tolerance, instead of receiving an error for a part of a query and terminating the entire query, you can obtain maximum query results from available data.

When the federated server encounters an allowable error, the server allows the error and continues processing the remainder of the query rather than returning an error for the entire query. The result set that the federated server returns can be a partial or an empty result.

When the federated server tolerates errors, it returns query results even when the data sources that the query accesses are not available. This mechanism is useful when you need to return as much information as is available, despite incomplete query results. For example, consider a doctor who needs data about a particular type of medical condition. A query is run to gather information from remote data sources at several different hospitals. If one or more hospital databases are not available, the results from only the available databases are still very valuable to the doctor.

Queries that contain UNION ALL branches can benefit from error tolerance. Without this mechanism, if processing of one branch of the query encounters an error, the federated server stops processing the query. With this mechanism, when you specify the error to tolerate on that same branch of the query, the federated server tolerates the error and continues to navigate to the rest of the available branches. The UNION ALL operation returns the results from any available data sources.

Example: The following query selects data from three nicknames on three different servers:

```
SELECT c1 from nickname1_on_server1
UNION ALL
SELECT c1 from nickname2_on_server2
UNION ALL
SELECT c1 from nickname3_on_server3
```

When nickname2_on_server2 is not available, or when the remote server server2 is not available during query processing, you can obtain the result set from nickname1_on_server1 and from nickname3_on_server3 by tolerating the errors with nickname2 and server2. A result set from two of the three query branches is equivalent to running the following query:

```
SELECT c1 from nickname1_on_server1
UNION ALL
SELECT c1 from nickname3_on_server3
```

You can specify the SQL errors that you want to allow in a nested table expression during query processing. The types of errors that the federated server tolerates are errors with remote connections, authorization, and authentication.

Specifying nested table expressions for error tolerance

You specify the errors to tolerate in a nested table expression with the RETURN DATA UNTIL clause.

About this task

When you use the RETURN DATA UNTIL clause, you must specify the error codes that you want to tolerate. The following table lists the errors that are allowed in the *specific-condition-value* clause. You must specify an SQLSTATE, or an SQLSTATE and SQLCODE, that matches a permissible error code. The SQLCODEs listed in the table are required.

Table 18. Errors allowed in the specific-condition-value clause

SQLSTATE	Error code	SQLCODE
08001	SQL30080N	-30080
08001	SQL30081N	-30081
08001	SQL30082N	-30082
08001	SQL1336N	-1336
08004	Any	Any
28000	Any	Any
42501	Any	Any
42512	Any	Any
42704	SQL0204N	-204
42720	Any	Any

Procedure

To specify nested table expressions for error tolerance, create an SQL statement that contains the RETURN DATA UNTIL clause.

```
RETURN DATA UNTIL specific-condition-value
```

RETURN DATA UNTIL
Any rows that are returned from the fullselect, before the specified condition is encountered, are returned in the result set from the fullselect.

specific-condition-value
Specifies the condition and values for error tolerance.

FEDERATED
Required keyword. The specific condition that you specify must only include errors that occur at a federated data source.

SQLSTATE VALUE *string-constant*
You can specify a specific condition as an SQLSTATE value. The length of the string constant must be 5 when VALUE is specified. An SQLSTATE value can be narrowed down to a particular SQLCODE value. You can specify multiple SQLCODE values that share the same SQLSTATE in one *specific-condition-value*.

Nested table expressions for error tolerance - example

The following examples illustrate how to use the RETURN DATA UNTIL clause to return query results when one or more federated data sources are not available.

Example: The following SQL statement selects data from three different servers: SQLServer, Oracle, and Sybase.

```
SELECT c1 FROM
TABLE RETURN DATA UNTIL
FEDERATED SQLSTATE '08001' SQLCODE -30080, -30082
WITHIN(SELECT c1 FROM n1_from_SQLServer) etq1
UNION ALL
SELECT c1 FROM
TABLE RETURN DATA UNTIL
FEDERATED SQLSTATE '08001' SQLCODE -30080, -30082
WITHIN (SELECT c1 FROM n2_from_Oracle) etq2
UNION ALL
SELECT c1 FROM
TABLE RETURN DATA UNTIL
FEDERATED SQLSTATE '08001' SQLCODE -30080, -30082
WITHIN(SELECT c1 FROM n3_from_Sybase) etq3;
```

Scenario 1: One server is not available.

In this scenario, the Oracle server is not available. The SQLServer server and Sybase server are available. In this situation, the query in the second branch of the UNION ALL operation returns an empty result set with the 30080 error, which is specified to be tolerated. The query returns the results from n1_from_SQLServer and from n3_from_Sybase. Warning sqlwarn5='E' is issued.

The result set is equivalent to running the following query:

```
SELECT c1 FROM n1_from_SQLServer
UNION ALL
SELECT c1 FROM n3_from_Sybase;
```

Scenario 2: All servers are not available.

In this scenario, all servers (SQLServer, Oracle, and Sybase) are unavailable. In this situation, the UNION ALL operation returns an empty result set. Warning sqlwarn5='E' is issued.

Scenario 3: All servers are available.

If all of the servers are available, the result set of the query is equivalent to running the same query without specifying the RETURN DATA UNTIL clause.

Data source support for nested-table-expressions for error tolerance

Error tolerance is supported for several relational data sources and for nonrelational nicknames.

Error tolerance in nested-table-expressions is supported for the following relational data sources:

- DB2 family (DRDA)
- Informix
- JDBC
- Microsoft SQL Server
- ODBC
- Oracle (NET8)
- Sybase (CTLIB)
- Teradata

You can use nonrelational nicknames within nested-table-expressions for error tolerance. The federated server can tolerate the allowable connection, authentication, or authorization errors when the nonrelational wrappers return a permissible error code.

Restrictions on nested-table-expressions for error tolerance

Some restrictions apply when you define error tolerant nested-table-expressions.

A query or view is read-only when you define the query or view with an expression that contains the RETURN DATA UNTIL clause. Cursors that are declared in expressions with the RETURN DATA UNTIL clause are read-only. Errors are returned for each of those situations.

If a connection failure occurs, an initial connection failure is tolerated. An initial connection failure occurs during a connection attempt to a data source. An attempt to connect to a data source occurs the first time a query is issued to a data source within a connection or after a connection loss. A connection loss or failure that is not an initial connection failure is not tolerated.

Chapter 16. High availability for federation

Federation provides a high availability feature. High availability disaster recovery (HADR) is a database replication feature that protects against data loss by replicating data changes from a source database to a target database.

The source database is also referred to as the *primary* database. The target database is also referred to as the *standby* database. When the primary database is unavailable or fails, the standby database takes over transaction processing. The standby database then becomes the new primary database. This transfer of workload from the primary to the standby database is called *failover*. When failover occurs, the client's database connection is switched to the new primary database through the automatic client reroute process.

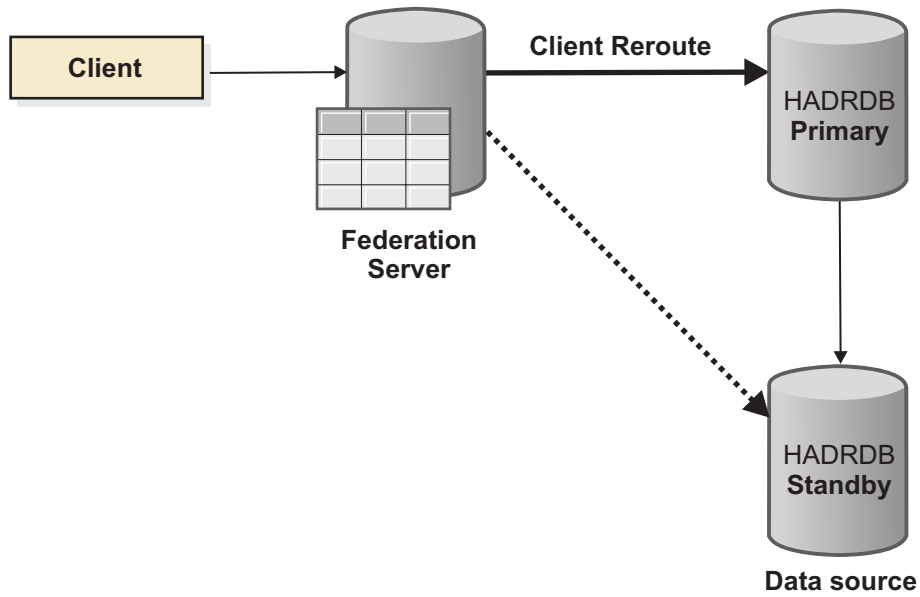
In a federated system, the HADR database can act as either the federated database or as the data source database.

High availability disaster recovery (HADR) and federated databases

A HADR database can act as the federated database.

When the HADR database acts as a federated database, if the primary federated database becomes unavailable, the standby database takes over transaction processing. If the client reroute process is enabled for the federated database, the client can automatically switch the database connection to the new primary database.

In this configuration, the data source can be any of the data sources that federation supports. The following figure depicts this configuration.



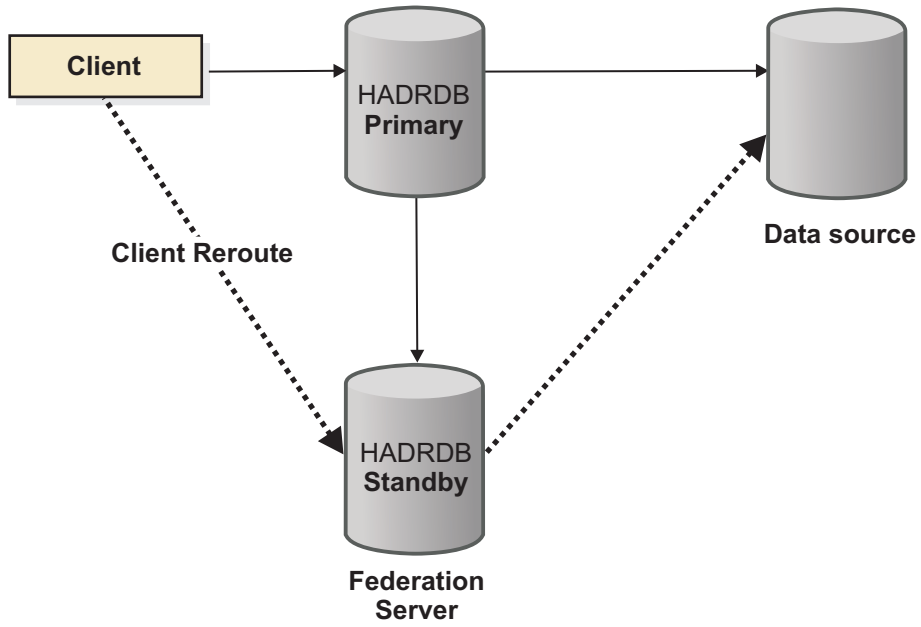
High availability and data source databases

Federation can work with data sources that have a high availability feature.

If the data source is DB2 Database for Linux, UNIX, and Windows, you can configure high availability disaster recovery (HADR) and automatic client reroute (ACR) on the data source database. The federation server is the client of the data source. At the data source, if the primary database becomes unavailable, the standby database takes over transaction processing. By using automatic client reroute (ACR), the federation server can automatically connect to the new primary data source database and continues processing the main functions that the federation supports.

For other data sources with high availability features, whether the federated server can automatically connect to the new active data source database after failover depends on the data source.

The following figure depicts this configuration.



Configuring high availability disaster recovery (HADR) for federation

You can configure the high availability feature on a data source database, such as HADR for the DB2 Database for Linux, UNIX, and Windows data source, or configure high availability disaster recovery on IBM InfoSphere Federation Server.

Configuring HADR for a federated database

To configure a federated database for HADR, you need to perform the required steps to set up the standby and the primary databases.

About this task

It is important to ensure the consistency between the primary database and the standby database when you configure HADR on the federated database. You should not make any changes to the federated database, (that is, DDL or DML) during the configuration.

Configuration of automatic client reroute is recommended for a federated database acting as the high availability database.

Procedure

1. Choose an existing federated database, or create the federated database by following the procedure to create a federated database and create each of the following objects on the primary database:
   a. Create a wrapper by using the CREATE WRAPPER statement.
   b. Create a server by using the CREATE SERVER statement.
   c. Create a user mapping by using the CREATE USER MAPPING statement.
   d. Create a nickname by using the CREATE NICKNAME statement.
2. Backup the database on the primary database server and restore the database on the standby database server. You must specify `logretain on` for the HADR database. For example:

   On the primary server:

   ```
   db2 update db cfg for feddb using logretain on

   db2 backup db feddb to /directory
   ```

   On the standby server:

   ```
   db2 restore db feddb from /directory
   ```

3. Configure the standby database for HADR. For example:

   ```
   db2 UPDATE DB CFG FOR feddb USING HADR_LOCAL_HOST hostnamest;
   db2 UPDATE DB CFG FOR feddb USING HADR_REMOTE_HOST hostnampr;
   db2 START HADR ON DB feddb AS STANDBY;
   ```

4. Configure the primary database for HADR. For example:

   ```
   db2 UPDATE DB CFG FOR feddb USING HADR_LOCAL_HOST hostnamepr;
   db2 UPDATE DB CFG FOR feddb USING HADR_REMOTE_HOST hostnamest;
   db2 START HADR ON DB feddb AS PRIMARY;
   ```

   For additional configuration information, see Database configuration for high availability disaster recovery (HADR).

   What to do next

   After HADR is configured on the federated database, you can issue operations such as federated DDL or DML. For example:

   - Create another nickname
   - Update rows by using any nicknames

   If the primary database fails, you can issue the TAKEOVER HADR command on the standby database. The standby database then becomes the new primary database that can take over transaction processing.

Configuring high availability for a data source database

To configure a high availability database as a data source, you need to perform the required steps to set up the standby and the primary databases.

About this task

It is important to ensure the consistency between the primary database and the standby database when you configure high availability on the data source database. You should not make any changes to the federated database or the data source database (that is, DDL or DML) during the configuration.

When the data source is DB2 Database for Linux, UNIX, and Windows, configuration of automatic client reroute is required for a data source database acting as the high availability database. The federated server requires automatic client reroute to automatically establish a connection to the new primary database after takeover.

The following procedure provides configuration instructions for the DB2 Database for Linux, UNIX, and Windows data source.

Procedure

1. Choose an existing data source database, or create the data source database on primary database server by using the CREATE DATABASE statement. For example:

   ```
   db2 create db ds
   create table t11 (i int, c char(20));
   insert into t11 values (11, 'dsds');
   ```

2. Backup the database on the primary database server and restore the database on the standby database server. You must specify `logretain on` for the high availability database. For example:

   On the primary server:

   ```
   db2 update db cfg for ds using logretain on

   db2 backup db ds to /directory
   ```

   On the standby server:

   ```
   db2 restore db ds from /directory
   ```

3. Configure the standby database for high availability and automatic client reroute. For example:

   ```
   db2 UPDATE DB CFG FOR ds USING HADR_LOCAL_HOST hostnamest;
   db2 UPDATE DB CFG FOR ds USING HADR_REMOTE_HOST hostnampr;
   db2 START HADR ON DB ds AS STANDBY;
   db2 UPDATE ALTERNATE SERVER FOR DATABASE ds USING HOSTNAME hostnamepr PORT port1
   ```

4. Configure the primary database for high availability and automatic client reroute. For example:

   ```
   db2 UPDATE DB CFG FOR feddb USING HADR_LOCAL_HOST hostnamepr;
   db2 UPDATE DB CFG FOR feddb USING HADR_REMOTE_HOST hostnamest;
   db2 START HADR ON DB ds AS PRIMARY;
   db2 UPDATE ALTERNATE SERVER FOR DATABASE ds USING HOSTNAME hostnamest PORT port2
   ```

   For additional information about configuring high availability, see Database configuration for high availability disaster recovery (HADR). For additional information about configuring automatic client reroute, see Configuring automatic client reroute and High Availability Disaster Recovery (HADR).

   What to do next

   After HADR is configured on the data source database, you can create a federated database by following the procedure to create a federated database and create each of the following objects on the federated database for accessing the data source database with high availability:

   a. Create a wrapper by using the CREATE WRAPPER statement.

b. Create a server by using the CREATE SERVER statement.
c. Create a user mapping by using the CREATE USER MAPPING statement.
d. Create a nickname by using the CREATE NICKNAME statement.

You can also access the data source database with high availability though nicknames on an existing federated database.

On the data source, if the primary database fails, you can issue the TAKEOVER HADR command on the standby database. The standby database then becomes the new primary database that can take over transaction processing. The federation server automatically switches the connection to the new primary database.

Restrictions on high availability disaster recovery (HADR) for federation

The same high availability restrictions that apply to DB2 Database for Linux, UNIX, and Windows also apply to federation. Some restrictions apply to specifically to federation support for HADR.

The following restrictions apply to a HADR database that is configured as a federated database:

- Transparent DDL is not supported.
- The reads on standby feature is not be supported.
- The standby database must contain the same clients of data sources as the primary database. Otherwise, access to nicknames fail after the standby database takes over.

Chapter 17. Monitoring federated servers and nicknames

You can use the health monitor and systems monitor elements to monitor a federated system.

Health indicators for federated nicknames and servers

You can use health indicators to monitor the status of your federated nicknames and servers.

The health indicator for nicknames is db.fed_nicknames_op_status. The health indicator for server definitions is db.fed_servers_op_status. The federated health indicators are installed when the health monitor is installed.

When the state of a nickname or server is not normal, the health indicators issue an alert. You can view the results of monitoring by using command line.

Federated servers that use AIX, HP-UX, Linux, Microsoft Windows, and Solaris operating systems support the health indicators.

Table 19 describes the health indicators for federated nicknames and servers.

Table 19. Nickname and server health indicators

Health indicator	Description
db.fed_nicknames_op_status	Indicates the aggregate health of all the relational nicknames defined in a database on a federated server. Alerts you if a nickname is invalid. Provides details about the invalid nicknames and recommends actions that you can take to repair them.
db.fed_servers_op_status	Indicates the aggregate health of all the federated servers defined in a database on a federated server. Alerts you if a server is unavailable. Provides details about the unavailable servers and recommends actions that you can take to make them available.

The health indicators can evaluate the following data sources:

- DB2 family (DRDA)
- Excel
- Informix
- JDBC
- Microsoft SQL Server
- ODBC
- Oracle (NET8)
- Sybase (CTLIB)
- Table-structured files
- Teradata
- XML (root nicknames only)

Activating the federated health indicators

To monitor the health of nicknames and servers, you must activate the federated health indicators. The health indicator for nicknames is db.fed_nicknames_op_status. The health indicator for server definitions is db.fed_servers_op_status.

Procedure

To activate the federated health indicators, you can use the command line processor.

Monitoring the health of federated nicknames and servers

Monitoring nickname and server status can help you determine and resolve problems in your federated system.

Before you begin

- Ensure that SELECT privileges on the nicknames are defined on the federated server.
- Set the FEDERATED database manager configuration parameter to YES.
- If the data source requires authentication, the data source must have user mappings from the Health Monitor's ID.

About this task

You can view the results of monitoring by using the command line. Use the command line processor to resolve the problems that are identified by the health indicators.

Restrictions

"Health indicators for federated nicknames and servers" on page 181 lists the data sources that the health indicators can evaluate.

Procedure

To do this task from the command line, issue the GET HEALTH SNAPSHOT command.

Monitoring the health of federated nicknames and servers - example

This topic provides an example of the health snapshot of a database.

The federated health indicator names are db.fed_nicknames_op_status and db.fed_servers_op_status. You must enable these health indicators by using the following commands in the CLP:

```
db2 update alert cfg for databases using db.fed_nicknames_op_status set
 THRESHOLDSCHECKED YES
db2 update alert cfg for databases using db.fed_servers_op_status set
 THRESHOLDSCHECKED YES
```

The following command retrieves a database health snapshot including the federated health indicators, if they have been enabled:

```
db2 get health snapshot for database on <database_name>
```

In this example, the database name is fedhi. The output of this command indicates that both health indicators are in normal states. Normal means that the nicknames and the servers are valid.

```
            Database Health Snapshot

Snapshot timestamp                         = 02/10/2006 12:10:55.063004

Database name                              = FEDHI
Database path                              = C:\DB2\NODE0000\SQL00006\
Input database alias                       = FEDHI
Operating system running at database server= NT
Location of the database                   = Local
Database highest severity alert state      = Attention

Health Indicators:

    Indicator Name                           = db.fed_servers_op_status
       Value                                 = 0
       Evaluation timestamp                  = 02/10/2006 12:09:10.961000
       Alert state                           = Normal

    Indicator Name                           = db.fed_nicknames_op_status
       Value                                 = 0
       Evaluation timestamp                  = 02/10/2006 12:09:10.961000
       Alert state                           = Normal

    Indicator Name                           = db.db_op_status
       Value                                 = 0
       Evaluation timestamp                  = 02/10/2006 12:08:10.774000
       Alert state                           = Normal

    Indicator Name                           = db.sort_shrmem_util
       Value                                 = 0
       Unit                                  = %
       Evaluation timestamp                  = 02/10/2006 12:08:10.774000
       Alert state                           = Normal


    Indicator Name                           = db.spilled_sorts
       Value                                 = 0
       Unit                                  = %
       Evaluation timestamp                  = 02/10/2006 12:09:10.961000
       Alert state                           = Normal
```

Snapshot monitoring of federated systems - Overview

You can use the snapshot monitor to capture information about federated data sources and any connected applications at a specific time.

Snapshots are useful for determining the status of a federated system. Taken at regular intervals, snapshots are also useful for observing trends and foreseeing potential problems.

The output of the snapshot monitor is available in the following formats:

- In textual form, through the snapshot monitor command-line processor interface.
- As the output of table functions. This output is useful for writing queries that limit output.

The snapshots that are particularly useful for federated workloads include:

Dynamic SQL statement snapshot
Provides a snapshot of all dynamic SQL statements currently in the statement cache that includes federated and non-federated statements.

Application snapshot
Provides information about a specific application, including the text of the currently running SQL statement.

Remote databases snapshot
Provides information about a specific federated system database.

All remote databases snapshot
Provides information about each federated system database that is active.

Remote applications snapshot
Provides application-level information for each federated system application that is active.

Monitoring federated queries

By monitoring queries, you can determine how your federated system is performing. To help you understand how your federated system is processing a query, you can get a snapshot of the remote query.

Before you begin

You must set the STATEMENT monitor switch ON for the federated database to collect snapshot information for remote queries.

About this task

The snapshot monitor tracks two aspects of each query processed by the federated server:

- The entire federated query as submitted by the application, which references nicknames, local tables, or both.
- For queries involving nicknames, one or more *remote fragments*. Remote fragments are the statements that are automatically generated and submitted to remote data sources in their native dialects on behalf of the federated query.

Monitoring federated queries requires that you consider both the work done locally at the federated server and work done at remote servers in response to remote query fragments. The dynamic SQL statement snapshot and the SNAPSHOT_DYN_SQL table function contain information about individual federated queries as they are submitted to the federated server, and about remote query fragments that the federated server, sends to other data sources.

Procedure

To monitor queries on the federated server, while connected to the federated database, use one of the following methods:

- Textual output:

  ```
  GET SNAPSHOT FOR DYNAMIC SQL on dbname
  ```

  where *dbname* is the name of the federated server database.
- Table function:

```
CREATE TABLE table snap AS (SELECT * FROM TABLE(SNAPSHOT_DYN_SQL ('dbname', -1))
 as snaptab) definition only;
INSERT INTO snap (SELECT * FROM TABLE(SNAPSHOT_DYN_SQL ('dbname', -1))
 as snaptab);
```

You can then write a query against the snap table that contains one row per query (federated or non-federated) and one row per query fragment in the statement cache of the server.

The name of remote query fragments is the server to which they were sent prepended to the remote query text, in square brackets, in the stmt_text field of the table function. For example, you can use the following query to look for long-running remote fragments:

```
SELECT total_exec_time, rows_read, total_usr_cpu_time, num_executions,
 substr(stmt_text,1,30)
FROM TABLE(SNAPSHOT_DYN_SQL ('dbname', -1))AS snaptab
-- remote fragments only
WHERE stmt_text LIKE '[%]%'
ORDER BY total_exec_time;
```

By comparing the execution time of an entire federated statement with the execution times of remote fragments sent to other data sources on behalf of the statement, you can understand where most of the processing time is spent.
To determine which query fragments are sent to remote sources on behalf of a federated query, consult an EXPLAIN execution plan for the query.

Snapshot monitoring of federated queries - example

This topic provides an example of output for the text-based dynamic SQL snapshot of a federated query that involves a remote Oracle data source.

The following statement retrieves a snapshot of all statements currently in the statement cache, including federated statements and remote fragments sent to other data sources:

```
GET SNAPSHOT FOR DYNAMIC SQL ON <database_name>
```

The database name is the name of the local federated database.

The output in the following example is the result of the statement:

```
GET SNAPSHOT FOR DYNAMIC SQL ON FEDDB
```

The example shows a federated statement and one remote fragment that is pushed down by that federated statement. You can identify remote fragments by finding the remote server name prepended to the remote statement text in square brackets. In this example, the remote Oracle server is named ORA9. The first entry shows the federated SQL statement that references nicknames, including its overall elapsed time. The second entry shows the remote statement sent to the source [ORA9] that references the remote Oracle table names.

```
   Dynamic SQL Snapshot Result

 Database name                       = FEDDB

 Number of executions                = 1
 Number of compilations              = 1
 Worst preparation time (ms)         = 475
 Best preparation time (ms)          = 475
 Internal rows deleted               = 0
 Internal rows inserted              = 0
 Rows read                           = 5
 Internal rows updated               = 0
 Rows written                        = 0
```

```
Statement sorts                    = 0
Statement sort overflows           = 0
Total sort time                    = 0
Buffer pool data logical reads     = Not Collected
Buffer pool data physical reads    = Not Collected
Buffer pool temporary data logical reads   = Not Collected
Buffer pool temporary data physical reads  = Not Collected
Buffer pool index logical reads    = Not Collected
Buffer pool index physical reads   = Not Collected
Buffer pool temporary index logical reads  = Not Collected
Buffer pool temporary index physical reads = Not Collected
Buffer pool xda logical reads      = Not Collected
Buffer pool xda physical reads     = Not Collected
Buffer pool temporary xda logical reads    = Not Collected
Buffer pool temporary xda physical reads   = Not Collected
Total execution time (sec.ms)      = 1.816884
Total user cpu time (sec.ms)       = 0.000000
Total system cpu time (sec.ms)     = 0.020000
Statement text                     = select count(*) from orat.supplier,
         orat.nation where s_nationkey =
         n_nationkey and n_name <> 'FRANCE'


Number of executions               = 1
Number of compilations             = 1
Worst preparation time (ms)        = 0
Best preparation time (ms)         = 0
Internal rows deleted              = 0
Internal rows inserted             = 0
Rows read                          = 1
Internal rows updated              = 0
Rows written                       = 0
Statement sorts                    = 0
Statement sort overflows           = 0
Total sort time                    = 0
Buffer pool data logical reads     = Not Collected
Buffer pool data physical reads    = Not Collected
Buffer pool temporary data logical reads   = Not Collected
Buffer pool temporary data physical reads  = Not Collected
Buffer pool index logical reads    = Not Collected
Buffer pool index physical reads   = Not Collected
Buffer pool temporary index logical reads  = Not Collected
Buffer pool temporary index physical reads = Not Collected
Buffer pool xda logical reads      = Not Collected
Buffer pool xda physical reads     = Not Collected
Buffer pool temporary xda logical reads    = Not Collected
Buffer pool temporary xda physical reads   = Not Collected
Total execution time (sec.ms)      = 1.337672
Total user cpu time (sec.ms)       = 0.000000
Total system cpu time (sec.ms)     = 0.000000
Statement text                     = [ORA9] SELECT COUNT(*) FROM "TPCH"."NATION"
         A0, "TPCH"."SUPPLIER" A1 WHERE
         (A0."N_NAME" <> 'FRANCE                  ') AND
         (A1."S_NATIONKEY" = A0."N_NATIONKEY")
```

The snapshot did not collect any buffer pool information, because buffer pool information is not applicable to remote queries.

Federated database systems monitor elements

This topic describes the monitor elements that provide information about federated systems.

A federated system access to diverse data sources that can reside on different platforms, both IBM and other vendors, relational and nonrelational. It integrates access to distributed data and presents a single database image of a heterogeneous environment to its users.

The following elements list information about the total access to a data source by applications running in a federated system and information about access to a data source by a given application running in a federated server instance. They include:

- datasource_name - Data Source Name monitor element
- disconnects - Disconnects monitor element
- insert_sql_stmts - Inserts monitor element
- update_sql_stmts - Updates monitor element
- delete_sql_stmts - Deletes monitor element
- dynamic_sql_stmts - Dynamic SQL Statements Attempted monitor element
- create_nickname - Create Nicknames monitor element
- passthrus - Pass-Through monitor element
- stored_procs - Stored Procedures monitor element
- remote_locks - Remote Locks monitor element
- sp_rows_selected - Rows Returned by Stored Procedures monitor elements
- select_time - Query Response Time monitor element
- insert_time - Insert Response Time monitor element
- update_time - Update Response Time monitor element
- delete_time - Delete Response Time monitor element
- create_nickname_time - Create Nickname Response Time monitor element
- passthru_time - Pass-Through Time monitor element
- stored_proc_time - Stored Procedure Time monitor element
- remote_lock_time - Remote Lock Time monitor element

The following example shows the dynamic_sql_statement snapshot:

```
Statement text = [ORACLE817]SELECT A0.C1,A0.C2 FROM ORA_T A0 WHERE A0.C3 = :H0
```

For all remote statements, the Statement text starts with the remote data source name, inside square brackets, followed by the actual text sent to the remote data source.

Chapter 18. How client applications interact with data sources

To client applications, the data sources in a federated system appear as a single collective database. To obtain data from data sources, applications submit queries in DB2 SQL to the federated database. The federated database then distributes the queries to the appropriate data sources, and either returns this data to the applications or performs the requested action.

The federated database can join data from local tables and remote data sources in the same SQL statement. For example, you can join data that is located in a local DB2 table, an Informix table, and a Sybase view in a single SQL statement. By processing SQL statements as if the data sources were ordinary relational tables or views within the federated database, the federated system can join relational data and nonrelational data.

In a federated system, you can access data sources through nicknames. A *nickname* is a federated database object that an application uses to reference a data source object, such as a table or view. To write to a data source—for example, to update a data source table—an application can use DB2 SQL (with nicknames). Alternatively, applications can use the SQL dialect of the data source (without nicknames) in a special session called *pass-through* to access the data sources directly.

Applications that use DB2 SQL and nicknames can access any data types that the federated database recognizes.

The federated database catalog contains information about the objects in the federated database and information about objects at the data sources. Because the catalog contains information about the entire federated database, it is called a *global catalog*.

Chapter 19. Reference data source objects by nicknames in SQL statements

With a federated system, you use the nicknames defined for data source objects to represent the objects in your SQL statements. The federated system does not recognize fully-qualified data source, schema, and object names in SQL statements.

Data source objects must have nicknames registered in the federated database before you can include them in your queries. In general, you can specify nicknames in an SQL statement where you can specify local tables in a SQL statement.

Example: Using nicknames in SELECT, INSERT, UPDATE, and DELETE statements

You define the nickname NFXDEPT to represent a table in an Informix table called PERSON.DEPT, where:

- PERSON is the data source schema
- DEPT is the data source table name

The statement SELECT * FROM NFXDEPT is allowed from the federated server. However, the statement SELECT * FROM PERSON.DEPT is not allowed (except in a pass-through session). The federated server does not have PERSON.DEPT registered as a nickname.

Example: Using nicknames in the CREATE TABLE statement

You want to create a local table based on a remote table for which you have defined a nickname. An example of the CREATE TABLE statement is:

```
CREATE TABLE table_name LIKE nickname
```

Nicknames in DDL statements

Data source objects must have nicknames registered in the federated database before you can include them in your DDL statements. This topic provides some examples of DDL statements that you use with federated systems.

Using nicknames in the COMMENT ON statement

The COMMENT ON statement adds or replaces comments in the federated database global catalog. The COMMENT ON statement is valid with a nickname and columns that are defined on a nickname. This statement does not update data source catalogs.

Using nicknames in the GRANT and REVOKE statements

The GRANT and REVOKE statements are valid with a nickname for certain privileges and for all users and groups. However, the federated system does not issue a corresponding GRANT or REVOKE statement on the object on the data source that the nickname references.

For example, suppose that user JON creates a nickname for an Oracle table that had no index. The nickname is ORAREM1. Later, the Oracle DBA defines an index for this table. User EILEEN now wants the federated database to know that this index exists, so that the query optimizer can devise strategies to access the table more efficiently. EILEEN can inform the federated database that a new index exists, by creating an index specification for ORAREM1.

The information about the index is stored in the SYSSTAT.INDEXES catalog view. Use the GRANT statement to give EILEEN the index privilege on this nickname, so that she can create the index specification.

```
GRANT INDEX ON NICKNAME ORAREM1 TO USER EILEEN
```

To revoke user EILEEN's privileges to create an index specification on nickname ORAREM1, use the REVOKE statement:

```
REVOKE INDEX ON ORAREM1 FROM USER EILEEN
```

Data source statistics impact applications

When a nickname is created for a data source object, the federated database global catalog is updated with information about that object. The query optimizer uses this information to plan how to retrieve data from the object.

It is important to make sure that the data source information is current. The federated database does not automatically detect changes to data source objects.

Database object statistics stored in the global catalog

The information stored in the global catalog about a data source object, depends on the type of object. For database tables and views, the name of the object, the column names and attributes, are stored in the global catalog.

In the case of a table or nickname, the information also includes:

- Statistics. For example, the number of rows and the number of pages on which the rows exist. To ensure that the federated database obtains the latest statistics, run the data source equivalent of the RUNSTATS command on the table before you create the nickname.
- Index descriptions. If the table has no indexes, you can supply the catalog with metadata that an index definition typically contains. For example, assume that a nickname is created for a remote table, and that an index is subsequently created on the table at the data source. You can create an index specification at the federated server that represents this remote index. You create an index specification by issuing the CREATE INDEX statement and referencing the nickname for the table. You use the SPECIFICATION ONLY clause with the CREATE INDEX statement to produce only an index specification. The index specification informs the federated optimizer that a remote index exists. However, only metadata is generated. No index is actually created on the federated server. In addition, no statistical information is supplied to the global catalog. If you give the index specification exactly the same signature as the remote index (that is, the same name, and the same columns in the same order), you can use SYSPROC.NNSTAT to update statistics on the nickname and index specification.

To determine what data source information is stored in the global catalog, query the SYSCAT.TABLES and SYSCAT.COLUMNS catalog views. To determine what

data source index information is stored in the catalog, or what a particular index specification contains, query the SYSCAT.INDEXES catalog view.

Updating statistics using the SYSSTAT view instead of the SYSCAT view

SYSCAT views are read-only catalog views in the SYSCAT schema. SYSSTAT views are updatable catalog views that contain statistical information that the optimizer uses. SYSSTAT views are in the SYSSTAT schema.

If you issue an UPDATE or INSERT operation on a view in the SYSCAT schema, it will fail. Use the updatable catalog views in the SYSSTAT schema to manually modify statistics on nicknames.

Defining column options on nicknames

Column options are parameters in the CREATE NICKNAME and ALTER NICKNAME statements. You can specify column options when you initially create a nickname or by modifying an existing nickname.

The information that you provide through the column options is stored in the global catalog.

Nonrelational data sources

Column options are unique for each nonrelational wrapper. These options are typically set when you issue the CREATE NICKNAME statement.

Relational data sources

There are two column options you can use for relational data sources: NUMERIC_STRING and VARCHAR_NO_TRAILING_BLANKS.

Setting the NUMERIC_STRING column option

If a data source string column contains only numeric digits, and no other characters including blanks, set the NUMERIC_STRING column option to Y.

Setting the NUMERIC_STRING column option to Y allows queries that use this column to be optimized for sorting operations and comparison operations. For example:

```
ALTER NICKNAME nickname
   ALTER COLUMN local_column_name
   OPTIONS (SET NUMERIC_STRING 'Y')
```

Setting the VARCHAR_NO_TRAILING_BLANKS column option

If the data source string column does not contain trailing blanks, set the VARCHAR_NO_TRAILING_BLANKS column option to Y.

Some data sources, such as Oracle, do not use the same blank-padded string comparison logic that the federated database uses. This applies to data types such as VARCHAR and VARCHAR2. As a result, predicates that involve these data types must be rewritten by the query optimizer to ensure consistent query results.

Rewriting query statements can impact performance. Setting this option for a specific column provides the query optimizer with information about these columns so that it can generate more efficient SQL statements.

The first time that you add an option, use the ADD keyword to add the option. If the option was added previously, use the SET keyword to change the option.

For example:

```
ALTER NICKNAME nickname
   ALTER COLUMN local_column_name
   OPTIONS (ADD VARCHAR_NO_TRAILING_BLANKS 'Y')
```

Chapter 20. Creating and using federated views

A view in that includes a reference to a nickname in the fullselect is a *federated view*. The base tables are referenced in the federated view using nicknames, instead of using the data source table names.

About this task

Restrictions

Federated views that are created from multiple data source objects are read-only views and cannot be updated.

Federated views that are created from only one data source object might or might not be read-only views.

- A federated view created from a single nonrelational data source is read-only.
- A federated view created from a single relational data source might allow updates, depending on what is included in the CREATE VIEW statement.

About this task

The advantages of using federated views are similar to the advantages of using views defined on local tables in a centralized relational database manager:

- Views provide an integrated representation of the data
- You can exclude table columns that contain confidential or sensitive data from a view

Procedure

You create a federated view from data source objects that have nicknames. The action of creating a federated database view of data source data is sometimes called “creating a view on a nickname”. This phrase reflects the fact that for the federated view to be created, the CREATE VIEW statement fullselect must reference the nickname of each data source table and view that the federated view is to contain.

Creating federated views - examples

This topic provides examples of creating federated views.

Example: Creating a federated view that merges similar data from several data source objects

You are working with customer data on three separate servers, one in Europe, one in Asia, and one in South America. The Europe customer data is in an Oracle table. The nickname for that table is ORA_EU_CUST. The Asia customer data is in a Sybase table. The nickname for that table is SYB_AS_CUST. The South America customer data resides in an Informix table. The nickname for that table is INFMX_SA_CUST. Each table has columns containing the customer number (CUST_NO), the customer name (CUST_NAME), the product number (PROD_NO), and the quantity ordered (QUANTITY). The syntax to create a view from these three nicknames that merges this customer data is:

```
CREATE VIEW FV1
   AS SELECT * FROM ORA_EU_CUST
   UNION
   SELECT * FROM SYB_AS_CUST
   UNION
   SELECT * FROM INFMX_SA_CUST
```

Example: Joining data to create a federated view

You are working with customer data on one server and sales data on another server. The customer data is in an Oracle table. The nickname for that table is ORA_EU_CUST. The sales data is in a Sybase table. The nickname for that table is SYB_SALES. You want to match up the customer information with the purchases made by those customers. Each table has a column containing the customer number (CUST_NO). The syntax to create a federated view from these two nicknames that joins this data is:

```
CREATE VIEW FV4
   AS SELECT A.CUST_NO, A.CUST_NAME, B.PROD_NO, B.QUANTITY
   FROM ORA_EU_CUST A, SYB_SALES B
   WHERE A.CUST_NO=B.CUST_NO
```

Chapter 21. Maintain data integrity with isolation levels

The isolation level defines the degree of isolation for an application process from other application processes that are running concurrently.

You can maintain data integrity for a data source table by requesting that the table rows be locked at a specific isolation level.

Locking occurs at the base table row at the data source. The database manager, however, can replace multiple row locks with a single table lock. This action is called *lock escalation*. An application process is guaranteed at least the minimum requested lock level.

The isolation levels for the federated database are as follows:

RR Repeatable read

RS Read stability

CS Cursor stability (default)

UR Uncommitted read

The types of isolation are the statement level isolation and connection level isolation.

You can set the isolation when you perform the following actions:

- Precompile or bind an application. You can specify isolation levels when you prepare or bind an application. The isolation level specified in the BIND and PREP command is the default isolation level when the federated server connects to the remote data source.
- Use the WITH clause in an SQL statement. This action is called statement level isolation. You can use the WITH clause in the SELECT, UPDATE, INSERT, and DELETE statements.

If the federated server does not find an isolation level for a statement, the federated server uses the isolation level that was established when the federated server connected to the data source.

The following table lists the data sources that use connection level isolation, the isolation levels that they use, and the equivalent isolation levels on the federated server.

Table 20. Data sources and isolation levels

Data sources	Most restrictive isolation level	More restrictive isolation level	Less restrictive isolation level	Least restrictive isolation level
Federated database	Repeatable read	Read stability	Cursor stability	Uncommitted read
DB2 family of products	Repeatable read	Read stability*	Cursor stability	Uncommitted read
Informix	Repeatable read	Repeatable read	Cursor stability	Dirty read
JDBC	Serializable	Repeatable read	Read committed	Read Uncommitted

Table 20. Data sources and isolation levels (continued)

Data sources	Most restrictive isolation level	More restrictive isolation level	Less restrictive isolation level	Least restrictive isolation level
Microsoft SQL Server	Serializable	Repeatable read	Read committed	Read Uncommitted
ODBC	Serializable	Repeatable read	Read committed	Read Uncommitted
Oracle	Serializable	Serializable	Read committed	Read committed
Sybase	Level 3	Level 3	Level 1	Level 0
*For DB2 for VM and VSE Server data sources, the isolation level is repeatable read.				

The CURRENT ISOLATION special register is not used by the federated server when it connects to a data source.

The nonrelational data sources do not have a concept like isolation levels. The OLE DB and Teradata do have the concept of isolation levels but are not supported by the federated server. There is no isolation-level mapping between the federated database isolation levels and the OLE DB, Teradata, and nonrelational data sources.

Statement level isolation in a federated system

For federated data sources, you must use the WITH isolation clause to specify the isolation of a statement.

You must use the WITH isolation clause in your statement if you want to use statement level isolation. If you use attributes with the Call Level Interface (CLI) or other application API for statement level isolation, it does not affect statement isolation.

The data sources that support statement level isolation in a federated system are the DB2 family of products and Microsoft SQL server. The statement isolation is sent to remote data sources for the DB2 family of products and SQL server.

Use the DB2_STATEMENT_ISOLATION server option to turn on or off statement level isolation. You can specify this option in the CREATE SERVER and ALTER SERVER statements. The server option is automatically set to Y.

You can use the WITH isolation clause in these statements:

SELECT
SELECT INTO
Searched DELETE
INSERT
Searched UPDATE
DECLARE CURSOR

Lock request clause

You can use the lock request clause in a SELECT or SELECT INTO statement. The federated data sources that support the lock request clause are the DB2 for Linux, UNIX, and Windows, and the DB2 for z/OS.

Limitations using the WITH clause to set the isolation level

The following conditions apply to isolation levels that are specified for statements:

- The WITH clause cannot be used in subqueries.
- The UR isolation level applies only if the result table of the fullselect or the SELECT INTO statement is read-only. In other cases, the UR isolation level for the statement is changed from UR to CS for the DB2 family of data sources. For the SQL server data source, the UR isolation level is upgraded to Read committed.
- If you specify the lock request clause for the following data sources, the clause is ignored by the federated server.
  - DB2 for System i
  - DB2 for VM
  - Microsoft SQL server

Connection level isolation in a federated system

The federated server maps your isolation level to a corresponding one at the data source.

For each connection to the data source, the wrapper determines the isolation level.

When the federated server connects to the data source, the isolation level at the remote data source is set to a level that is equivalent to the level of the federated server. If there is no exact equivalent, then the federated server sets the isolation level to the next restrictive level. After a connection is made to a data source, the isolation level for the duration of the connection cannot be changed.

All wrappers except Teradata keep track of the connection isolation level. When setting up a connection, the wrappers set the connection isolation level to the equivalent of the current isolation level. The current isolation level is the isolation level of the current section (first federated statement to a data source). The Teradata wrapper is always in the READ isolation level, the default, because the Teradata wrapper does not have a way to change the connection isolation level.

Chapter 22. Federated LOB support

With a federated database system, you can access and manipulate large objects (LOBs) at remote data sources.

A federated system supports SELECT operations on LOBs at DRDA, Informix, Microsoft SQL Server, Oracle, and Sybase data sources. For example:

```
SELECT empname, picture FROM infmx_emp_table
   WHERE empno = '01192345'
```

Where *picture* represents a LOB column and *infmx_emp_table* represents a nickname referencing an Informix table containing employee data.

A federated system supports SELECT, INSERT, UPDATE, and DELETE operations on LOBs at the following data sources, using the DRDA wrapper:

- DB2 for z/OS (Version 7 or higher)
- DB2 for System i (Version 5)
- DB2 Database for Linux, UNIX, and Windows (Version 7 or higher)

The read and write operations supported by DB2 Database for Linux, UNIX, and Windows are listed in the following table:

Table 21. Read and write support for LOBs

Data source	Type of operations
DB2 for z/OS, DB2 for System i, DB2 Database for Linux, UNIX, and Windows[1]	read and write
BioRS	read only
Informix	read only
JDBC	read only
Microsoft SQL Server	read only
Oracle (NET8 wrapper) [2]	read and write
ODBC	read only
Sybase	read only
Teradata	read and write
Web services	read only and bind-out for CLOB only
XML	read only

Note:

1. DB2 for System i (Version 5 or later) is required for LOB support.
2. To run insert, update, and delete operations on Oracle LONG columns, you need to migrate the remote columns from LONG to LOBs and recreate the nicknames.

Teradata LOBs

Teradata LOBs are slightly different than DB2 LOBs. Teradata does not have any data types as large as the LOBs supported in DB2 for Linux, UNIX, and Windows software. However, there are some Teradata data types that can be up to 64000 bytes long. These data types are CHAR, VARCHAR, BYTE, VARBYTE, GRAPHIC, and VARGRAPHIC. These Teradata data types are mapped to DB2 LOB data types when the length of

the Teradata data type exceeds the limits of the corresponding DB2 data type. However, you cannot apply LOB write operations on LOB columns that are mapped from Teradata non-LOB columns.

LOB lengths
Some data sources, such as Oracle and Informix, do not store the lengths of LOB columns in their system catalogs. When you create a nickname on a table, information from the data source system catalog is retrieved including column length. Since no length exists for the LOB columns, the federated database assumes that the length is the maximum length of a LOB column in DB2 Database for Linux, UNIX, and Windows. The federated database stores the maximum length in the federated database catalog as the length of the nickname column.

LOB locators

Applications can request LOB locators for LOBs that are stored in remote data sources. A LOB locator is a 4-byte value stored in a host variable. An application can use the LOB locator to refer to a LOB value (or LOB expression) held in the database system.

Using a LOB locator, an application can manipulate the LOB value as if the LOB value was stored in a regular host variable. When you use LOB locators, there is no need to transport the LOB value from the data source server to the application (and possibly back again).

The federated database can retrieve LOBs from remote data sources, store them at the federated server, and then issue a LOB locator on the stored LOB. LOB locators are released when:

- Applications issue FREE LOCATOR SQL statements
- Applications issue COMMIT statements
- The federated instance is restarted

Restrictions on LOBs

Federated systems impose some restrictions on LOBs.

The following restrictions apply to LOBs:

- The federated database is unable to bind remote LOBs to a file reference variable
- LOBs are not supported in pass-through sessions
- LOBs are not supported as parameters of stored procedures
- LOBs in a Teradata nickname do not support write operations if they are mapped from Teradata non-LOB columns (CHAR, VARCHAR,BYTE, VARBYTE, GRAPHIC, VARGRAPHIC).

Performance considerations for LOB processing

When developing federated applications that fetch and process LOB data, application designers and database administrators need to understand how LOB processing affects performance.

When an application fetches data from a federated data source, the federated server must fetch the data into its own application buffers before sending the data to the application. Because LOBs are not processed in a buffer pool, the LOB data

must first pass through a temporary table space defined for the federated server. To help improve performance and reduce resource consumption, application designers should only materialize LOB data when necessary.

Similarly, when the federated server updates remote LOB data, the data must pass through a temporary table space assigned to the federated server before it is passed to the data source.

Transient LOBs use the temporary table space assigned to the federated server. Therefore, database administrators might need to increase the size of this temporary table space to ensure that the working area is sufficient for processing the LOBs.

Recommendation: To maximize performance when working with LOBs, define the temporary table space as System Managed (SMS) and ensure that the temporary table space is located on disks with a high I/O bandwidth.

Using the DB2 Call Level Interface to access federated LOBs

The federated server supports two DB2 CLI APIs for selecting LOB data:

- The SQLFetch API fetches the LOB from the federated server or data source into the application buffers in a single operation.
- The SQLGetData API fetches the LOB a chunk at a time and can require repeated calls to the API to fetch the entire LOB into the application buffers.

Recommendation: For optimal performance, use the SQLGetData API when fetching LOBs through a federated server.

The federated server supports the SQLExecute and SQLPutData APIs for updating LOB data. The SQLExecute API updates the LOB data in a single operation, whereas the SQLPutData API can require repeated calls to send all of the LOB data from the application buffers to the server. Each API performs at the same level in a federated environment.

Trusted and fenced wrappers

Nicknames created for wrappers defined as trusted or fenced perform equally when fetching or updating LOBs.

Chapter 23. Distributed requests for querying data sources

Queries submitted to the federated database can request results from a single data source, but typically are requests that include multiple data sources. Because a typical query is distributed to multiple data sources, it is called a *distributed request*.

In general, a distributed request uses one or more of three SQL conventions to specify where data is to be retrieved from subqueries, set operators, and join subselects.

Distributed requests for querying data sources - examples

The examples in this topic illustrate distributed requests with a subquery, set operators, and a join operation.

In the following examples, the federated server is configured to access a DB2 for z/OS data source, a DB2 for System i data source, and an Oracle data source. Stored in each data source is a table that contains employee information. The federated server references these tables by nicknames that point to where the tables reside.

zOS_EMPLOYEES
Nickname for a table on a DB2 for z/OS data source that contains employee information.

SYSTEMi_EMPLOYEES
Nickname for a table on a DB2 for System i data source that contains employee information.

ORA_EMPLOYEES
Nickname for a table on an Oracle data source that contains employee information.

ORA_REGIONS
Nickname for a table on an Oracle data source that contains information about the regions that the employees live in.

The following examples illustrate the three SQL conventions used with distributed requests, using the nicknames defined for each of the tables.

Example: A distributed request with a subquery

SYSTEMi_EMPLOYEES contains the phone numbers of employees who live in Asia. It also contains the region codes associated with these phone numbers, but it does not list the regions that the codes represent. ORA_REGIONS lists both codes and regions. The following query uses a subquery to find the region code for China. Then it uses the region code to return a list of those employees in SYSTEMi_EMPLOYEES who have a phone number in China.

```
SELECT name, telephone FROM db2admin.SYSTEMi_employees
   WHERE region_code IN
   (SELECT region_code FROM dbadmin.ora_regions
   WHERE region_name = 'CHINA')
```

Example: A distributed request with set operators

The federated server supports three set operators: UNION, EXCEPT, and INTERSECT.

- Use the UNION set operator to combine the rows that satisfy any of two or more SELECT statements.
- Use the EXCEPT set operator to retrieve those rows that satisfy the first SELECT statement but not the second.
- Use the INTERSECT set operator to retrieve those rows that satisfy both SELECT statements.

All three set operators can use the ALL operand to indicate that duplicate rows are not to be removed from the result. This eliminates the need for an extra sort.

The following query retrieves all employee names and region codes that are present in both SYSTEMi_EMPLOYEES and zOS_EMPLOYEES, even though each table resides in a different data source.

```
SELECT name, region_code
   FROM as400_employees
INTERSECT
SELECT name, region_code
   FROM zOS_employees
```

Example: A distributed request for a join

A relational join produces a result set that contains a combination of columns retrieved from two or more tables. You should specify conditions to limit the size of the rows in the result set.

The query below combines employee names and their corresponding region names by comparing the region codes listed in two tables. Each table resides in a different data source.

```
SELECT t1.name, t2.region_name
   FROM dbadmin.SYSTEMi_employees t1, dbadmin.ora_regions t2
   WHERE t1.region_code = t2.region_code
```

Optimizing distributed requests with server options

In a federated system, use parameters called *server options* to supply the global catalog with information that applies to a data source as a whole or to control how the federated database interacts with a data source.

About this task

About this task

Server options describe the capabilities of a particular data source and enhance the knowledge that the federated server has about that data source. For example, you can use these server options:

- The VARCHAR_NO_TRAILING_BLANKS server option informs the optimizer that every VARCHAR column residing on the data source server is free of trailing blanks. Use this option only when you are certain that all VARCHAR2 columns for every object that is referenced by a nickname on the server has no trailing blanks. Otherwise, use a column option to specify the columns for individual objects on the server that have no trailing blanks. The column option is also named VARCHAR_NO_TRAILING_BLANKS.

- The PLAN_HINTS server option provides Sybase data sources with statement fragments, called *plan hints*. Plan hints help the data source optimizer decide which index to use in accessing a table and which table join sequence to use in retrieving data for a result set.

Typically, the database administrator sets server options for a federated system. However, a programmer can make good use of the server options that help optimize queries. For example, for data sources SYB1 and SYB2, the PLAN_HINTS server option is set to the default, N (no, do not furnish this data source with plan hints). You write a distributed request that selects data from SYB1 and SYB2. You expect that the optimizers at these data sources can use the plan hints to improve their strategies for accessing this data. You can override the default with a setting of Y (yes, furnish the plan hints) while your application is connected to the federated database. When the connection to the data sources terminates, the setting automatically reverts to N.

Procedure

To set server options:

Procedure

Use the SET SERVER OPTION statement to set or change server options. To ensure that the setting takes effect, specify the SET SERVER OPTION statement immediately following the CONNECT statement. The server option is set for the duration of the connection to the federated database.

What to do next

Consider preparing the statement dynamically. The SET SERVER OPTION statement affects only dynamic SQL statements.

For static SQL, using the SET SERVER OPTION statement affects only the execution of the static SQL statement. Using the SET SERVER OPTION statement has no affect on the plans that the optimizer generates.

Canceling a federated query

You can interrupt an application and cancel a running query. With this support, you can cancel a query on the remote data source application before it finishes, and propagate the query interrupt and cancellation to the remote data source.

About this task

Restrictions

Starting in Version 9.7, you can interrupt an application and cancel a running query on the DB2 Database for Linux, UNIX, and Windows data source only.

You can cancel federated SELECT statements, INSERT, UPDATE and DELETE statements, federated stored procedures, and statements in pass-through sessions.

A federated statement can contain multiple segments that access multiple data sources. If a segment accesses a data source that is not supported, for example, multiple query segments against the Sybase data source, you cannot cancel the statement.

You cannot cancel a federated query when ATQ (asynchronous table queue) is enabled.

About this task

When the federated server is running a federated SQL statement and is blocked waiting for results and for a response from the remote data source, the application on the federated server is in 'federated request pending' state. You can issue the LIST APPLICATIONS command to identify applications in ‘federated request pending' state.

When an application is in 'federated request pending' state, you can use Ctrl-C to interrupt and cancel an application or use the FORCE APPLICATION command to stop an application on the remote server.

Procedure

To cancel a federated query, use either of the following methods:

- Press Ctrl-C to interrupt the application

  You can press Ctrl-C to interrupt the application and cancel the query running on the remote server. The current SQL statement on the federated server is interrupted and canceled, and local database consistency is maintained. The remote statement is also canceled. Both the outbound and inbound connections remain active.
- Issue the FORCE APPLICATION command

  You can issue the FORCE APPLICATION command to stop an application. The application is stopped on the federated server and transaction consistency is maintained both locally and on the remote server. The inbound and outbound connections are closed and the remote portion of the application execution on the remote data source is canceled.

Chapter 24. Querying data sources directly with pass-through

Use pass-through sessions to perform operations that are not possible with the DB2 SQL/API.

About this task

About this task

Pass-through sessions are useful when:

- Applications must create objects at the data source or perform INSERT, UPDATE, or DELETE operations.
- The federated database does not support a unique data source operation.

Procedure

To query data sources directly with pass-through:

Procedure

- Use the SET PASSTHRU statement to start a pass-through session and access a server directly. This statement can be issued dynamically. An example of this statement is: SET PASSTHRU ORACLE1 This SET PASSTHRU statement opens a pass-through session to the data source using the server name ORACLE1. ORACLE1 is the name you registered for the data source server when you created the server definition.
- When the pass-through session is opened, ensure that you use the true name of the object and not the nickname when you reference objects in a pass-through session. You must use the SQL dialect of the data source, unless the federated database is the data source that is being referenced.
- If a static statement is submitted in a pass-through session, it is sent to the federated server for processing. If you want to submit an SQL statement to a data source for processing, you must prepare it dynamically in the pass-through session and have it executed while the session is still open. To prepare statements dynamically in a pass-through session:
  - To submit a SELECT statement, use the PREPARE statement with it, and then use the OPEN, FETCH, and CLOSE statements to access the results of your query.
  - For a supported statement other than SELECT, you have two options. You can use the PREPARE statement to prepare the supported statement, and then the EXECUTE statement to execute it. Alternatively, you can use the EXECUTE IMMEDIATE statement to prepare and execute the statement.

What to do next

If you issue the COMMIT or ROLLBACK command during a pass-through session, this command will complete the current unit of work, but does not end the pass-through session.

Federated pass-through considerations and restrictions

This topic explains the considerations and restrictions you need to be aware of when you use a pass-through session.

The following considerations and restrictions apply to all data sources:

- Statements prepared within a pass-through session must be executed within the same pass-through session. Statements prepared within a pass-through session, but executed outside of the same pass-through session, will fail and result in a SQLSTATE 56098 error. Statements prepared outside of a pass-through session, but executed within a pass-through session, are handled as a SET PASSTHRU statement.
- An application can issue multiple SET PASSTHRU statements, however only the last session is active. When a new SET PASSTHRU statement is invoked, it terminates the previous SET PASSTHRU statement. You cannot pass through to more than one data source in the same pass-through session.
- If multiple pass-through sessions are used in an application, be sure to issue a COMMIT before you open another pass-through session. This will conclude the unit of work for the current session.
- Parameter markers are not supported in pass-through sessions. Use host variables instead of parameter markers.
- You can use the WITH HOLD semantics on a cursor defined in a pass-through session. However, you might receive an error if you try to use the WITH HOLD semantics following a COMMIT and the data source does not support the WITH HOLD semantics.
- Host variables defined in SQL statements within a pass-through session must take the form :H*n* where H is uppercase and *n* is a unique whole number. The values of *n* must be numbered consecutively beginning with zero.
- Pass-through does not support LOBs.
- Pass-through does not support stored procedure calls.
- Pass-through does not support the SELECT INTO statement.
- Pass-through does not support SQL or external user-defined functions.
- You cannot execute dynamic SQL COMMIT or ROLLBACK statements during a pass-through session.
- When performing update or delete operations during a pass-through session, you cannot use the WHERE CURRENT OF CURSOR condition.
- In pass-through mode, dynamic SQL statements are executed remotely, whereas static SQL statements are sent to the federated server for processing. If you want to submit an SQL statement to a data source for processing, you must prepare it dynamically in the pass-through session and it must be executed while the session is still open.
- Static SET PASSTHRU statements in SQL-bodied stored procedures are blocked when stored procedures are created, and error SQL0104N is issued. To get into and out of pass-through mode, use the EXECUTE IMMEDIATE statement.

  Example:

```
create procedure stp()
dynamic result sets 1
language sql
modifies sql data
begin
declare stmt varchar(100);

DECLARE cur1 CURSOR WITH RETURN TO CALLER
```

```
FOR SELECT * FROM t1 ;

set stmt = 'set passthru mvs7';
execute immediate stmt;

set stmt = 'insert into t1 values (20, ''passthru_insert'')';
execute immediate stmt;
commit;

insert into t1 values (20, 'stp_insert');
commit;

OPEN cur1;

end
DB20000I The SQL command completed successfully.

call stp()


Result set 1
--------------------------------
10 local
20 stp_insert

2 record(s) selected.

Return Status = 0


select * from t1

C1 C2
--------------------------------
10 remote
20 passthru_insert

2 record(s) selected.


set passthru reset
DB20000I The SQL command completed successfully.

select * from t1

C1 C2
--------------------------------
10 local
20 stp_insert

2 record(s) selected.
```

- The return from a stored procedure or compound SQL statement does not terminate pass-through mode automatically. To exit from pass-through mode, the SET PASSTHRU RESET statement must be called explicitly, as shown in the example above.

Pass-through sessions to Oracle data sources

This topics identifies some SQL considerations to be aware of before you submit SQL statements to Oracle data sources in a pass-through session.

- Any DDL statement issued on an Oracle server is performed at parse time and is not subject to transaction semantics. The operation, when complete, is automatically committed by Oracle. If a rollback occurs, the DDL is not rolled back.

- When you issue a SELECT statement from raw data types, use the RAWTOHEX function to receive the hexadecimal values. When you perform an INSERT into raw data types, provide the hexadecimal representation.

Chapter 25. Tuning query processing

You can tune a federated system to improve the performance of query processing.

About this task

About this task

When you submit SQL queries to the federated database, the SQL compiler processes the query, and the query optimizer analyzes it and creates an access plan. The query optimizer stores the access plan information in the Explain tables in the federated database. You can use the Explain table format, db2expln and dynexpln tools to understand the access plan for a particular SQL statement.

Procedure

To tune federated systems to improve query processing:

Procedure

1. Submit the SQL queries that you want to tune to the federated database.
2. Analyze where the query is being evaluated.
3. Investigate the reasons for the decisions that were made in the access plan and modify your system to increase opportunities for pushdown.

Publications about federated performance

You can refer to many IBM documents that contain detailed information about performance tuning.

- Asynchronous execution of federated queries in WebSphere Federation Server V9.1, at http://www.ibm.com/developerworks/db2/library/techarticle/dm-0611norwood/
- Maximize the performance of WebSphere Information Integrator with Materialized Query Tables, at http://www.ibm.com/developerworks/db2/library/techarticle/dm-0605lin/index.html
- Performance enhancements in IBM WebSphere Federation Server V9.1, Part 1: Improve performance of federated queries with new WebSphere Federation Server capabilities, at http://www.ibm.com/developerworks/db2/library/techarticle/dm-0612englert/
- Performance enhancements in IBM WebSphere Federation Server V9.1, Part 2: Performance characteristics of new functionality in WebSphere Federation Server, at http://www.ibm.com/developerworks/db2/library/techarticle/dm-0612englert2/
- Using data federation technology in IBM WebSphere Information Integrator: Data federation usage examples and performance tuning, at http://www.ibm.com/developerworks/db2/library/techarticle/dm-0507lin/
- Parallelism in WebSphere Information Integrator V8.2, at http://www-128.ibm.com/developerworks/db2/library/techarticle/dm-0502harris/
- Data Federation with IBM DB2 Information Integrator V8.1, at http://publib-b.boulder.ibm.com/Redbooks.nsf/RedbookAbstracts/sg247052.html?Open

- "Using the federated database technology of IBM DB2 Information Integrator", at ftp://ftp.software.ibm.com/software/data/pubs/papers/iifed.pdf

Query analysis

An important part of query processing is the analysis that determines how to tune the query for optimal performance.

To obtain data from data sources, clients (users and applications) submit queries in SQL to the federated database. The SQL compiler then consults information in the global catalog and the data source wrapper to help it process the query. This includes information about connecting to the data source, server attributes, mappings, index information, and nickname statistics.

As part of the SQL compiler process, the *query optimizer* analyzes a query. The compiler develops alternative strategies, called *access plans*, for processing the query. The access plans might call for the query to be:

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

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

- The amount of data that needs to be processed.
- The processing speed of the data source.
- The amount of data that the fragment will return.
- The communication bandwidth.

Pushdown analysis is only performed on relational data sources. Nonrelational data sources use the request-reply-compensate protocol.

The following figure illustrates the steps performed by the SQL compiler when it processes a query.

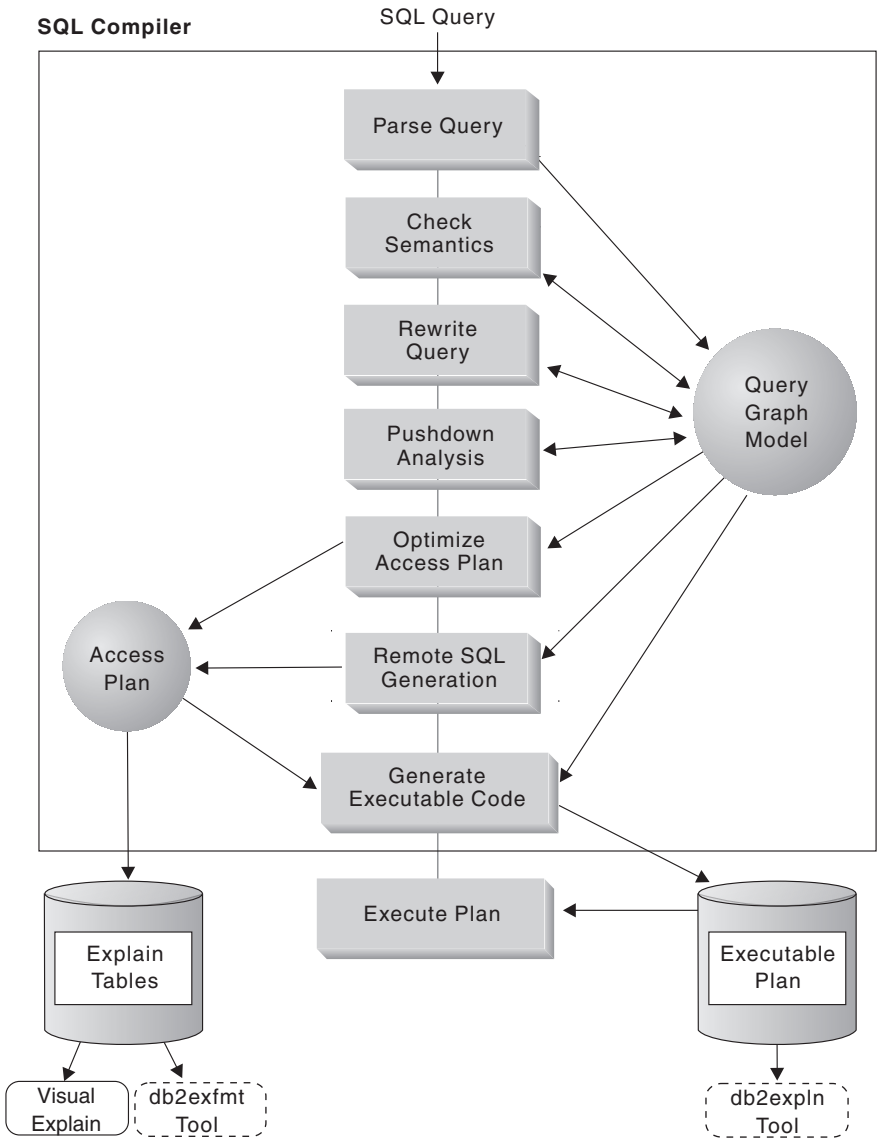


Figure 9. SQL compiler query analysis flowchart

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

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

The primary task of pushdown analysis is to determine which operations can be evaluated remotely. Pushdown analysis does this based on the SQL statement it receives and its knowledge of the capabilities and semantics of the remote data source. Based on this analysis, the query optimizer evaluates the alternatives and chooses the access plan based on cost. The optimizer might choose to not perform an operation directly on a remote data source because it is less cost-effective. A

secondary task is to attempt to rewrite the query to compensate for the difference in semantics and SQL operations between the federated server and the data source so that the query is better optimized.

The final access plan selected by the optimizer can include operations evaluated at the remote data sources. For those operations that are performed remotely, the SQL compiler creates efficient SQL phrased in the SQL dialect of the remote data source during the generation phase. The process of producing an optimal query plan that takes all sources into account is called *global optimization*.

For nonrelational sources, the wrappers use the request-reply-compensate protocol.

Pushdown analysis

Pushdown analysis tells the query optimizer if a remote data source can perform an operation. An operation can be a function, such as relational operator, system or user functions, or an SQL operator (GROUP BY, ORDER BY, and so on). The optimizer then makes a cost-based decision about whether or not to push down the operator. Even if pushdown analysis determines that a particular operation can be performed at the remote source, the optimizer might decide to execute it locally at the federated server, if doing so appears to consume fewer resources.

Pushdown analysis is performed on relational data sources. Nonrelational sources use the request-reply-compensate protocol.

Functions that cannot be pushed-down, can significantly impact query performance. Consider the effect of forcing a selective predicate to be evaluated locally instead of at the remote data source. This approach could require the federated server to retrieve the entire table from the remote data source, and then filter the table locally using the predicate. If your network is constrained—and the table is large—query performance could suffer.

Operators that are not pushed-down can also significantly impact query performance. For example, having a GROUP BY operator aggregate remote data locally could, once again, require the federated server to retrieve the entire table from the remote data source.

For example, suppose that the nickname EMP references the table EMPLOYEE. This table has 10,000 rows. One column contains the zip codes, and one column contains the salary for each employee. The following query is sent to the federated server to count the number of employees per city who earn greater than 50,000 that live in a particular ZIP code range:

```
SELECT CITY, COUNT(*) FROM EMP
 WHERE ZIP BETWEEN 'CA1' AND 'CA5' AND SALARY > 50000
 GROUP BY CITY;
```

When the SQL compiler receives this statement, it considers several possibilities:

- The collating sequences of the data source and the federated server are the same. It is likely that both predicates will be pushed down, because they are likely to reduce the size of the intermediate result set sent from the data source to the federated server. It is usually more efficient to filter and group results at the data source instead of copying the entire table to the federated server and performing the operations locally. Pushdown analysis determines if operations can be performed at the data source. Since the collating sequences are the same, the predicates and the GROUP BY operation can take place at the data source.

- The collating sequences are the same, and the query optimizer knows that the federated server is very fast. It is possible that the query optimizer will decide that performing the GROUP BY operation locally is the best (least cost) approach. The predicates will be pushed-down to the data source for evaluation. This is an example of pushdown analysis combined with global optimization.
- The collating sequences are not the same. Most likely, the SALARY predicate will be pushed down to the data source, because numeric columns are sorted the same, regardless of collating sequence. However, the predicate on ZIP will not be pushed down because it is order-dependant on a character column. The GROUP BY will not be pushed down unless the predicates on both ZIP and SALARY are also pushed down.

The SQL compiler will consider the available access plans, and then choose the plan that is the most efficient.

In general, the goal is to ensure that the query optimizer considers pushing down the functions and operators to the data sources for evaluation. Many factors can affect whether a function or an SQL operator is evaluated at a remote data source. The key factors which influence the query optimizer are: server characteristics, nickname characteristics, and query characteristics.

Server characteristics affecting pushdown opportunities

Server characteristics that affect pushdown include SQL support, collating sequence, federated server options, and type and function mappings.

The factors that affect pushdown opportunities for nonrelational data sources are different than the factors that affect pushdown opportunities for relational data sources. The SQL dialect is not a factor for most nonrelational data sources, since they do not use SQL.

The following topics describe the data source-specific factors that can affect pushdown opportunities.

SQL differences

SQL characteristics that affect pushdown include SQL capabilities, restrictions, limitations, and SQL specific to a server.

- SQL capabilities. Each data source supports a variation of the SQL dialect and different levels of functionality. For example, consider the GROUP BY list. Most data sources support the GROUP BY operator. However some data sources have restrictions on the number of items on the GROUP BY list, or restrictions on whether an expression is allowed on the GROUP BY list. If there is a restriction at the remote data source, the federated server might perform the GROUP BY operation locally.
- SQL restrictions. Each data source can have different SQL restrictions. For example, some data sources require parameter markers to bind in values to remote SQL statements. Therefore, parameter marker restrictions must be checked to ensure that each data source can support such a bind mechanism. If the federated server cannot determine a good method to bind in a value for a function, this function must be evaluated locally.
- SQL limitations. The federated server might allow the use of larger integers than the remote data sources. However, limit-exceeding values cannot be embedded in statements that are sent to the data sources. Therefore, the function or operator that operates on this constant must be evaluated locally.

- Server specifics. Several factors fall into this category. One example is sorting NULL values (highest, or lowest, depending on the ordering). For example, if the NULL value is sorted at a data source differently from the federated server, ORDER BY operations on a nullable expression cannot be remotely evaluated.

VARCHAR2 data type in federated systems

To tune your query processing, your access plan must account for blank padding with VARCHAR2 data.

You use the VARCHAR2_COMPAT server option to enable support for VARCHAR2 compatible data sources.

The VARCHAR2 compatibility semantics determine how your federated server and data source handle the VARCHAR2 data type depending on whether the federated server, the data source, or both are VARCHAR2 compatible. The systems have these possible configurations:

- Your federated server is VARCHAR2 compatible but connects to a data source that is not VARCHAR2 compatible.
- Your federated server is not VARCHAR2 compatible but connects to a data source that is VARCHAR2 compatible.
- Your federated server and data source are both VARCHAR2 compatible.

When your federated server or DB2 data source is VARCHAR2 compatible, the VARCHAR2 data type is handled as a synonym for the VARCHAR data type.

Tip: To ensure improved performance, in systems that only connect to VARCHAR2-compatible data sources, your federated server should also be VARCHAR2 compatible.

In the following scenarios, the federated server handles your empty string values and pushes down the operations to the data source based on the VARCHAR2 compatibility semantics.

Scenario 1

The federated server is not VARCHAR2 compatible but connects to a data source that is VARCHAR2 compatible. The federated server allows empty strings, but the data source does not. Thus, all empty strings that are pushed down to the data source are converted to NULL values.

The default behavior of the federated server is to use blank-padding semantics on the CHAR data type before pushing the result down to the data source. But, the federated server does not use blank padding in operations that only specify empty strings.

Examples

- The following INSERT and UPDATE statements only include empty strings are not blank padded. The statements push down empty string values that are converted to NULL values by the data source.

  For example, the following statements are not blank padded by the federated server:

  ```
  INSERT INTO n1 (c1) VALUES ('')
  UPDATE n1 SET c1=''
  INSERT INTO n1 (c1) VALUES (''), ('')
  ```

- The following INSERT statement sends 10 blanks to the data source instead of the empty string:

  ```
  INSERT INTO n1 (col_char) VALUES (''), ('ibm')
  ```

Scenario 2

The federated server is VARCHAR2 compatible but connects to a data source that is not VARCHAR2 compatible. The VARCHAR2 compatibility semantics of the federated server does not allow empty strings nor does it use blank-padding semantics. But, the semantics of the data source allows empty strings and uses blank-padding semantics. Thus, the federated server restricts operations that involve empty strings from being pushed down, unless the federated server semantics and data consistency are preserved.

Examples

- The following INSERT statement is pushed down to the data source, because the empty string is converted to a NULL value and cannot be handled locally:

  ```
  INSERT INTO n1 (c1) VALUES ('')
  ```

- The following INSERT statement is pushed down to the data source if you split the statement into two separate statements:

  ```
  INSERT INTO n1 SELECT * FROM n2
  ```

  You are required to use two statements when the target or source tables are from a data source that is not VARCHAR2 compatible, for example:

  ```
  SELECT * FROM n2
  INSERT INTO n1 VALUES (:H0)
  ```

- Functions that are nested within expressions are run locally in the federated server, for example:

  ```
  SELECT LENGTH(TRIM(c1)) FROM n1
  ```

- All comparison operations are run locally in the federated server because inconsistent results are possible when comparison operations are run by the data source.

Scenario 3

The federated server and data source are both VARCHAR2 compatible. Neither the federated server nor the data source allows empty strings or use blank-padding semantics. Thus, all operations that involve empty strings are pushed down to the data source, because the semantics are preserved.

Collating sequence

If you set the COLLATING_SEQUENCE server option to 'Y', you are telling the federated database that the data source collating sequence matches the collating sequence of the federated server. This setting allows the optimizer to consider pushing down order-dependent processing to a data source, which can improve performance.

If the data source collating sequence is not the same as the federated database collating sequence and you set the COLLATING_SEQUENCE server option to 'Y', you can receive incorrect results. For example, if your plan uses merge joins, the optimizer might push down ordering operations to the data sources. If the data source collating sequence is not the same, the join results might not have a correct

result set. Set the COLLATING_SEQUENCE server option to 'N', if you are not sure that the collating sequence at the data source is identical to the federated database collating sequence.

Alternatively, you can configure a federated database to use the same collating sequence that a data source uses. You then set the COLLATING_SEQUENCE server option to 'Y'. This allows the optimizer to consider pushing down order-dependent operations on character columns.

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

- National language support

  The collating sequence is related to the language supported on a server. Compare the federated database NLS information for your operating system to the data source NLS information.
- Language-aware collations

  Check whether the federated database or the data source uses language-aware collations. If they use different collations, then you should not set COLLATING_SEQENCE to Y.
- Data source characteristics

  Some data sources are created using case-insensitive collating sequences, which can yield different results from the federated database in order-dependent operations.
- Customization

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

When a query fragment from a federated server requires sorting, the place where the sorting is processed depends on several factors. If the federated database's collating sequence is the same as the data source collating sequence, the sort can take place at the data source or at the federated server. The query optimizer can determine if a local sort or a remote sort is the most efficient way to complete the query.

Numeric comparisons, in general, can be performed at either location even if the collating sequence is different. You can get incorrect results, however, if the weighting of null characters is different between the federated database and the data source.

Likewise, for comparison statements, be careful if you are submitting statements to a case-insensitive data source. The weights assigned to the characters "I" and "i" in a case-insensitive data source are the same. For example, in a case-insensitive data source with an English code page, **STEWART**, **SteWArT**, and **stewart** would all be considered equal. The federated database, by default, is case-sensitive and would assign different weights to the characters.

If the collating sequences of the federated database and the data source differ, the federated server retrieves the data to the federated database, so that it can do the sorting locally. The reason is that users expect to see the query results ordered according to the collating sequence defined for the federated server; by ordering the data locally, the federated server ensures that this expectation is fulfilled.

If your query contains an equality predicate on the character column, it is possible to push down that portion of the query even if the collating sequences are different

(set to 'N'). For example, the predicate C1 = 'A' would retrieve the same values regardless of collating sequence and therefore could be pushed down to a data source that has a different collating sequence than the federated server. However, such predicates cannot be pushed down when the collating sequence at the data source is case-insensitive (COLLATING_SEQUENCE='I'). When a data source is case-insensitive, the results from C1= 'A' and C1 = 'a' are the same, which is not consistent with a case-sensitive environment such as DB2 Database for Linux, UNIX, and Windows.

Administrators can create federated databases with a particular collating sequence that matches the data source collating sequence. This approach can speed performance if all data sources use the same collating sequence or if most or all column functions are directed to data sources that use the same collating sequence. For example, in DB2 for z/OS, sorts defined by ORDER BY clauses are implemented by a collating sequence based on an EBCDIC code page. If you want to use the federated server to retrieve DB2 for z/OS data sorted in accordance with ORDER BY clauses, it is advisable to configure the federated database so that is uses a predefined collating sequence based on the EBCDIC code page.

If the collating sequences at the federated database and the data source differ, and you need to see the data ordered in the data source's sequence, you can submit your query in a pass-through session, or define the query in a data source view.

Federated server options

The server options that you set refine the knowledge that the federated server has about the remote data source.

The previously listed factors that affect pushdown opportunities are characteristics of the database servers, and you can not change them. By carefully considering the following server options, it is possible to improve query performance:

- COLLATING_SEQUENCE. If a data source has a collating sequence that differs from the federated database collating sequence, any order-dependent operations on character values cannot be remotely evaluated at the data source. An example is executing MAX column functions on a nickname character column at a data source with a different collating sequence. Because results might differ if the MAX function is evaluated at the remote data source, the federated database will perform the aggregate operation and the MAX function locally.
- VARCHAR_NO_TRAILING_BLANKS. This option is for varying-length character strings that contain no trailing blanks. Some data sources, such as Oracle, do not apply blank-padded comparison semantics like the federated database does. This padding difference can cause unexpected results.

  For example:

  ```
  'HELLO' = 'HELLO         ' in DB2
  'HELLO' <> 'HELLO         ' in Oracle!
  ```

  If trailing blanks are present on VARCHAR columns on an Oracle data source, you should set this option to N (the default for Oracle). This option impacts performance, because the federated server must compensate for the difference in semantics, but guarantees a consistent result set. Setting this value to Y when an Oracle data source column contains trailing blanks can cause inconsistent results.

  If you are certain that all VARCHAR and VARCHAR2 columns at a data source contain no trailing blanks, consider setting this server option for a data source. Ensure that you consider all objects that can potentially have nicknames, including views.

Recommendation: Set this option on a column by column basis using the VARCHAR_NO_TRAILING_BLANKS column option.

- DB2_MAXIMAL_PUSHDOWN. This option specifies the primary criteria that the query optimizer uses when choosing an access plan. The query optimizer can choose access plans based on cost or based on the user requirement that as much query processing as possible be performed by the remote data sources. With DB2_MAXIMAL_PUSHDOWN set to Y, reducing network traffic becomes the overriding criteria for the query optimizer. The query optimizer uses the access plan that performs the fewest number of "sends" to the data sources. Setting this server option to Y forces the federated server to use an access plan that might not be the lowest cost plan. Using an access plan other than the lowest cost plan can decrease performance. When the DB2_MAXIMAL_PUSHDOWN server option is set to Y a query that results in a Cartesian product is not pushed down the remote data sources. Queries that will result in a Cartesian product will be processed by the federated database. The DB2_MAXIMAL_PUSHDOWN server option does not need to be set to Y for the federated server to pushdown query processing to the remote data sources. When this server option is set to N (the default), the query optimizer will pushdown query processing to the data sources. However, the primary criteria the optimizer uses when the option is set to N is cost instead of network traffic.

"Server characteristics affecting global optimization" on page 253 describes the COMM_RATE, CPU_RATIO, and IO_RATIO server options that also can affect query performance.

Creating statement-level optimization guidelines

With support for statement-level server options, you can set server options to affect a single SQL statement by using optimization profiles.

About this task

An optimization profile is an XML document in which you can specify optimization parameters for one or more SQL statements. You define the contents of the optimization profile in an optimization profile schema. For detailed information about optimization profiles and guidelines, see Optimization profiles and guidelines.

You can specify any federation server option in an optimization profile. The guidelines defined in the optimization profile influence the DB2 optimizer and can improve the performance and efficiency of federation query processing.

You can set statement-level server options in an optimization profile for a single SQL statmenet.

Restriction: Statement-level server options are limited to the DB2_MAXIMAL_PUSHDOWN and the COLLATING_SEQUENCE server options.

Procedure

Create an entry in the SYSTOOLS.OPT_PROFILE table to define an optimization profile.
The SYSTOOLS.OPT_PROFILE table contains all optimization profiles.

Example

Create an entry in SYSTOOLS.OPT_PROFILE.

```
drop table SYSTOOLS.OPT_PROFILE;
call sysinstallobjects('opt_profiles', 'c', '', '');
```

Edit the optimization profile and create a statement profile. Specify the server options after the statement key in the optimization guidelines as shown in bold:

```
<xml version="1.0" encoding="UTF-8"?>
<OPTPROFILE VERSION="10.0.1">
<STMTPROFILE ID="QRY1">
<STMTKEY>
<![CDATA[select * from NICK1 as n1, NICK2 as n2
  where n1.c2 = n2.c1 and n1.c2 > 'a']]>
</STMTKEY>
<OPTGUIDELINES>
<SERVEROPTIONS>
 <SERVER NAME="DATASTORE2">
  <OPTION NAME="DB2_MAXIMAL_PUSHDOWN" VALUE="Y">
  <OPTION NAME="COLLATING_SEQUENCE" VALUE="Y"><
  </SERVER>
  <SERVER NAME="DATASTORE1">
  <OPTION NAME="DB2_MAXIMAL_PUSHDOWN" VALUE="N"/>
 </SERVER>
</SERVEROPTIONS>
</OPTGUIDELINES>
</STMTPROFILE>
</OPTPROFILE>
```

An optimization profile can also contain global guidelines that apply to all SQL statements that are run while the profile is in effect.

Type and function mapping factors

Both the default data type mappings and default function mappings are built into the data source wrappers. Data type mappings describe the relationship between the data source data type and the federated server data type. You can customize default data type mappings. Function mappings describe the relationship between a data source function and a semantically equivalent function at the federated server. In certain cases, the federated database will compensate for functions mappings that are not supported by a data source.

The default data type mappings are designed so that sufficient buffer space is given to each data source data type to avoid runtime buffer overflow and truncation. You can customize the type mapping for a specific data source or for a particular nickname to suit specific applications and in some cases improve performance. For example, Oracle DATE types can contain both a date and a timestamp portion and are therefore mapped to DB2 TIMESTAMPs by default. If you are accessing an Oracle date column, and you know that it contains only date portions (no timestamps), you can use the ALTER NICKNAME statement to change the local data type of the nickname from TIMESTAMP to DATE when the date_compat server option is off. When evaluating predicates based purely on a date, such as SalesDate=DATE('2009-01-04'), this change bypasses the use of a SCALAR function that is used to extract the date from the date and timestamp information held in the column, which can improve performance.

The federated database compensates for functions that are not supported by a data source. Functional compensation usually involves retrieving the necessary data

from the data source and applying the function locally, which often has a performance impact. There are several cases where function compensation occurs:

- A function simply does not exist at the data source. Some of the SYSFUN functions, for example, do not exist on DB2 for z/OS data sources, and thus require local compensation.
- A function exists at the data source; however, the characteristics of the operand violate function restrictions. An example is the IS NULL relational operator. Most data sources support it, but some have restrictions such as only allowing a column name on the left hand side of the IS NULL operator.
- A function, if evaluated remotely, can return a different result. An example is the '>' (greater than) operator. For those data sources with different collating sequences, the greater than operator can return different results than if it is evaluated locally by the federated database.

Nickname characteristics affecting pushdown opportunities

Nickname characteristics that affect pushdown include the local data type of a nickname column, federated column options, and materialized query tables.

There are several nickname-specific factors that can affect pushdown opportunities. The local data type of a nickname column can affect the number of possibilities in a joining sequence evaluated by the optimizer. Nicknames can be flagged with a column option to indicate the columns contain no trailing blanks. This gives the SQL compiler the opportunity to generate a more efficient form of a predicate for the SQL statement sent to the data sources.

Local data type of a nickname column

Ensure that the local data type of a column does not prevent a predicate from being evaluated at the data source.

The default data type mappings are provided to avoid any possible overflow. However, a joining predicate between two columns of different data types or lengths will prevent the hash join technique from being considered by the optimizer. For the optimizer to consider the hash join, both the data types and lengths of the joining columns must match exactly. For example, Oracle data source columns that are designed to hold just integer values are often created as NUMBER within the Oracle database, which defaults to NUMBER (38). A nickname column for this Oracle data type is given the local data type FLOAT because the range of a DB2 integer is only roughly equal to NUMBER (9). In this case, joins between a DB2 integer column and an Oracle column that is defined as NUMBER (but only holding integer values) cannot use the hash join technique because the Oracle column is mapped as a FLOAT type. However, if the domain of this Oracle NUMBER column can be accommodated by the DB2 INTEGER data type, you can change its local data type with the ALTER NICKNAME statement. Then the optimizer can consider the hash join technique, which might improve performance.

Federated column options

You can define federated column options that the query optimizer uses for developing access plans.

The column options tell the wrapper to handle the data in a column differently than it normally would handle it. The SQL complier and query optimizer use the metadata to develop better plans for accessing the data. The federated database

treats the object that a nickname references as if it is a table. As a result, you can set column options for any data source object that you create a nickname for.

The ALTER NICKNAME statement can be used to add or change column options for nicknames. There are two column options:

- NUMERIC_STRING. This column option applies to character type columns (CHAR and VARCHAR). Suppose that a data source has a collating sequence that differs from the federated database collating sequence. The federated server would not sort any columns that contain character data at the data source. It would return the data to the federated database and perform the sort locally. However, suppose that the column is a character data type and contains only numeric characters ('0','1',...,'9'). You can indicate this by assigning a value of 'Y' to the NUMERIC_STRING column option. This gives the query optimizer the option of performing the sort at the data source because numerics, even when represented as character strings, always sort the same, regardless of collating sequence. If the sort is performed remotely, you can avoid the overhead of porting the data to the federated server and performing the sort locally.
- VARCHAR_NO_TRAILING_BLANKS. Unlike the server option with the same name, this column option can be used to identify specific Oracle columns that contain no trailing blanks. The SQL compiler pushdown analysis step will then take this information into account when checking all operations performed on columns which have this setting. Based on the VARCHAR_NO_TRAILING_BLANKS setting, the SQL compiler can generate a different but semantically equivalent form of a predicate that is used in the remote SQL statement sent to the data source. A value of 'Y' is likely to enable the use of remote indexes (if available) which can improve query performance.

Query characteristics affecting pushdown opportunities

An SQL operator that references multiple data sources affects pushdown.

A query can reference a SQL operator that involves nicknames from multiple data sources. When the federated server combines the results from two referenced data sources by using one operator, the operation must take place at the federated server. An example of this is a set operator, like UNION. The operator cannot be evaluated at a remote data source directly.

Analyzing where a query is evaluated

Detailed query optimizer information is kept in Explain tables separate from the actual access plan itself. This information allows for in-depth analysis of an access plan. By examining the SHIP operator of a federated access plan, you can determine what SQL operations were pushed down to a data source and which operations were executed at the federated server.

Explain tables are accessible on all supported operating systems, and contain information for both static and dynamic SQL statements. The following tools are typically used to obtain access plan information from the explain tables:

- Explain table format tool. Use the db2exfmt tool to present the information from the explain tables in a predefined format.
- db2expln and dynexpln tools. You can use these tools to understand the access plan chosen for a particular SQL statement. Both dynamic and static SQL statements can be explained using the Explain Facility. One difference from the Explain tools is that with Visual Explain the Explain information is presented in

a graphical format. Otherwise the level of detail provided in the two methods is equivalent. To fully use the output of db2expln , and dynexpln you must understand:

- The different SQL statements supported and the terminology related to those statements (such as predicates in a SELECT statement)
- The purpose of a package (access plan)
- The purpose and contents of the system catalog tables
- General application tuning concepts

You can also access explain tables using SQL statements. This allows for easy manipulation of the output, for comparison among different queries, or for comparisons of the same query over time.

Analyzing where a query is evaluated with the DB2_MAXIMAL_PUSHDOWN server option

You can use the DB2_MAXIMAL_PUSHDOWN server option in conjunction with the Explain utilities to determine whether a particular operator was not pushed down to execute at a data source because of a cost-based optimizer decision or because pushdown analysis determined it was not possible.

About this task

Procedure

To run the Explain tools on a query with the DB2_MAXIMAL_PUSHDOWN server option:

Procedure

1. Set the DB2_MAXIMAL_PUSHDOWN server option to 'N'. This is the default setting for this option. Pushdown analysis determines which parts of the SQL can be pushed down. Then the query optimizer generates all the alternative plans that do not violate the criteria set by pushdown analysis. The query optimizer estimates the cost of each plan, and will select the plan with the lowest estimated cost. You can analyze the operators that were pushed down to the data source by viewing the details of the appropriate SHIP operator. If an operator you expect to be pushed down was not pushed down, proceed to step 2.
2. Set the DB2_MAXIMAL_PUSHDOWN server option to 'Y'. Use the Explain tools to analyze the SQL statement again. The plan displayed in the Explain output shows all of the SQL operations that can be pushed down to the data source.
   - If the operator is pushed down after resetting the option to 'Y', the optimizer determined that it was more cost-efficient to execute the operator locally, rather than remotely. If the operator is not pushed down after resetting the option to 'Y', it is likely that pushdown analysis did not allow the operator to be executed remotely.
   - If the optimizer made a cost-based decision not to push down the operator, consider checking the nickname statistics to ensure that they are accurate. If pushdown analysis made the decision not to push down the operator, consider checking server options, data type mappings, and function mappings.

Understanding access plan evaluation decisions

The topics in this section list typical access plan analysis questions, and areas that you can investigate to increase pushdown opportunities.

Why isn't this predicate being evaluated remotely?

This question arises when a predicate is very selective and thus could be used to filter rows and reduce network traffic. Remote predicate evaluation also affects whether a join between two tables of the same data source can be evaluated remotely.

Areas to examine include:

- Server options. How do the settings for the server options COLLATING_SEQUENCE and VARCHAR_NO_TRAILING_BLANKS affect where the predicate is evaluated?
- Subquery predicates. Does this predicate contain a subquery that pertains to another data source? Does this predicate contain a subquery involving an SQL operator that is not supported by this data source? Not all data sources support set operators in a predicate.
- Predicate functions. Does this predicate contain a function that cannot be evaluated by this remote data source? Relational operators are classified as functions.
- Predicate bind requirements. Does this predicate, if remotely evaluated, require bind-in of some value? If so, would it violate SQL restrictions at this data source?
- Global optimization. The optimizer decided that local processing is more cost-effective.

Why isn't the GROUP BY operator evaluated remotely?

There are several areas that you can check to determine why a GROUP BY operator is not evaluated remotely.

The areas that you can check include:

- Is the input to the GROUP BY operator evaluated remotely? If the answer is no, examine the input.
- Does the data source have any restrictions on this operator? Examples include:
  - Limited number of GROUP BY items
  - Limited byte counts of combined GROUP BY items
  - Column specification only on the GROUP BY list
- Does the data source support this SQL operator?
- Global optimization. The optimizer decided that local processing is more cost-effective.

Why isn't the SET operator evaluated remotely?

You can check the operands and check data source restrictions to determine why the SET operator is not evaluated remotely.

Considerations:

- Are both of its operands completely evaluated at the same remote data source? If the answer is no and it should be yes, examine each operand.

- Does the data source have any restrictions on this SET operator? For example, are large objects or long fields valid input for this specific SET operator?

Why isn't the ORDER BY operation evaluated remotely?

You can check the input to the operation, what the clause contains, and check data source restrictions to determine why the ORDER BY operator is not evaluated remotely.

Considerations:

- Server options. How do the settings for the server options COLLATING_SEQUENCE and VARCHAR_NO_TRAILING_BLANKS affect where the predicate is evaluated?
- Is the input to the ORDER BY operation evaluated remotely? If the answer is no, examine the input.
- Does the ORDER BY clause contain a character expression? If yes, does the remote data source have a different collating sequence than the federated server collating sequence?
- Does the data source have any restrictions on this operator? For example, is there a limited number of ORDER BY items? Does the data source restrict column specification to the ORDER BY list?

Why is a remote INSERT with a fullselect statement not completely evaluated remotely?

You can check several elements of the subselect to determine why a remote INSERT with a fullselect statement is not completely evaluated remotely.

Considerations:

- Could the subselect be completely evaluated on the remote data source? If no, examine the subselect.
- Does the subselect contain a set operator? If yes, does this data source support set operators as input to an INSERT?
- Does the subselect reference the target table? If yes, does this data source allow this syntax?

Why is a remote INSERT with VALUES clause statement not completely evaluated remotely?

You can check the VALUES clause and the expression to determine why a remote INSERT with a VALUES clause statement is not completely evaluated remotely.

Considerations:

- Can the VALUES clause be completely evaluated at the remote data source? In other words, does an expression contain a function not supported by the remote data source?
- Does the expression involve a scalar subquery? Is that syntax supported?
- Does the expression reference the target table? Is that syntax supported?

Why is a remote, searched UPDATE statement not completely evaluated remotely?

You can check elements of the SET clause and search condition to determine why a remote, searched UPDATE statement is not completely evaluated remotely.

Considerations:

- Can the SET clause be completely evaluated at the remote data source? In other words, does an update expression contain a function not supported by the remote data source?
- Does the SET clause involve a scalar subquery? Does the data source allow this syntax?
- Can the search condition be completely evaluated at the remote data source? If the answer is no, examine the search condition instead.
- Does the search condition or SET clause reference the target table? Does the data source allow this syntax?
- Does the search condition or SET clause reference the target table with correlation? Does the data source allow this syntax?

Why is a positioned UPDATE statement not completely evaluated remotely?

This happens when the federated database chooses to evaluate the update expression locally before sending the UPDATE statement to the data source. This approach should not significantly affect performance.

Considerations:

- Can the SET clause be completely evaluated at the remote data source? In other words, does an update expression contain a function not supported by the remote data source?
- Does the SET clause involve a scalar subquery? Does the data source allow this syntax?

Why is a remote, searched DELETE statement not completely evaluated remotely?

You can check elements of the search condition to determine why a remote, searched DELETE statement is not completely evaluated remotely.

Considerations:

- Can the search condition be completely evaluated at the remote data source? If the answer is no, examine the search condition instead.
- Does the search condition reference the target table? Does the data source allow this syntax?
- Does the search condition reference the target table with correlation? Does the data source allow this syntax?

Data source upgrades and customization

When data sources are upgraded or customized, you need to update global catalog information.

The SQL compiler relies on information that is stored in the global catalog to provide it with the SQL capabilities of the data sources. This information periodically needs to be updated. The SQL capabilities of the data sources might change in new versions of the data sources. When data sources are upgraded or customized, update the global catalog information so that the SQL compiler is using the most current information.

Use SQL DDL statements, such as CREATE FUNCTION MAPPING and ALTER SERVER, to update the catalog.

Pushdown of predicates with function templates

In a federated system, each remote data source has its own functions. Most of these functions have semantically equivalent DB2 functions and have associated function mappings by default. However, some remote source functions might not have equivalent functions on the federated server. Consequently, only the remote data source can execute these functions. To write queries that use these functions, you must create a function template on the federated server.

A function template acts as a local description of the remote function. You create a function template with the CREATE FUNCTION statement by using the AS TEMPLATE clause. There is no executable code associated with the function template at the federated server. When the template is defined, you can use it to create a function mapping, which maps the function template to its remote counterpart. Then it is possible to refer to the function template in the SQL statements that are issued to the federated server and for the function to be evaluated at the data source.

The query optimizer makes cost-based decisions to determine where a predicate can be evaluated. When possible, the optimizer generates a plan to evaluate a predicate with a function template at the corresponding remote server. In some cases, it might not be possible for the optimizer to generate a plan that evaluates the function template at the data source. When this occurs you might receive an SQL0142N error with the following error message:

```
The SQL statement is not supported.
```

To avoid this error, the query can be rewritten to enable pushdown while maintaining the semantics of the original query.

For a function template to be pushed down, it must be defined with the DETERMINISTIC and NO EXTERNAL ACTION clauses.

Chapter 26. Parallelism with queries that reference nicknames

Queries that contain nicknames can participate in three types of intra-query parallelism.

The three types of intra-query parallelism are:

- Intrapartition query parallelism on single partition, multiprocessor configurations
- Interpartition query parallelism on multiple partition configurations
- Mixed query parallelism that consists of both intrapartition and interpartition parallelism where each partition runs on an SMP computer

Intrapartition parallelism with queries that reference nicknames

Intrapartition parallelism refers to the process of dividing a query into multiple concurrent parts that run in parallel by multiple processes on a single database partition.

In federated queries, the part of a query that involves local data can run in parallel while the part that involves nicknames runs serially, using a single agent process.

When multiple processors can work on the local portions of the query, the performance of queries that reference local tables and nicknames can improve.

The DFT_DEGREE database configuration parameter and the CURRENT DEGREE special register control the degree of intrapartition parallelism.

Enabling intrapartition parallelism with queries that reference nicknames

For queries that reference local tables and nicknames in a multiprocessor environment, you can enable intrapartition parallelism. The federated server can then process the local tables in parallel.

About this task

Restrictions

The federated system can process only the portion of a query that references local tables in parallel. The coordinator partition processes all operations on the remote portion of a query in serial.

Procedure

To enable intrapartition parallelism:

Procedure

1. Set the INTRA_PARALLEL database configuration parameter to YES.
2. Set the MAX_QUERYDEGREE database configuration parameter to a value greater than 1.

3. Set the DFT_DEGREE database configuration parameter to a value greater than 1, or set the special register CURRENT DEGREE. If you set the DFT_DEGREE parameter to ANY, the default level of intrapartition parallelism equals the number of processors on the computer.

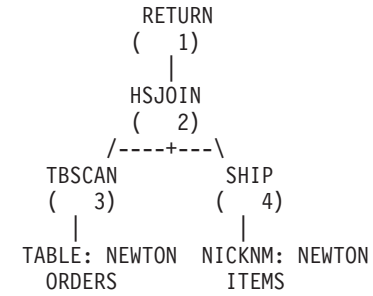
Intrapartition parallelism with queries that reference nicknames - examples of access plans

You can use the DB2 Explain facility to view the access plan that the optimizer uses during query processing. The following examples show how the optimizer accesses nickname data in an intrapartition parallelism environment.

Example 1: Without parallelism support

In this example, the federated server processes the join of the local table, ORDERS, and nickname, ITEMS, serially. No intrapartition parallelism is used.

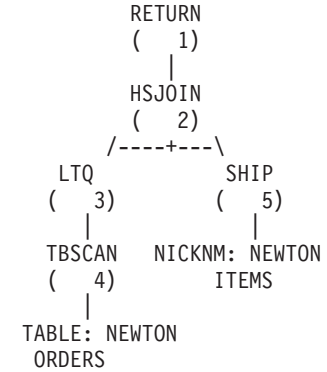
```
SELECT *
FROM ORDERS A, ITEMS B
WHERE A.ID1 = B.ID1 AND B.ITEM = 3
```



Example 2: With parallelism support

In this example of a join, the query can run faster by having the local table read in parallel before the serial join with the nickname. The LTQ operator indicates where parallelism is introduced into the plan.

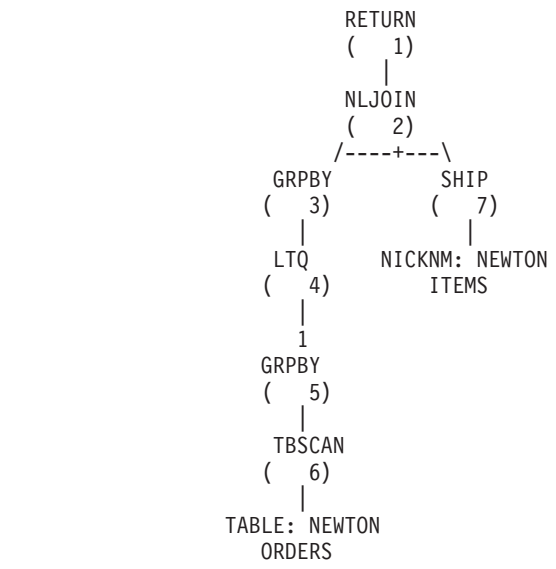
```
SELECT *
FROM ORDERS A, ITEMS B
WHERE A.ID1 = B.ID1 AND B.ITEM = 3
```



Example 3: Intrapartition parallelism with aggregation

In this example, the database aggregates the local table data in parallel in the partition, improving the execution of the aggregation. The join of the local table and the nickname occurs serially.

```
SELECT *
FROM ITEMS A
WHERE ID =
   (SELECT MAX(ID)
   FROM ORDERS
   WHERE NUMBER = 10)
```



Interpartition parallelism with queries that reference nicknames

Interpartition parallelism refers to the process of dividing a single query into multiple parts that run in parallel on different partitions of a partitioned database.

In queries that reference local and remote data, the federated server can distribute the remote data to each of the local partitions. Figure 10 on page 234 and Figure 11 on page 234 show the concept of interpartition parallelism that involves local and remote data sources.

Figure 10 on page 234 shows how this type of query is processed without interpartition parallelism. The remote nickname data and the local partitioned data are processed serially at the coordinator partition. This technique does not exploit the parallel power of the database partitions because most processing is performed on a single partition. If data volumes are very large, this technique is likely to result in long-running queries.

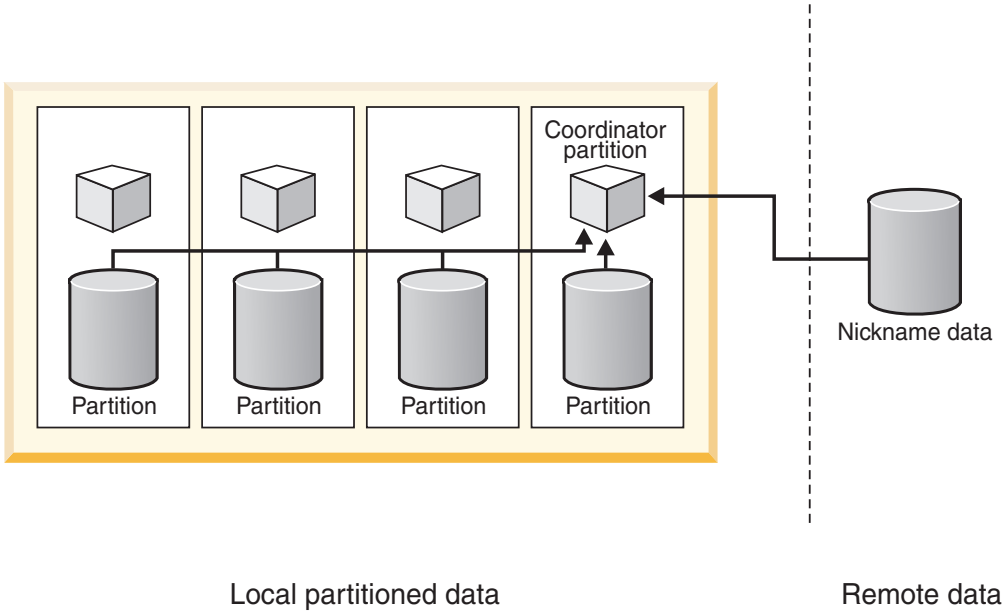


Figure 10. Query without interpartition parallelism of local and remote data sources

Figure 11 shows how processing occurs when the optimizer distributes the nickname data to the partitions. The coordinator partition fetches the nickname data and distributes the data to the database partitions for parallel processing. When parallel processing is complete, the results are sent back to the coordinator partition for final processing before they are returned to the application.

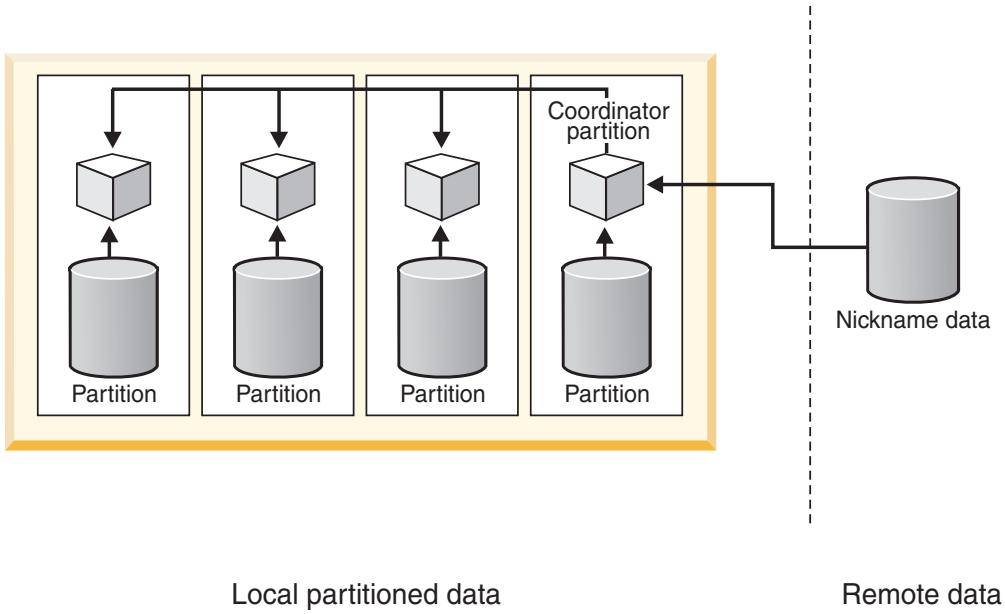


Figure 11. Query with interpartition parallelism of local and remote data sources

Figure 12 on page 235 and Figure 13 on page 235 show the concept of interpartition parallelism that involves only remote data sources.

Figure 12 on page 235 shows serial processing of the remote nickname data at the coordinator partition. The coordinator partition, which also acts as the federated

server, retrieves the nickname data and process it serially.

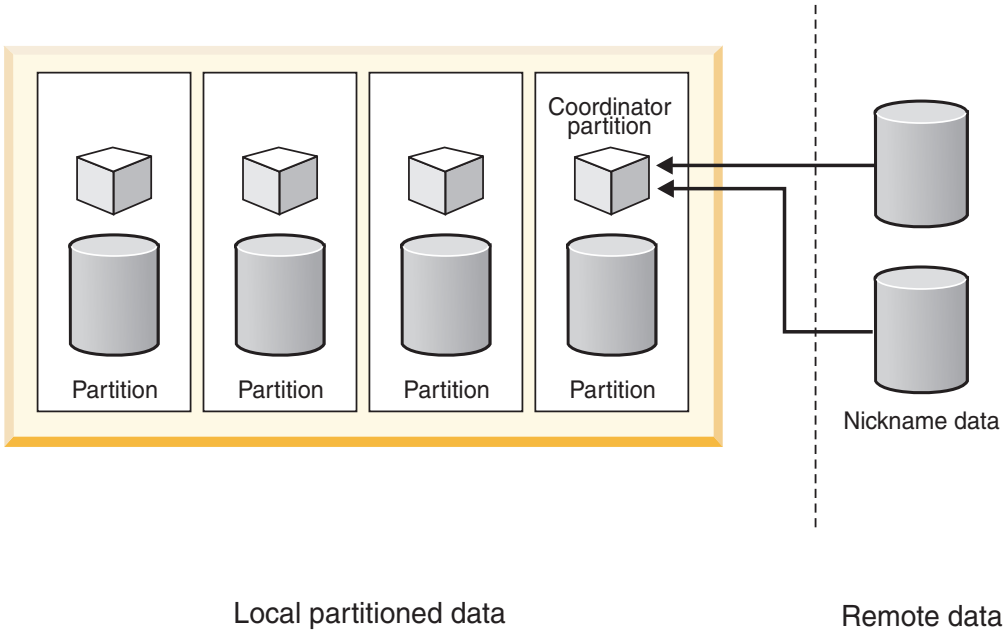


Figure 12. Query without interpartition parallelism that references remote data sources only.

Figure 13 shows the coordinator partition distributing the data across a computational partition group. Computational partition groups allow the optimizer to generate access plans that distribute nickname data across the partitions of a partitioned database server for processing in parallel.

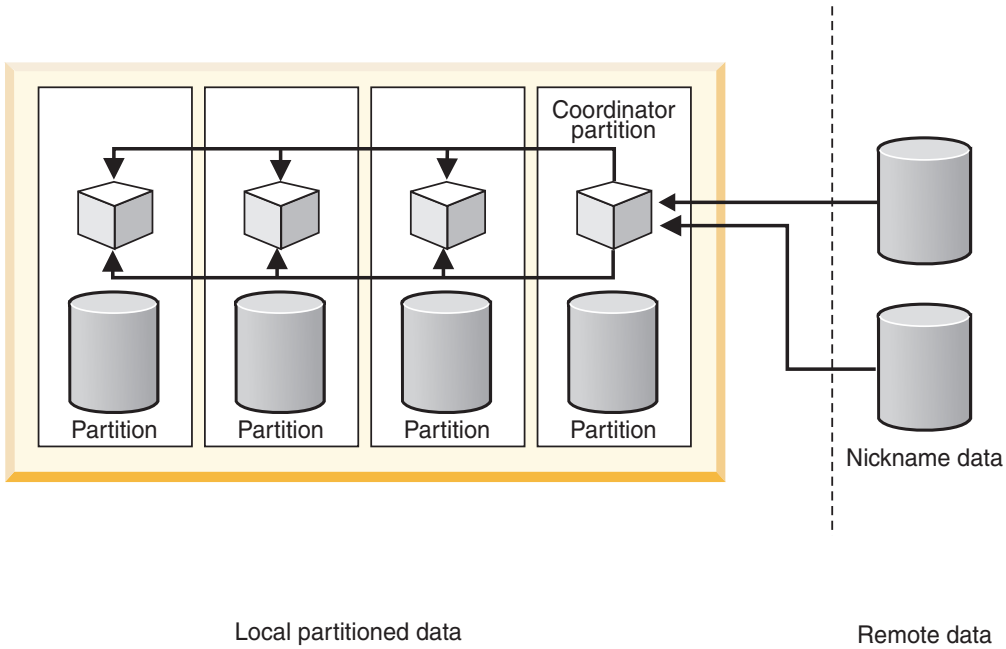


Figure 13. Query with interpartition parallelism that references remote data sources only.

Regardless of the plan that the optimizer chooses, access to the nickname data always occurs serially from the coordinator partition.

Enabling interpartition parallelism with queries that reference nicknames

You can configure a partitioned federated server so that queries involving nicknames can potentially run in parallel on multiple partitions. Parallel execution might significantly reduce the elapsed time of federated queries in a partitioned environment.

About this task

Restrictions

Only parts of a query that reference nicknames that use fenced wrappers can run in parallel. Any parts of a query that reference nicknames that use trusted wrappers cannot run in parallel.

About this task

In a partitioned database environment, federated queries can run in parallel under the following conditions:

- They involve a combination of nicknames that are defined by using a fenced wrapper and local partitioned tables
- They involve nicknames defined using a fenced wrapper, and a computational partition group is defined.

You do not need to set any database parameters or database configuration parameters in a partitioned environment to make interpartition parallelism available for federated queries.

Procedure

To enable interpartition parallelism:

Procedure

1. Issue the CREATE WRAPPER or ALTER WRAPPER statement with the DB2_FENCED option set to Y.
2. **Optional:** Set up a computational partition group, if you are interested in enabling parallelism for portions of queries that involve only nicknames. If queries involve a combination of nicknames and local partitioned tables, you do not need to set up a computational partition group.

Interpartition parallelism with queries that reference nicknames - examples of access plans

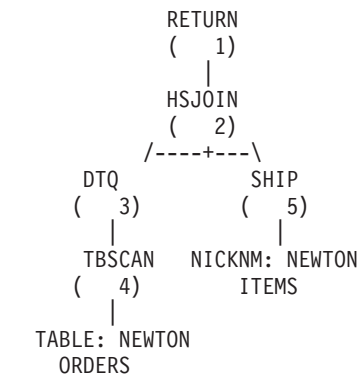
You can use the DB2 Explain facility to view the access plan that the optimizer uses during query processing. The following examples shows how the optimizer accesses nickname data in an interpartition parallelism environment.

Example 1: Trusted mode

In this example, the nickname uses a trusted wrapper. The database serially performs the join between the local table and the nickname at the coordinator partition. The database brings the local data, which is distributed over two partitions, to the coordinator partition. The federated server then joins the local data with the nickname data. The database serially joins nicknames that are

defined by using a trusted wrapper at the coordinator partition. The database cannot distribute the data across multiple partitions to create a parallel join.

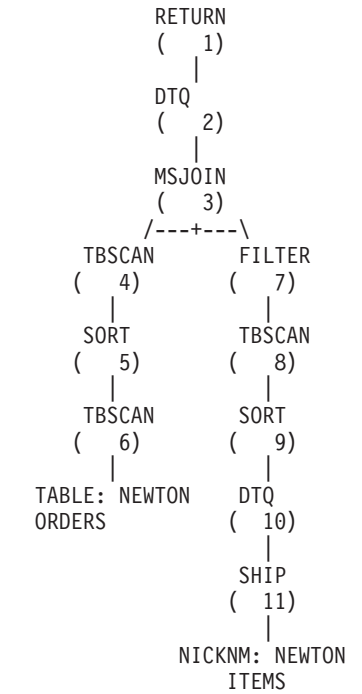
```
SELECT *
FROM ORDERS A, ITEMS B
WHERE A.ID1 = B.ID1 AND B.ITEM = 3
```



Example 2: Fenced mode

In this example, the nickname uses a fenced wrapper. The federated server distributes the nickname data to the other partitions and performs the join with the local data in parallel. The DTQ (Distributed Table Queue) operator above the SHIP indicates that the nickname data is distributed to the local partitions using hash partitioning to achieve a co-located parallel join. In a co-located parallel join, nickname data is distributed to the local partitions in such a way that matching nickname and local data for the join will always be located on the same partition.

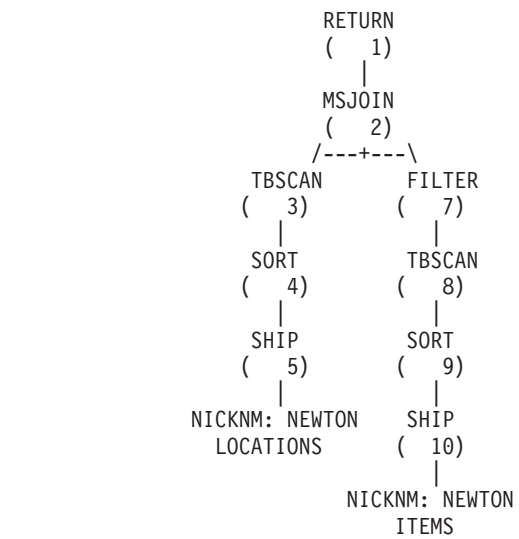
```
SELECT *
FROM ORDERS A, ITEMS B
WHERE A.ID1 = B.ID1 AND B.ITEM = 3
```



Example 3: Fenced mode without a computational partition group

In this example, the two nicknames use a fenced wrapper, and a computational partition group is not defined. The federated server performs the join at the coordinator partition. The federated server does not distribute the data to the other partitions for processing. The lack of TQ operators above any of the SHIP operators indicates that the nickname data is not distributed across the partitions.

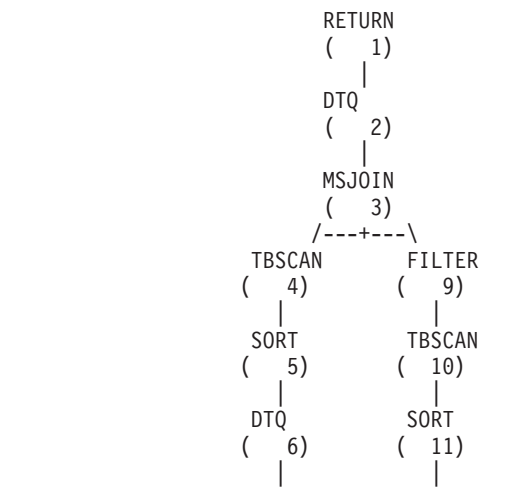
```
SELECT *
FROM ITEMS A, LOCATIONS B
WHERE A.ID1 = B.ID1
```

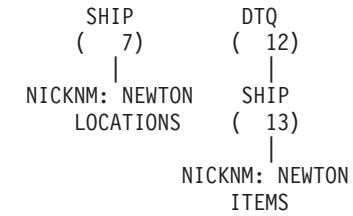


Example 4: Fenced mode with a computational partition group

In this example, the nicknames use fenced wrappers, and a computational partition group is defined. In this case, the optimizer selects a plan that distributes the data from the coordinator partition to the other partitions in the computational partition group. The DTQ operators above both nicknames hash-partition the incoming remote data so that matching join keys are located on the same partition of the computational partition group. The join takes place in parallel on each partition, and the results are then collected at the coordinator partition.

```
SELECT *
FROM ITEMS A, LOCATIONS B
WHERE A.ID = B.ID
```





Computational partition groups

A computational partition group defines a set of database partitions that the optimizer can use to dynamically redistribute nickname data. A computational partition group is for the portions of queries that involve only nicknames.

The coordinator partition fetches nickname data serially and then redistributes the data across the partitions in the computational partition group, at which point parallel processing occurs. The use of computational partition groups by the optimizer often results in performance improvements, particularly when large amounts of nickname data are involved or the queries are complex.

A computational partition group is a database partition group, other than IBMCATNODEGROUP, that is specified in the system catalog, SYSCAT.DBPARTITIONGROUPS.

You use the DB2_COMPPARTITIONGROUP registry variable to specify the computational partition group.

Defining a computational partition group

Defining a computational partition group enables the optimizer to use a plan that distributes nickname data to the partitions of the computational partition group. You define a computational partition group to enable interpartition query parallelism for queries or parts of queries that reference only nicknames.

Before you begin

Before you begin

All partition groups used to represent the computational partition group on all the databases in the instance must have the same name. You can define these partition groups differently in each database, but they must have the same name. For example, three databases called DB1, DB2, and DB3 define a computational partition group that contains different nodes:

- DB1: CPG contains nodes 1, 2, 3, and 4
- DB2: CPG contains nodes 49, 50, and 53
- DB3: CPG contains nodes 78 and 96

You can set the db2set variable to the name CPG. The name CPG is common to all databases, but the contents of the CPG are different for each database.

About this task

Restrictions

The optimizer uses computational partition groups for only the parts of a query that reference nicknames without referencing local data.

Procedure

To define a computational partition group, issue the following command at the DB2 command line.

```
db2set DB2_COMPPARTITIONGROUP=partitiongroup_name
```

partitiongroup_name is the name of the partition group that you want to define as the computational partition group. The partition group must already be defined.

The following example shows how to define the computational partition group, FINANCE3, using the DB2_COMPPARTITIONGROUP registry variable.

```
db2set DB2_COMPPARTITIONGROUP=FINANCE3
```

Interpartition parallelism with queries that reference nicknames - performance expectations

For queries that reference a combination of local partitioned tables and nicknames, the optimizer can choose an execution plan that redistributes nickname data across appropriate partitions.

Redistribution plans can make queries run faster if the amount of nickname data in the join is smaller than the amount of local partitioned data. If the amount of nickname data in the join is considerably larger than the local data, then a parallel plan with redistribution of the nickname data is unlikely to be used. If the optimizer does not choose a parallel plan, the federated server performs the joins serially between nicknames and local tables at the coordinator partition.

For joins between two nicknames, an execution plan that distributes the data among all partitions of a computational partition group can be beneficial if it involves a large amount of data. The advantage of processing the large join in parallel offsets the additional cost of redistributing the data across multiple partitions. If the amount of nickname data is relatively small, the join is not expensive enough to merit the extra cost of redistributing the data across partitions. In general, the optimizer chooses computational partition group plans if the nicknames involved are large; otherwise, the federated server joins the nicknames serially at the coordinator partition.

Mixed parallelism with queries that reference nicknames

For queries that contain local tables and nicknames in a partitioned environment, the optimizer can use both intrapartition and interpartition parallelism. Interpartition parallelism is an option for the optimizer in a partitioned environment. Intrapartition parallelism is an option, if it has been enabled in the database configuration or database manager configuration.

For interpartition parallelism, the federated server can distribute remote data among partitions and process data in parallel within each partition.

For intrapartition parallelism, multiple subagent processes within a partition are used to process local data in parallel.

Enabling mixed parallelism with queries that reference nicknames

You can improve the performance of queries that reference local and remote data by the use of intrapartition and interpartition parallelism.

About this task

Procedure

To enable interpartition parallelism on a partitioned federated server:

1. Issue the CREATE WRAPPER or ALTER WRAPPER statement with the DB2_FENCED option set to Y.
2. **Optional:** Set up a computational partition group to enable parallelism for nickname-only joins.

To enable intrapartition parallelism on a federated server:

1. Set the MAX_QUERYDEGREE database configuration parameter to a value greater than 1.
2. Set the DFT_DEGREE database configuration parameter to a value greater than 1, or you must set the special register CURRENT DEGREE. If you set the DFT_DEGREE parameter to ANY, the default level of intrapartition parallelism equals the number of SMP processors on the computer.

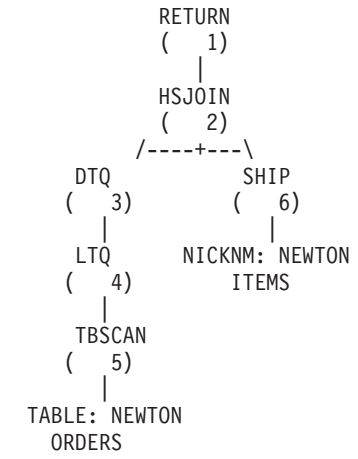
Mixed parallelism with queries that reference nicknames - examples of access plans

You can use the DB2 Explain facility to view the access plan that the optimizer uses during query processing. The following examples shows how the optimizer accesses nickname data in an environment that uses both intrapartition parallelism and interpartition parallelism.

Example 1: Trusted mode

The following example shows a join between a local table and a nickname in trusted mode. The federated server processes the local data in parallel in each partition before it joins the local data with the nickname data at the coordinator partition. The federated server does not process the nickname data in parallel across the partitions or the processors on any given partition.

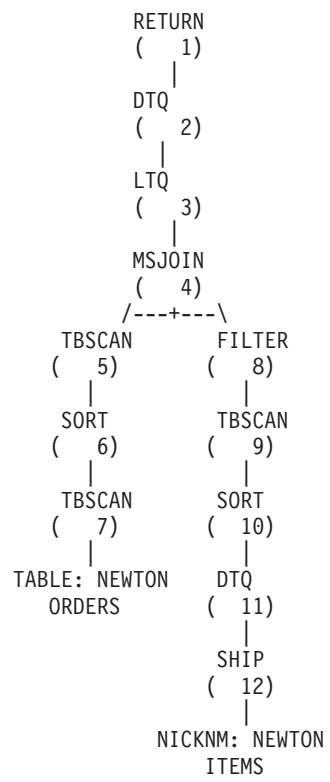
```
SELECT *
FROM ORDERS A, ITEMS B
WHERE A.ID1 = B.ID1 AND B.ITEM = 3
```



Example 2: Fenced mode

The following example shows a join between a local partitioned table and a nickname in fenced mode. The federated server serially fetches the nickname data onto the coordinator partition and then distributes this data to the other partitions in the database. The data is then joined in parallel with the local data across all appropriate database partitions. Within each partition, multiple subagents are reading the local table and joining to the nickname data. This operation is intrapartition parallelism, identified in the plan by the LTQ operator. The result is returned to the coordinator partition for final processing and returned to the application.

```
SELECT *
FROM ORDERS A, ITEMS B
WHERE A.ID1 = B.ID1 AND B.ITEM = 3
```



Chapter 27. Asynchronous processing of federated queries

Asynchrony is a method of improving query performance by running multiple parts of an access plan concurrently to reduce the elapsed time for a given query.

In a federated system, data is distributed across systems at multiple data sources, and each system has its own resources. Asynchrony overlaps the operations that use those resources so that multiple systems remain active at the same time. Overlapping operations enables multiple parts of an access plan to run concurrently rather than serially.

Complex queries that involve time-consuming operations on remote data sources can benefit from asynchrony. Asynchrony enables the following types of operations to take place concurrently:

- Two or more operations on remote data sources
- Operations on the federated server and at least one remote data source

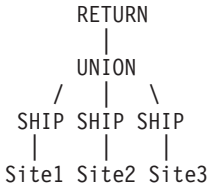
As query operations consume resources, asynchrony can benefit systems in which resources are idle or when only one of the data sources, or the federated system, is doing work at any point in time.

Asynchronous processing of federated queries - examples

Examples of queries with a union and with a merge join illustrate the difference between query operations with and without asynchronous processing.

Example: Query with a union operation

A simple query performs a union operation on data from three different data sources. The computation required to generate the data on each data source is time-consuming. The access plan looks like this:

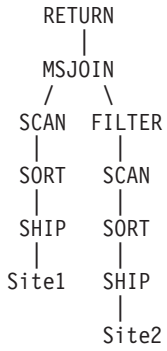


Without asynchrony, the union operation reads data from the query branches one branch at a time, from left to right. When the data from the Site1 server is read, Site2 server and Site3 server are idle. For this example, each branch of the union takes about two hours to return the result rows from one site. The total execution time of the three branches is approximately the sum of the time it takes to process each branch, in this example, about six hours.

With asynchrony, each branch of the union starts to process at the same time, and the three remote servers are active concurrently. The run time of the query is roughly equivalent to the run time of the slowest branch of the union. In this example, the run time is reduced to approximately two hours (about 66% faster) in comparison to six hours without asynchrony.

Example: Query with a merge join operation

A query that joins data from two different data sources uses a merge join operation (MSJOIN). The optimizer access plan looks like this:



Without asynchrony, the merge join operator first processes the outer (left) branch and does not process the inner (right) branch until the left branch starts to return rows. For this example, each branch executes a complex query and therefore takes a long time to execute. The approximate total time to execute the merge join is the sum of the time it takes to execute each branch.

With asynchrony, both branches of the merge join start at the same time, thus reducing the overall execution time of the query.

Asynchrony optimization

The query optimizer makes decisions about the asynchronous processing of remote operations in a query execution plan. *Asynchrony optimization* is the process by which the optimizer analyzes an existing query execution plan and looks for opportunities to allow remote operations to execute concurrently.

Access plans without asynchrony

In an execution plan, the SHIP or RPD operator defines a portion of the plan that is executed at a remote data source.

Without asynchrony, the SHIP or RPD operator becomes active and initiates remote processing only when its data is required by other operators located above the SHIP or RPD operator in the execution plan.

Access plans optimized for asynchrony

The optimizer can make the remote operations that the SHIP or RPD operator defines execute asynchronously.

In an asynchronous operation, a table queue (TQ) operator is inserted directly above the SHIP or RPD operator in the execution plan. The TQ operator defines a portion of the plan, called a *subplan*. A separate process or thread, with its own memory, runs the subplan. A subplan initiates immediately when the query starts.

You can think of the TQ operator as a pipe between the SHIP or RPD operator (producer of data) and the operator above it (consumer of data) in the plan. This pipe decouples the execution of the SHIP in the subplan below it from the main plan, and allows the asynchronous exchange of data between the two plan sections.

A TQ operator that appears directly above a SHIP or RPD operator in the plan enables the remote operations that the SHIP or RPD operator define to initiate at

the beginning of the query and to deliver results to the federated server asynchronously. When asynchrony is beneficial for a given remote operation, the optimizer places a TQ operator directly above the corresponding SHIP or RPD operator in the plan.

TQ operators occur in execution plans for different purposes. A TQ operator usually denotes parallel operations in partitioned databases or in databases enabled for intrapartition parallelism. Another type of TQ operator, that enables asynchronous execution of a subplan, is called an asynchrony TQ (ATQ).

The optimizer makes a given SHIP or RPD operator asynchronous when:

- Query performance will improve
- The number of ATQs is below the per-server and per-query limits
- The operator is not already asynchronous due to the use of another optimization technique
- Restrictions on asynchrony are not violated.
- The semantics of the query do not change

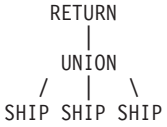
Access plans - examples

Examples of access plans illustrate the difference between plan execution with and without asynchrony optimization.

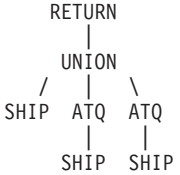
The first two examples show how the union and merge join plans in "Asynchronous processing of federated queries - examples" on page 243 look when asynchrony is enabled.

For simplicity, the examples show plans with SHIP operators only. Asynchrony optimization transforms the plan the same way for RPD operators as for SHIP operators. SHIP and RPD operators are interchangeable, unless otherwise noted.

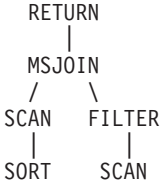
Example 1a: Plan without asynchrony

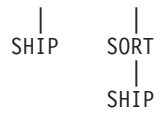


Example 1b: Plan with asynchrony

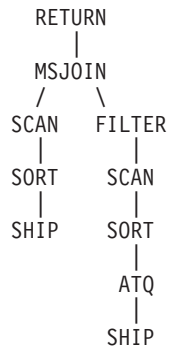


Example 2a: Plan without asynchrony

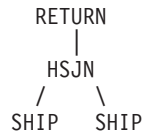




Example 2b: Plan with asynchrony

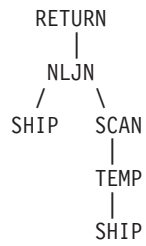


Example 3a: Plan without asynchrony

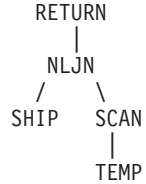


Example 3b: Plan with asynchrony

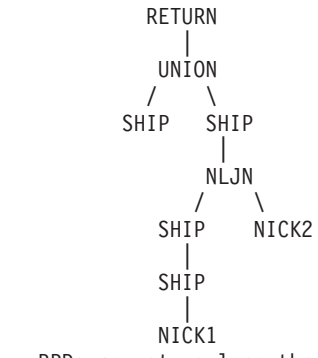
Example 4a: Plan without asynchrony



Example 4b: Plan with asynchrony

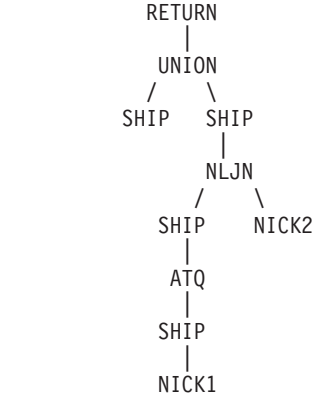


Example 5a: Plan without asynchrony

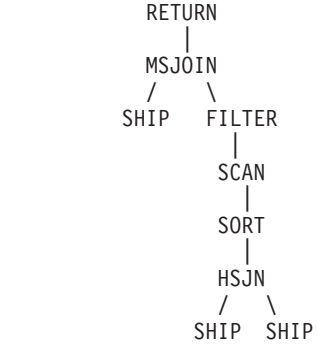


RPDs cannot replace the SHIP-SHIP pair in this plan.

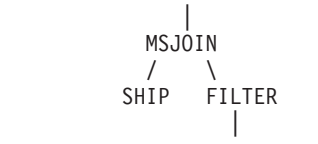
Example 5b: Plan with asynchrony

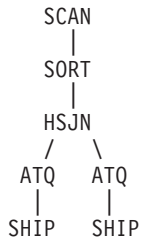


Example 6a: Plan without asynchrony

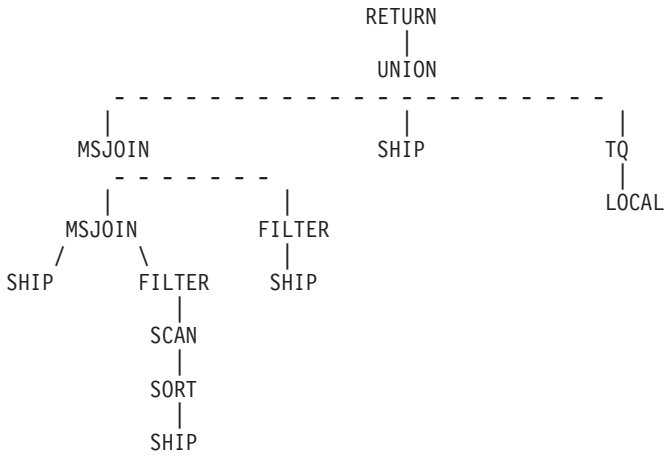


Example 6b: Plan with asynchrony

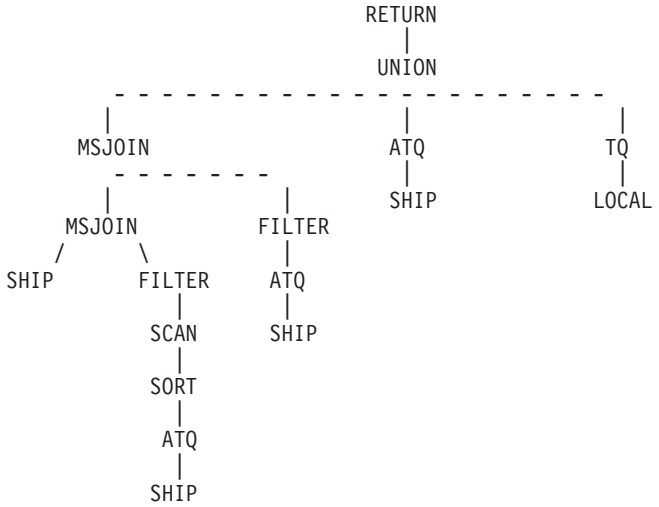




Example 7a: Plan without asynchrony



Example 7b: Plan with asynchrony



Controlling resource consumption

In addition to enabling asynchronous execution of a remote query, the ATQ operator affects the federated server and remote data sources.

Because each ATQ operator creates a new process, and consumes some memory (for buffering), inserting numerous ATQs into an execution plan might use too many system resources on the federated server. In addition, if several SHIP or RPD operators in a query execute on a particular remote data source, making several of the operators asynchronous opens multiple concurrent cursors on that data source and can produce an unacceptable load.

To control resource consumption on the federated system, or on the data sources, you can set configuration parameters. The parameters define limits on the total number of ATQs allowed within a query and on the number of ATQs allowed for each server within a query.

Enabling asynchrony optimization

To enable asynchrony optimization, you specify the number of asynchrony TQ operators for a given query and set a server option for the data source.

About this task

Restrictions

Asynchrony optimization requires:

- A federated system with the database partitioning feature (DPF) that includes more than one logical database partition.
- Access to data sources through fenced wrappers.

This optimization does not support these objects:

- Nicknames on a data source that are accessed through a trusted wrapper.
- Queries with insert, update, or delete operations.

Procedure

To enable asynchronous processing:

Procedure

1. Set one or more of the following parameters:
   - Set the FEDERATED_ASYNC database manager configuration parameter to a value between 0 and 32 767 inclusive, or to ANY. MAXAGENTS is a database configuration parameter that specifies the maximum number of ATQs allowed per query. The value ANY allows the optimizer to determine the number of ATQs for a given access plan. The default is 0.
   - Optionally, set the FEDERATED_ASYNCHRONY bind and precompile options for the static statements in the package to override the configuration parameter setting for the query. The default is 0.
   - Optionally, set the CURRENT FEDERATED ASYNCHRONY special register to dynamically override the database manager configuration parameter setting and the bind option for the query.

   These parameters form the following hierarchy:

   a. Special register
   b. Bind option
   c. Database manager configuration parameter

   The special register value, if specified, takes precedence over the bind option, which in turn takes precedence over the database manager configuration parameter.
2. Set the DB2_MAX_ASYNC_REQUESTS_PER_QUERY server option to a numeric value. This server option specifies the maximum number of asynchronous requests that the server allows for the query.

   The range for the server option is -1 to 64000.

- The default is 1. Therefore, one SHIP operator or one RPD operator that belongs to a server is considered for asynchrony in a query.
- The default for the ODBC data source only is 0. Asynchrony requires multiple cursors per connection. This value should be 0 for the ODBC wrapper unless the data source supports multiple cursors per connection.

Tuning considerations for asynchrony optimization

When asynchrony is enabled, you need to consider several factors that affect performance.

If your system has available process, memory, and CPU resources, enabling asynchrony can improve the performance of your federated queries. Enabling asynchrony can also increase the use of remote-source systems by the federated server, because a remote source can potentially process more than one request at a time on behalf of a federated query. If your system has resource constraints, asynchrony might degrade performance.

You can tune your system for asynchrony by changing configuration parameters to achieve the degree of asynchrony that is best for your system.

Each asynchronous TQ that asynchrony optimization introduces into a plan requires an additional subagent. If enough subagents are available in the system, consider tuning the MAXAGENTS database manager configuration parameter.

Restrictions on asynchrony optimization

When the optimizer applies asynchrony optimization to a given query, some restrictions apply to the number of ATQs that can be used in the query execution plan.

The number of SHIPs or RPDs that are eligible to be coupled with an ATQ in a plan and benefit from asynchrony might be greater than either the maximum that is set by the FEDERATED_ASYNC parameter or the per server limit of the DB2_MAX_ASYNC_REQUESTS_PER_QUERY server option. In this case, the optimizer chooses SHIPs or RPDs to couple with an ATQ in such a way that:

- The total number of ATQs in the plan is less than or equal to the value set for the following parameters, in the order listed:
  1. FEDERATED ASYNCHRONY special register, if specified
  2. FEDERATED_ASYNCHRONY bind or precompile option, if specified
  3. FEDERATED_ASYNC parameter
- The total number of ATQs for a given server is less than or equal to the DB2_MAX_ASYNC_REQUESTS_PER_QUERY server option for that server.
- The eligible SHIPs or RPDs can benefit from being coupled with an ATQ and improve query performance.

Determining if asynchrony optimization is applied to a query

To determine if asynchrony optimization is applied to a given query, you can use one of several methods to check the access plan for the query and check for a specific operator.

You can check any of the following outputs for the query in question:

- db2exfmt output

- Visual explain output
- dynexpln output

Each output shows asynchronous query requests in the access plan as ATQ. The 'origin' property in the description of the ATQ shows 'Asynchrony'.

The following plan fragment shows how the ATQ operator is used and shows its detailed properties.

```
                         1.6e+06
                         HSJOIN
                         (   2)
                         4213.74
                           131
                      /--------+--------\
                   40000                  1000
                  HSJOIN                 SHIP
                  (   3)                 (  10)
                  1122.26                427.733
                    117                    14
               /------+-----\               |
            1000             1000          1000
           ATQ              ATQ          NICKNM: NEWTON
           (   4)           (   7)         S1_NN07
           511.795          532.669
             16               101
             |                |
            1000             1000
           SHIP             SHIP
           (   5)           (   8)
           478.773          499.887
             16               101
             |                |
            1000             1000
      NICKNM: NEWTON    NICKNM: NEWTON
           S2_NN02          S3_NN15

- show the origin of the ATQ as ASYNCHRONY:
 4) TQ    : (Table Queue)
  Cumulative Total Cost:   511.795
  Cumulative CPU Cost:   2.79486e+06
  Cumulative I/O Cost:   16
  Cumulative Re-Total Cost:  68.9489
  Cumulative Re-CPU Cost:  1.72372e+06
  Cumulative Re-I/O Cost:  0
  Cumulative First Row Cost:  30.8308
  Cumulative Comm Cost:  18.2176
  Cumulative First Comm Cost: 0
  Estimated Bufferpool Buffers:  16
  Remote communication cost: 538.297

  Arguments:
  ---------
  JN INPUT: (Join input leg)
   OUTER
  LISTENER: (Listener Table Queue type)
   FALSE
  TQMERGE : (Merging Table Queue flag)
   FALSE
  TQORIGIN: (Table Queue Origin type)
   ASYNCHRONY
  TQREAD  : (Table Queue Read type)
   READ AHEAD
  TQSEND  : (Table Queue Write type)
   DIRECTED
  UNIQUE  : (Uniqueness required flag)
   FALSE
```

If you do not find any ATQ operators in the access plan for a query, verify that the following conditions are met:

- The system is a federated system that is enabled for DPF.
- The query accesses nicknames on a data source that is accessed through a fenced wrapper.
- Asynchrony is enabled using the database manager configuration parameter FEDERATED_ASYNC, the special register CURRENT FEDERATED ASYNCHRONY, or the bind option FEDERATED_ASYNCHRONY.
- The server option DB2_MAX_ASYNC_REQUESTS_PER_CONNECTION is set to a non-zero value for the nicknames at each server.

Chapter 28. Global optimization

The SQL compiler works in three phases, which help to produce an optimal access strategy for evaluating a query referencing a remote data source. These phases are pushdown analysis, global optimization, and remote SQL generation.

For a query submitted to the federated database, the access strategy might involve breaking down the original query into a set of query fragments and then combining the results of these query fragments.

Using the output of the pushdown analysis phase as a recommendation, the query optimizer decides where each operation is evaluated. An operation might be evaluated locally at the federated server or remotely at the data source. The decision is based on the output of the sophisticated cost model that the optimizer uses. This model determines:

- The cost to evaluate the operation
- The cost to transmit the data or messages between the federated server and the data sources

The goal of global optimization is to produce an access plan that optimizes the query operations on all data sources globally, across the federated system. An access plan that is globally optimal has the least overall cost of execution in a federated system. The remote SQL generation phase reverse translates the globally optimal plan into query fragments that are executed by individual data sources.

The SQL compiler has a knowledge base that contains characteristics of supported data sources and metadata about the data at those data sources. The optimizer does not generate SQL, query fragments, or plan hints that the remote data source cannot understand or accept.

Many factors can affect the output from global optimization and thus affect query performance. The key factors are server characteristics and nickname characteristics.

Relational and nonrelational wrappers differ in the details of how an access plan is produced, but the concept and final effect are the same.

Server characteristics affecting global optimization

When you create or alter a server definition, some of the options that you choose can affect query performance.

You provide the query optimizer with information about the data source server characteristics through the server option settings. The server option settings are part of the data source server definition. You can set server options in the CREATE SERVER statement, when you initially establish the server definition. Use the ALTER SERVER statement to add server options to an existing server definition. The server option settings are stored in the federated database global catalog.

These options are classified as location options (such as the data source computer name), security options (such as authentication information), and performance options (such as the CPU ratio).

The performance options help the optimizer determine if the evaluation of an operation can be done at a data source and if the evaluation of an operation on the data source makes execution faster. The server options affecting performance that might require your tuning are:

- CPU_RATIO
- IO_RATIO
- COMM_RATE
- COLLATING_SEQUENCE
- PLAN_HINTS
- VARCHAR_NO_TRAILING_BLANKS
- NO_EMPTY_STRING

Use caution when tuning the CPU_RATIO, IO_RATIO, or COMM_RATE server options as you can get unexpected errors if the cost calculation for a query causes overflows or underflows. In most cases, the default values for these options are sufficient. Typically, ensuring that the statistics about the objects referenced in your queries are correct is more important than tuning the values of these server options.

Relative ratio of CPU speed

This value indicates the ratio between the CPU speed of the federated server and the CPU speed of the server on which the remote data source is located.

This value is defined as the CPU speed of the federated server divided by the CPU speed of the server for the remote data source. For example, if the CPU speed for the federated server is twice as fast as the CPU speed for the remote server, then CPU_RATIO should be set to 2. If the CPU speed for the federated server is only one third as fast as the CPU speed for the remote server, then CPU_RATIO should be set to 0.33.

When you do not set the CPU ratio server option explicitly, the federated optimizer uses a default value of 1, which indicates that the federated CPU speed and the data source CPU speed are equal.

A low ratio indicates that the data source server CPU is faster than the federated server CPU. For low ratios, the optimizer will consider pushing-down operations that are CPU-intensive to the data source. A low ratio is a value that is less than 1.

Relative ratio of I/O speed

This value indicates the ratio between the I/O rate of the federated server and the I/O rate of the server on which the remote data source is located.

This value is defined as the I/O rate for the federated server divided by the I/O rate for the remote server. For example, if the I/O rate for the federated server is twice the I/O rate for the remote server, then IO_RATIO should be set to 2. But if the I/O rate for the federated server is half that of the remote server, then IO_RATIO should be set to 0.5.

When you do not set the I/O ratio server option explicitly, the federated optimizer uses a default value of 1, which indicates that the I/O rates of both the federated and remote servers are equal.

A low IO_RATIO of less than one indicates that the remote server has a higher I/O rate than the federated server. In this case, the optimizer will tend to favor pushing down I/O-intensive operations to the remote data source. A low ratio is a value that is less than 1.

Communication rate between the federated server and the data source

A low communication rate indicates slow network communication between the federated server and the data source.

The setting of the COMM_RATE server option determines the communication rate. The COMM_RATE represents the speed of the network connection between the data source server and the federated server. The rate is measured in megabytes per second. The default is 2 MBPS.

Lower communication rates encourage the query optimizer to reduce the number of messages sent to or from this data source. If the COMM_RATE server option is set to a very small number, the optimizer produces a query requiring minimal network traffic.

Data source collating sequence

The collating sequence that you choose might affect performance of the federated database. You can use the COLLATING_SEQUENCE server option to indicate if a data source collating sequence matches the local federated database collating sequence.

The federated server can push down order-dependent processing that involves character data to the data source, if the COLLATING_SEQUENCE server option indicates that the collating sequence of the data source and the federated database match. If a data source collating sequence does not match the federated database collating sequence, the optimizer considers data that is retrieved from this data source unordered. The federated database will retrieve the relevant data and perform all order-dependent processing on character data locally, which can slow down the query and affect performance.

Remote plan hints

Plan hints are statement fragments that provide extra information to data source optimizers.

Use the PLAN_HINTS server option to generate remote plan hints. This information can, for certain query types, improve query performance. The plan hints can help the data source optimizer decide whether to use an index, which index to use, or which table join sequence to use.

You should run some tests to determine if this server option will improve the performance of your queries.

You cannot code your own plan hints in a query.

If plan hints are enabled, the query sent to the data source contains additional information. For example, a statement sent to an Oracle optimizer with plan hints could look like this:

```
SELECT /*+ INDEX (table1, tlindex)*/
 col1
 FROM table1
```

The plan hint is the string `/*+ INDEX (table1, tlindex)*/`

Nickname characteristics affecting global optimization

There are several nickname-specific factors that can affect global optimization, including the index information and the global catalog statistics.

It is important that the index information and global catalog statistical data available to the SQL compiler is current.

Index specifications

The SQL compiler uses index information to optimize queries.

The index information for a data source table is only acquired when the nickname is created for that table. After the nickname is created, any changes to the index on that data source table are not updated on the federated server. When the remote index information changes, you can update the index information stored on the federated server by dropping the nickname for the table and creating the nickname again. Alternatively, if a new index is added for the data source table, you can define an index specification for the nickname on the federated server.

Index information is not gathered for nicknames on objects that do not have indexes such as views, synonyms, or nonrelational data source objects.

If an object that has a nickname defined for it does not have an index, you can create an index specification for it. Index specifications build an index definition in the global catalog. The index specification is not an actual index. Use the CREATE INDEX statement with the SPECIFICATION ONLY clause to create an index specification. The syntax for creating an index specification on a nickname is similar to the syntax for creating an index on a local table.

Consider creating index specifications when:

- A table acquires a new index.
- You create a nickname for a data source object that does not contain indexes such as a view or a synonym.

When you create an index specification (SPECIFICATION ONLY) on a nickname and specify that the index is unique, the federated database does not verify that the column values in the remote table are unique. If the remote column values are not unique, then queries against the nickname that include that index column might return incorrect data or result in errors.

Consider your needs before issuing CREATE INDEX...SPECIFICATION ONLY statements on a nickname for a data source view:

- If the remote view is a simple SELECT statement on a data source table with an index, creating an index specification on the nickname that matches the index on the data source table can significantly improve query performance.
- If an index specification is created for a remote view that is not a simple SELECT statement (for example, a view created by joining two tables), query performance might suffer.

For example, consider an index specification that is created for a remote view that is a join of two tables. The optimizer can choose that view as the inner element in a nested loop join. The query might have poor performance because the join will be evaluated several times. An alternative is to create nicknames for each of the tables referenced in the data source view and create a federated view that references both nicknames.

Global catalog statistics

The global catalog on the federated server contains statistical information that is used to optimize queries.

The federated server relies on statistics for data source objects for which nicknames have been defined to optimize queries that involve those nicknames. These statistics are retrieved from the data source when you create a nickname for a data source object using the CREATE NICKNAME statement. The federated database verifies the presence of the object at the data source, and then attempts to gather existing data source statistical data. Information useful to the optimizer is read from the data source catalogs and put into the global catalog on the federated server. Because some or all of the data source catalog information might be used by the optimizer, it is advisable to update statistics (using the data source command equivalent to RUNSTATS) at the data source before you create a nickname.

Catalog statistics describe the overall size of tables and views, and the range of values in associated columns. The information retrieved includes, but is not limited to:

- The number of rows in a nickname object
- The number of pages that a nickname occupies
- The number of distinct values in each column of a table
- The number of distinct values in columns of an index
- The highest/lowest values of a column

While the federated database can retrieve the statistical data held at a data source, it cannot automatically detect updates to existing statistical data at data sources. Furthermore, the federated database has no mechanism for handling object definition or structural changes to objects at the data sources (such as when a column is added to a table).

If the statistical data or structural characteristics for a remote object on which a nickname is defined change, you have the following choices for updating the statistics:

- Manual statistics collection
  - Run the equivalent of RUNSTATS at the data source. Then drop the current nickname and create the nickname again. This is the recommended method for updating statistics.

    An advantage of this method is that in addition to updated statistics, any information about new indexes or structural changes to the remote object will be reflected in the new nickname. A disadvantage of this method is that any views or packages based on the old nickname will be invalidated.
  - Use the stored procedure SYSPROC.NNSTAT(), available from the command line processor.

    SYSPROC.NNSTAT() only updates the nickname statistics; it does not alter the nickname to match any structural changes to the remote object. For

example, if the remote object has a new column, then updating nickname statistics does not add a column to the nickname.

– Manually update the statistics in the SYSSTAT.TABLES catalog view. Use this method only when you know that the statistical information on the remote data source is incorrect or incomplete.

• Automatic statistics collection

  This feature runs by default to improve performance by collecting up-to-date table and nickname statistics automatically.

Updating row changes

If a large number of rows are added to or deleted from an object at the data source, the federated database is not aware of these changes because the catalog statistics for the nickname continue to indicate the old number of rows.

However you might notice degradation in performance because the optimizer continues to make decisions based on nickname statistics information that is no longer accurate. After updating statistics for the remote object at the data source, you can update the statistics for the nickname to ensure that the optimizer can use accurate statistics when it generates and chooses access plans for processing queries on the data source.

Updating statistics when columns change

When there are structural changes to a data source object, for example, when a column is added to a table, you must complete several steps to update the statistics for that object in the federated database catalog.

About this task

About this task

If columns at the data source are added, deleted, or altered, you might notice incorrect results or receive an error message. For example, assume that the nickname *EUROSALES* refers to the *europe* table in a Sybase database. If a new column called *CZECH* is added to the table, the federated database will not be aware of the *CZECH* column. Queries that reference that column will result in an error message.

Procedure

To update the statistics for that object when column changes occur:

Procedure

1. Run the utility on the data source that is equivalent to DB2 RUNSTATS. This will update the statistics stored in the data source catalog.
2. Drop the current nickname for the data source object using the DROP NICKNAME statement.
3. Re-create the nickname using the CREATE NICKNAME statement.

Analyzing global optimization

Detailed information about access plans, including some of the information that the global optimizer uses to choose the optimal plan, is kept in explain tables separate from the actual access plan itself.

This information allows for in-depth analysis of an access plan. The explain tables are accessible on all supported operating systems, and contain information for both static and dynamic SQL statements. You can access the explain tables using SQL statements. This allows for easy manipulation of the output, for comparison among different queries, or for comparisons of the same query over time.

There are multiple ways to get global access plan information from the Explain tables:

- You can use the Explain table format tool, db2exfmt, to present the information from the explain tables in a predefined format.
- You can use the db2expln and dynexpln tools to understand the access plan that is chosen for a particular SQL statement.

  To fully understand the output of db2exfmt, Visual Explain, db2expln, or dynexpln you must understand:

  – The different SQL statements supported and the terminology related to those statements (such as predicates in a SELECT statement)
  – The purpose of a package (access plan)
  – The purpose and contents of the system catalog tables
  – Basic query processing operators such as joins, group-by, aggregation, and sorts

Understanding access plan optimization decisions

This section lists typical optimization questions, and areas you can investigate to improve performance.

Why isn't a join between two nicknames of the same data source being evaluated remotely?

You can check elements of the join operation, join predicates, and the number of rows in the result to evaluate why a join between two nicknames of the same data source is not evaluated remotely

Areas to examine include:

- Join operations. Can the data source support a join?
- Join predicates. Can the join predicate be evaluated at the remote data source?
- Number of rows in the join result. You can determine the number of rows with Visual Explain. Does the join produce a much larger set of rows than the two nicknames combined? Do the numbers match reality? If the answer is no, consider updating the nickname statistics with the SYSPROC.NNSTAT() stored procedure.

Why isn't the GROUP BY operator being evaluated remotely?

Areas to examine include:

- Operator syntax. Verify that the operator can be evaluated at the remote data source.
- Number of rows. Check the estimated number of rows in the GROUP BY operator input and output using Visual Explain. Are these two numbers very close? If the answer is yes, the optimizer considers it more efficient to evaluate this GROUP BY locally. Also, do these two numbers reflect reality? If the answer is no, consider updating the nickname statistics using the SYSPROC.NNSTAT() stored procedure.

Why is the statement not being completely evaluated remotely?

The federated server seeks to ensure that query semantics and results obtained for federated queries are exactly the same as if they had been completely evaluated by DB2 Database for Linux, UNIX, and Windows. The pushdown analysis phase of the query compiler decides whether pushing down processing to remote sources will maintain DB2 semantics. Thus, federated query operations can be safely pushed down only if corresponding operations at the remote source have the same meaning and result. The most common reason for failure to completely push down processing of a query to a single remote source is the existence of subtle differences in functionality between the federated server and the remote source for one or more operations in the query.

The optimizer performs cost-based optimization. Even if pushdown analysis indicates that every operator can be evaluated at the remote data source, the optimizer still relies on its cost estimate to generate a globally optimal plan. There are many factors that contribute to the decision to choose that plan. Suppose that the remote data source can process every operation in the original query. However, its CPU speed is much slower than the CPU speed of the federated server. It might turn out to be more beneficial to perform the operations at the federated server instead. If the desired performance is not achieved, verify the server statistics in the SYSSTAT.SERVEROPTIONS catalog table.

Why does a plan generated by the optimizer and completely evaluated remotely, have much worse performance than the original query executed directly at the remote data source?

Areas to examine include:

- The remote SQL statement generated by the query optimizer. In addition to the replacement of nicknames by corresponding remote table names, the generated remote SQL statement typically differs from the original federated statement in the following ways:
  - The ordering of predicates in the query might have changed.
  - Predicates found in the original query might have been removed, replaced by equivalent ones, or augmented by additional predicates.
  - Subqueries might have been rewritten as joins.
  - Additional functions that do conversion or string truncation might have been added to maintain DB2 semantics

  With the exception of the last item listed above, these changes usually have a favorable impact on performance. However, in a few cases, the changes might cause the remote query optimizer to generate a different (and slower) plan than it would have for the original query

  A good query optimizer should not be sensitive to the predicate ordering of a query. Unfortunately, not all DBMS optimizers are identical. It is likely that the optimizer at the remote data source will generate a different plan based on the input predicate ordering. If this is true, this is a problem inherent in the remote optimizer. Consider either modifying the predicate ordering or contacting the service organization of the remote data source for assistance.

  Also, check for predicate replacements. A good query optimizer should not be sensitive to equivalent predicate replacements. It is possible that the optimizer at the remote data source will generate a different plan based on the input predicate. For example, some optimizers cannot generate transitive closure statements for predicates.

- The number of returned rows. You can get this number from Visual Explain. If the query returns a large number of rows, network traffic is a potential bottleneck.
- Additional functions. Does the remote SQL statement contain more functions than the original query? Some of the extra functions might be generated to convert data types. Ensure that they are necessary.

Chapter 29. System monitor elements affecting performance

The federated database system monitor gathers statistical information regarding the current state of the database manager, and activity information such as counters and other measurements of database processing.

In a federated system, you can use the database system monitor to gather information about database activity, system performance, and application performance.

The Timestamp monitor switch is used to track the response times of interactions that the federated database has with a data source. The federated data elements tracked by the timestamp switch are:

- Create nickname response time
- Delete response time
- Insert response time
- Pass-through time
- Query response time
- Remote lock time
- Stored procedure time
- Update response time

The default setting for the Timestamp monitor switch is ON.

Recommendation: You can increase performance by changing the setting for the Timestamp monitor switch to OFF for all applications. If one application has the Timestamp switch set to ON, the system will continue to collect the response times. Therefore, you will not increase performance by turning off the timestamp switch for only some of your applications.

Turning off the switch does have other implications.

- Turning off the Timestamp monitor switch for all applications requires that you stop and restart the DB2 instance to implement the change.
- Turning off the Timestamp monitor switch disables the gathering of timestamp information for both federated and non-federated applications. The local database will not receive timestamp information either.

If you need timestamp information for local non-federated applications, then you should not turn off the Timestamp monitor switch.

You can set the timestamp switch to OFF for all applications by using this command:

```
update dbm cfg using dft_mon_timestamp off
```

Then issue:

```
db2stop
db2start
```

Stopping and starting the federated server will ensure that the switch is off for all applications.

Specific information about each of the elements tracked by the timestamp switch is discussed in a separate topic.

Chapter 30. Materialized query tables

A materialized query table is a table that caches the results of a query. When you submit the query again, the database engine can return the data from the materialized query table. You can use materialized query tables with nicknames to improve the performance of a query.

Materialized query tables are used when you create a cache table. The cache table stores local data that is defined by the materialized query tables that are associated with it.

Materialized query tables and federated systems - overview

A materialized query table is a table that caches the results of a query. When you submit the query again, the database engine can return the data from the materialized query table instead of repeating the query computation.

You can use materialized query tables with nicknames to improve the performance of a query and to encapsulate a part of logic. Materialized query tables are used when you create cache tables.

The SQL optimizer determines if a query will run more efficiently with a materialized query table than the base tables or nicknames. The optimizer uses the following factors to select a materialized query table:

- The materialized query table must match part or all of the query.
- The refresh age criterion must be met.
- The access plan that uses a materialized query table must be cheaper than the access plan that uses the base tables or nicknames.

Materialized query tables that involve nicknames for objects from the following data sources are supported:

- Relational data sources
  - DRDA
  - Informix
  - JDBC
  - ODBC
  - Oracle
  - Sybase
  - Microsoft SQL Server
  - Teradata
- Nonrelational data sources
  - BioRS
  - Excel
  - Table-structured files
  - Web Services
  - XML

Creating a federated materialized query table

You use materialized query tables to cache data locally and to improve the performance of your queries. You can use nicknames from relational and nonrelational data sources to create materialized query tables.

About this task

Restrictions

- "Data source specific restrictions for materialized query tables"
- If a query has a function template in a predicate or a select list, the function template must be part of the materialized query table.
- "Restrictions on using materialized query tables with nicknames" on page 267

Procedure

To create a materialized query table, issue a CREATE TABLE statement that references the nicknames that represent the remote data source objects that you want to include.

You can populate a user-maintained materialized query table by using an INSERT statement in a subselect statement. For example:

```
insert into my_mqt (select ..from n1, n2 where ..)
```

where the select portion of the query matches the materialized query table definition. The optimizer might use my_mqt to replace the select portion of the query. In that case, the statement becomes:

```
insert into my_mqt (select .. from my_mqt);
```

In this case, the materialized query table becomes the source of the insert operation. To prevent this from happening, you can by issue one of the following commands to temporarily disable the materialized query table:

```
SET CURRENT REFRESH AGE 0
SET CURRENT MAINTAINED TABLE TYPE FOR OPTIMIZATION SYSTEM
```

Data source specific restrictions for materialized query tables

When you create materialized query tables, you need to be aware of restrictions for specific data sources.

This topic describes the restrictions on creating materialized query tables for the following data sources:

- BioRS
- Table-structured files
- Web services
- XML

BioRS Search restrictions

The BioRS wrapper requires at least one predicate in the WHERE clause. You must create a materialized query table that satisfies the predicate requirements of the wrapper. If you do not specify a predicate, a refresh of the materialized query table fails.

Table-structured file restrictions

If you define a nickname for a table-structured file with the DOCUMENT option, the materialized query table must have a predicate that specifies the file path. If you do not specify a predicate, a refresh of the materialized query table fails.

Web services restrictions

You can only create a materialized query table over a flattened view of a hierarchy of nicknames. You cannot create a materialized query table for each nickname of a hierarchy.

XML restrictions

You cannot create a materialized table on a child table.

If you define a nickname for an XML table with the DOCUMENT option, the materialized query table requires a predicate that specifies the file path. If you do not specify a predicate, a refresh of the materialized query table fails.

Restrictions on using materialized query tables with nicknames

When you tune your federated system, consider these restrictions on materialized query tables that reference nicknames should be considered when you are tuning your federated system.

Label-based access control (LBAC) for data source objects

Nicknames for data source objects that use label-based access control or Oracle Label Security cannot be cached, and materialized query tables cannot be created on them.

System-maintained materialized query tables

The federated system does not support system-maintained materialized query tables that reference nicknames in a partitioned database environment.

The federated wrapper must be fenced for the federated rewrite facility to route the query to the MQT. You can either create the wrapper as FENCED or change the wrapper by using the ALTER WRAPPER statement. For example:

```
CREATE WRAPPER <wrapper name>
   LIBRARY <libname>
   OPTIONS (DB2_FENCED  'N');
ALTER WRAPPER <wrapper name> OPTIONS (SET DB2_FENCED 'Y');
```

You need to set the maintained table types for optimization to ALL or USER by using the database configuration parameter DFT_MTTB_TYPES or the set current special register issued prior to the SQL.

To change the configuration parameter to the value USER issue:

```
update db cfg for <dbalias> using DFT_MTTB_TYPES USER
```

To use the set current special register, issue:

```
set current maintained table types for optimization ALL
```

To work around this restriction, you can use user-maintained materialized query tables.

For example, for a nonrelational nickname named DEPART, you can issue the following commands to simulate a system-maintained materialized query table.

```
SET CURRENT MAINTAINED TABLE TYPES FOR OPTIMIZATION ALL;

CREATE TABLE AST1(C1, C2)
 AS (SELECT EMPNO, FIRSTNME FROM DEPART WHERE EMPNO>'000000')
 DATA INITIALLY DEFERRED REFRESH DEFERRED
 ENABLE QUERY OPTIMIZATION MAINTAINED BY USER;

SET INTEGRITY FOR AST1 ALL IMMEDIATE UNCHECKED;

INSERT INTO AST1 (SELECT EMPNO, FIRSTNME FROM DEPART WHERE EMPNO>'000000');

SET CURRENT REFRESH AGE ANY;
```

The following SELECT statement can be answered by the materialized query table defined above:

```
SELECT EMPNO, FIRSTNME FROM DEPART
 WHERE EMPNO > '000000' AND FIRSTNME LIKE 'AN%';
```

Chapter 31. Cache tables

You use cache tables to store data that you access frequently but that does not change often.

A cache table can improve query performance by storing the data locally instead of accessing the data directly from the data source.

You can cache data from these data sources:

- DB2 family
- Informix
- Microsoft SQL Server
- Oracle
- Sybase

A cache table consists of these components:

- A nickname on your federated database system. The nickname has the same column definitions and same data access as the data source table.
- One or more materialized query tables that you define on the nickname. The type of materialized query tables is FEDERATED_TOOL maintained materialized query table. The materialized query table usually contains a subset of high-use data from the data source table.
- A replication schedule for each materialized query table. The replication schedule keeps the local materialized query tables current with your data source tables. You define the replication schedule.

The following figure illustrates a cache table.

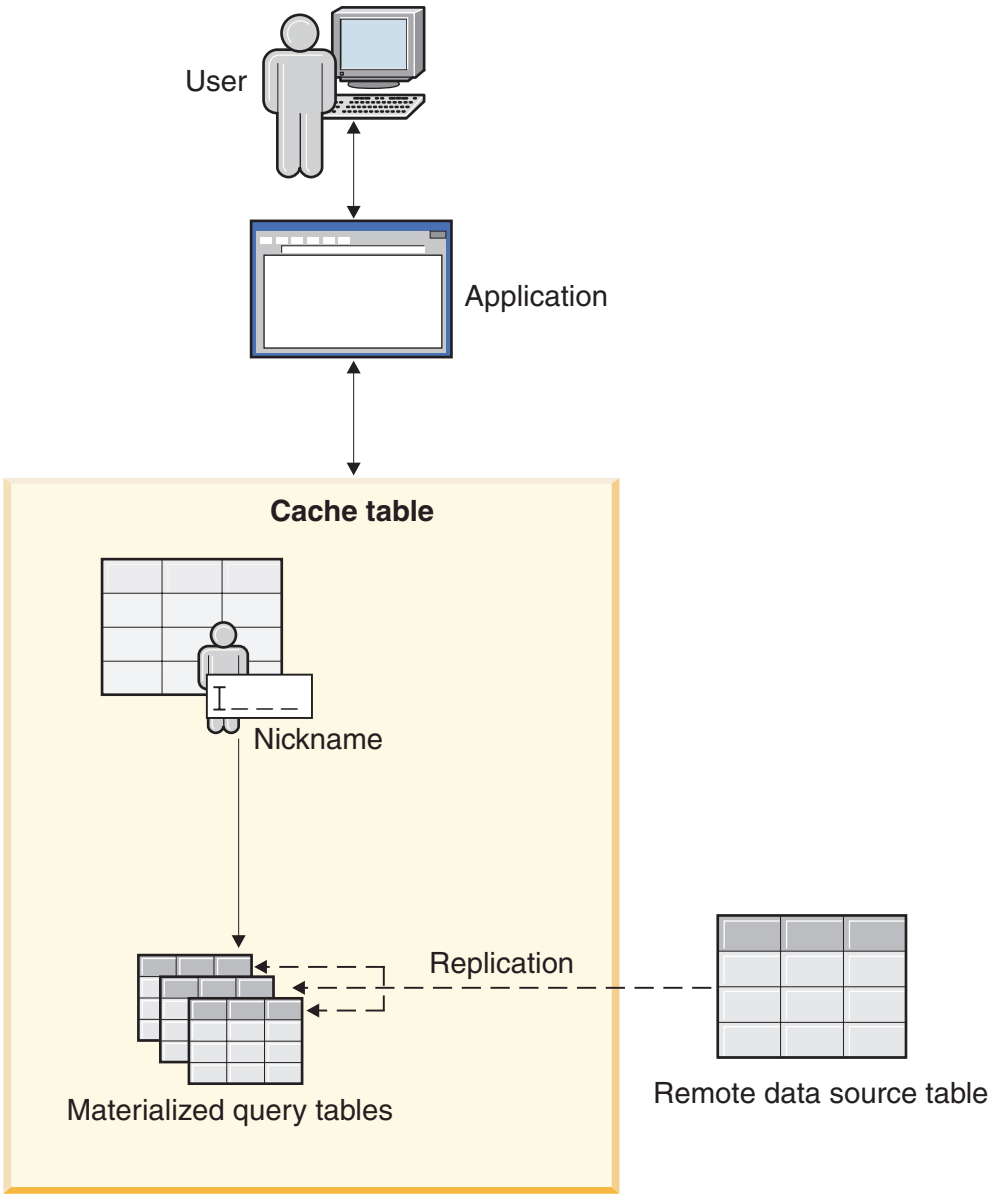


Figure 14. Cache table.

The cache table has the same name as the nickname. You can associate a cache table with only one data source table.

When a cache table is enabled, the query optimizer directs queries to the cache table if the data that the query requests can be found in the materialized query table.

Chapter 32. Creating sample cache tables

You can use a sample to set up distributed caching.

Before you begin

- Set the FEDERATED parameter to YES on the federated server. The FEDERATED parameter is a database manager configuration parameter.
- To access Informix data sources, install and configure the Informix Client software development kit (SDK) in the federated server.
- To cache data from DB2 Database for Linux, UNIX, and Windows tables, configure the DB2 database for archive logging.
- The federated database or the source database must be on the computer from which you are creating the cache tables. If the federated database or the source database are not local, you must catalog the databases on the local computer. The alias name that you use when you catalog the database must be the same name as the database name.
- The user ID in the user mapping between the databases must have the authority to create tables in the source database.

About this task

The sample DB2cacheTables.zip provides command scripts that you can use to perform the following tasks:

- Enable federation and create federated objects in the cache database.
- Create one or more materialized query tables (MQTs) to participate in caching.
- Set up replication to keep cache tables updated.

Procedure

Follow the instructions in the readme file to run the commands in the DB2cacheTables.zip sample.

Chapter 33. Bulk insert

Bulk inserts of data can improve the performance of insert operations to the Netezza data source.

Federation supports bulk inserts of data to the Netezza data source based on the evaluation of the insert operation.

Federation automatically performs bulk inserts when the insert operation includes a source table or includes multiple values for the INSERT operator in the access plan. A bulk insert is typically chosen if you insert data to a Netezza nickname by using the following insert operations:

- Insert data to Netezza from a result set of sub-query as shown in the following example:

  ```
  INSERT INTO n1 SELECT c1, c2 FROM local_t1;
  ```

- Insert more than one value into Netezza as shown in the following example:

  ```
  INSERT INTO n1 VALUES (1, 'a'), (2, 'b');
  ```

To enable bulk insert, set the ODBC server option ENABLE_BULK_INSERT to Y.

Chapter 34. Security for federated servers

The federated server supports Secure Socket Layer (SSL) for data encryption, as well as HTTP and SOCKS proxies for specific data sources.

Encryption

Encryption provides a level of security that goes beyond what is provided by just a name and password. The Internet standard for encryption between endpoints is Secure Socket Layer (SSL) and Transport Layer Security (TLS). SSL uses signed certificates to ensure secure communications. A *certificate* is a digital document that provides assurance of a user's or server's identify. A certificate is signed by a certificate authority, such as VeriSign, or can be self-signed. Each communications partner determines whether or not to accept a particular certificate as being authentic.

Each communications partner has a certificate store, or keystore. The keystore holds the certificates that the partner accepts from others, as well as the certificates that represent itself. Each endpoint determines whether or not it will send a certificate when opening communications and whether or not it will accept communications from a partner who does not supply a certificate.

For the purposes of wrappers and functions, these SSL features are relevant:

- Server-side identification of a certificate to send
- Server-side verification of a client certificate
- Client-side verification of a server certificate
- Client-side identification of a certificate to send
- Both client and server support type, location, and access to a local keystore

The interaction between SSL and a proxy depends on the type of proxy. In general, SSL communications are tunneled, or relayed, through a proxy. The proxy session is established in clear text.

The IBM Global Security Kit (GSKit) provides encryption services for wrappers and user-defined functions.

Proxies

In an effort to foil numerous kinds of Internet attacks, many companies implement a firewall. A *firewall* is a network configuration that is usually comprised of both hardware and software and that is often located at a communication boundary, for example between a corporate intranet and the Internet. The firewall acts as a gatekeeper to regulate traffic across the boundary. In most cases, the firewall prevents unwanted communications from crossing the boundary; however, occasionally the firewall can block legitimate communications

To ensure that all legitimate communications pass through the firewall, you implement a proxy. A *proxy* is a server program that is authorized to communicate through the firewall. Then when a user program needs to connect to a remote server, the user program makes a request to the proxy, which connects to the remote server. After making the connection, the proxy controls the traffic between the user program and the remote server. This process ensures that the suer

program can cross the firewall and enforces security: the remote server knows only the address of the proxy, not the address of the user program.

SOCKS and HTTP proxy servers control the traffic between user programs and remote servers. A SOCKS proxy operates at the transport layer and relays arbitrary transport messages (TCP or UDP) between two addresses. The federated server supports both SOCKS4 and SOCKS5. SOCKS4, which supports IPv4, does not support user authentication. Therefore, anyone can communicate through a SOCKS4 proxy without having to provide user credentials. SOCKS5, which supports IPv6, supports several modes of user authentication. The Internet Engineering Task Force (IETF) approved SOCKS5 as a standard. For more information, go to www.ietf.org and see RFC1928, RFC1929, and RFC1961.

To use a SOCKS proxy, the transport layer must be configured, in advance, to use a proxy. After opening a connection to the proxy, the transport layer requests that the proxy open a connection to the remote server. If configured to require authentication, the SOCKS proxy might ask the program to provide an ID and password before opening the connection to the remote server.

An HTTP proxy works with the HTTP protocol, which is an application-layer protocol. After opening a TCP/IP connection to the proxy, the user program sends a request that includes the name of the remote server. Then the user program continues to submit requests through the proxy server. This process changes slightly if the remote server requires authentication. If the remote server requires authentication, the proxy server sends a response message that includes the proxy-authenticate challenge header. This header includes information about the kind of authentication to be performed. The user program then resubmits the request and includes a proxy-authorization header, which contains the response to the authentication challenge.

Chapter 35. Public user mappings

To ensure the security of your federated server there are restrictions for using public user mappings to access your data sources.

For all data sources in InfoSphere Federation Server Version 9.7 Fix Pack 2 and later, you can create public user mappings to map all local database users to a single authorization ID and password. To ensure the security of your federated server while using a public user mapping, the following restrictions exist:

Public user mappings and non-public user mappings cannot coexist for the same server definition

For a given server definition, only one type of user mapping is allowed, either public or non-public user mappings. If one type of user mapping exists for the server, you receive error SQL1515N if you attempt to create a user mapping of the other type. For example, if a public user mapping has been defined for a server, you cannot create non-public user mappings for the same server. The reverse is also true. If non-public user mappings exist for a server definition, you receive error SQL1515N if you attempt to create a public user mapping for that server.

Restricting the coexistence of public and non-public user mappings also restricts the use of outbound federated trusted connections. If the FED_PROXY_USER server option exists for a server definition, you receive error SQL1515N if you attempt to create a public user mapping. You also receive error SQL1516N if you attempt to run the ALTER SERVER statement to add the FED_PROXY_USER server option when a public user mapping exists.

Public user mappings are not retrieved from an external user mapping repository

You are not restricted from attempting to create public user mappings on external user mapping repositories but the federated server does not search or retrieve them to ensure your security.

Chapter 36. Federated trusted contexts and trusted connections

Enhance system performance and minimize or completely reduce the use and maintenance of user mappings.

A *trusted context* is a DB2 database object that defines a trust relationship between a client and a data source, for example, between an application server and a federated server or between a federated server and a remote database server. To define a trusted relationship, the trusted context specifies *trust attributes*. There are three types of trust attributes:

- The system authorization ID that makes the initial database connection request
- The IP address or domain name from which the connection is made
- The encryption setting for data communications between the database server and the database client

A *trusted connection* is established when all of the attributes of a connection request match the trust attributes that are specified in any trusted context object that is defined on the server. After an explicit trusted connection is established, users can be switched on the same physical connection, with or without authentication. In addition, users can be granted roles that specify privileges that are for use only within the trusted connection.

This example creates a trusted context object for BOSS:

```
CREATE TRUSTED CONTEXT MYCTX
 BASED UPON CONNECTION USING SYSTEM AUTHID BOSS
 ATTRIBUTES (ADDRESS '9.26.111.111')
 WITH USE FOR MARY WITH AUTHENTICATION ROLE MANAGER,
  PUBLIC WITHOUT AUTHENTICATION
 DEFAULT ROLE AUDITOR
 ENABLE
```

In this example, only BOSS can initiate a trusted connection from IP address 9.26.111.111. Mary can reuse the connection, but she must first authenticate. Then she gains the additional role of MANAGER, which specifies the privileges that Mary can use within this trusted connection. Other users, specified as PUBLIC, can reuse the connection, and they do not need to authenticate. These other users gain the additional role of AUDITOR, which specifies privileges that they can use within this trusted connection. These additional privileges are available to users only while they are active users of the trusted connection.

A trusted connection is either explicit or implicit. The type of connection determines whether the connection can be reused and whether users can gain additional roles.

An *implicit trusted connection* is established when a trusted connection is not explicitly requested but the connection attributes match the trust attributes of a trusted context object on the server. After an implicit trusted connection is established, only the originator of the trusted connection can inherit roles that are not otherwise available to him. An implicit trusted connection cannot be reused by other users.

An *explicit trusted connection* is established when an application uses an API to request a trusted connection. If the connection attributes match the trust attributes of a trusted context, a trusted connection is established. Otherwise, a regular connection is established. After an explicit trusted connection is established, other users can reuse the connection; and both the connection originator and the connection users can inherit additional roles that are not otherwise available to them.

Benefits of federated trusted connections

In a multi-tier application model, federated trusted connections reuse a single physical connection to propagate each user's real identity through the tiers to the database server.

To understand the benefits of federated trusted connections, consider the problems inherent in a typical multi-tier application model. A multi-tier application model consists enterprise users (Tier 1) who interact with an application that runs on an application server (Tier 2), which routes all database access through the federated server (Tier 3), which manages the communications with various database servers (Tier 4). In this model, the application server authenticates users and manages interactions with the federated server. The federated server translates user requests into data-source specific formats, establishes connections with the remote data sources, and sends requests to them.

This model uses the application server ID and password to create a connection to the database server; the federated server merely passes the ID and password from the application server to the database server. The database server uses the database privileges that are associated with this ID to authorize and audit all transactions that the application server performs, including all transactions that the application server performs on the behalf of enterprise users.

Using the application server ID presents these problems:

- The real identity of the user who performs a transaction is unknown because the application server performs all transactions.
- Users cannot be held accountable for transactions because they cannot be audited.
- The principle of least privilege is violated because the application server ID requires the superset of all privileges that all users require.
- Data is vulnerable if the application server ID is compromised.

Federated trusted connections address these problems and provide these benefits that can enhance the security and performance of the system:

User identity is known
: Because users are switched on the connection, you know the actual identity of the users who access the database.

Users are held accountable
: The audit logs for the federated database and for the remote data source database identify the transactions that the application server performs for its own purposes and the transactions that individual users perform. Therefore, you can hold specific users accountable for specific transactions.

Privileges are limited
: When you create a trusted context, you can grant a default database role to all users and can grant specific roles to specific users. Only trusted

database connections that match the definition of that trusted context can take advantage of the privileges that are associated with that role.

Data is less vulnerable
In a system that uses federated trusted connections, the application server ID does not require the superset of all privileges that all users need. Therefore, if the application server ID is ever compromised, data is less vulnerable than it is when the ID has the superset of all of the privileges that all users require.

Administrative maintenance is minimized
The need to create and maintain user mappings is significantly reduced.

Performance is improved
After an explicit trusted connection is established, the federated server can switch the current user ID on the connection to a different user ID, with or without requiring that the user authenticate. The reuse of the same physical connection for different users can improve performance.

Types of federated trusted connections

Federated trusted connections are either end-to-end trusted connections or outbound trusted connections. Which type of connection is made depends on how you configure the system and whether or not the inbound connection request is trusted.

A typical federated configuration is multi-tier; that is, it includes an application server, a federated server, and a remote data source server. In this configuration, the federated server receives inbound connection requests from the application server and sends outbound connection requests to the data source server.

End-to-end federated trusted connections

An end-to-end federated trusted connection provides connection reuse and identity assertion for both inbound and outbound connections. For example, when an inbound connection to the federated server is trusted, either explicitly or implicitly, the federated server automatically requests an outbound trusted connection. If the data source provides identity-assertion functionality, a trusted outbound connection is made; and the user's identity is propagated to the remote data source. The inbound and outbound connections are reused each time another user requests to reuse the trusted connection, and that new user's identity is propagated through the system.

For data sources that do not provide identity-assertion functionality, the federated server provides end-to-end identity assertion but does not provide connection reuse. In those cases, each time the user is switched, the federated server closes the outbound connection for the previous user and creates a new connection for the new user. By doing so, the user's identity is propagated through the system, even though the outbound connection is not reused.

Outbound federated trusted connections

An outbound federated trusted connection leverages the ability of data sources to reuse trusted connections without authentication to eliminate the need to store data-source passwords in user mappings. If the inbound connection is trusted, the federated server automatically requests a trusted outbound connection that is

reused without authentication. For non-trusted inbound connections, outbound federated trusted connections provide the ability to reuse connections without requiring users to authenticate.

In this configuration, you use the FED_PROXY_USER option in the server definition to specify the authorization ID that initially establishes the outbound connection. The authorization ID that you specify must have a user mapping that includes both the REMOTE_AUTHID and REMOTE_PASSWORD options.

Depending on how you configure the trusted context on the remote data source server, you can reduce the number of user mappings in the database catalog to as few as one. For example, if you allow PUBLIC to connect without authentication, then only the federated proxy user requires a user mapping. However, if the federated trusted context on the remote data source specifies that certain users must authenticate or if users do not use the same user ID on the federated server and on the remote data source, you must create or alter the user's user mapping to use only the REMOTE_AUTHID option, to use only the REMOTE_PASSWORD option, or to use both the REMOTE_AUTHID option and the REMOTE_PASSWORD option; and you must set the USE_TRUSTED_CONTEXT option to 'Y'.

Because there are significant additional configuration and maintenance tasks affiliated with outbound federated trusted connection, design the federated system to implement end-to-end trusted connections wherever possible. Then use outbound federated trusted connections only where absolutely necessary.

APIs for federated trusted connections

The APIs that you use to request and to reuse a trusted connection depend on the type of application.

To request a federated trusted connection, the application must specify these APIs.

CLI/ODBC applications
Use SQLSetConnectAttr with the attribute SQL_ATTR_USE_TRUSTED_CONTEXT to indicate whether the client is requesting a trusted connection. Then issue an SQLConnect.

XA CLI/ODBC applications
In an xa_open string request, set the TCTX attribute to true to request a trusted connection.

Java applications
Use getDB2TrustedPooledConnection or getDB2TrustedXAConnection to request a trusted connection.

To reuse the connection for another user, with or without requiring authentication, the application must specify these APIs:

CLI/ODBC applications
Use SQLSetConnectAttr to set the SQL_ATTR_TRUSTED_CONTEXT_USERID and SQL_ATTR_TRUSTED_CONTEXT_PASSWORD to specify the user ID and password of the user to whom the connection is being switched.

XA CLI/ODBC applications
Use SQLSetConnectAttr to set the SQL_ATTR_TRUSTED_CONTEXT_USERID and

SQL_ATTR_TRUSTED_CONTEXT_PASSWORD to specify the user ID and password of the user to whom the connection is being switched.

Java applications
Use getDB2Connection and reuseDB2Connection.

Scenarios for implementing federated trusted connections

Each federated system is unique; therefore, there are no step–by-step instructions for implementing federated trusted contexts. Use these scenarios to gain a complete understanding of federated trusted connections; then plan and implement your own solution.

Each scenario uses the same multi-tier federated system that includes users, an application that runs on an application server, a federated server, and a remote DB2 database server. The scenarios illustrate configuring remote sources and federated servers to use federated trusted connections.

The first scenario, which does not require user mappings, is the simplest to implement and maintain. In fact, this is the recommended scenario to use when you implement trusted connections in a new system.

The second scenario illustrates implementing trusted connections in a system that includes user mappings. The third scenario illustrates leveraging federated trusted connections to eliminate user mappings. Keep in mind that these scenarios are more complex and require more effort to configure and maintain than the first scenario does.

Scenario: End-to-end federated trusted connections, without user mappings

User mappings require a significant amount of administrative maintenance. This scenario illustrates how to configure federated trusted connections so that the federated system does not require user mappings.

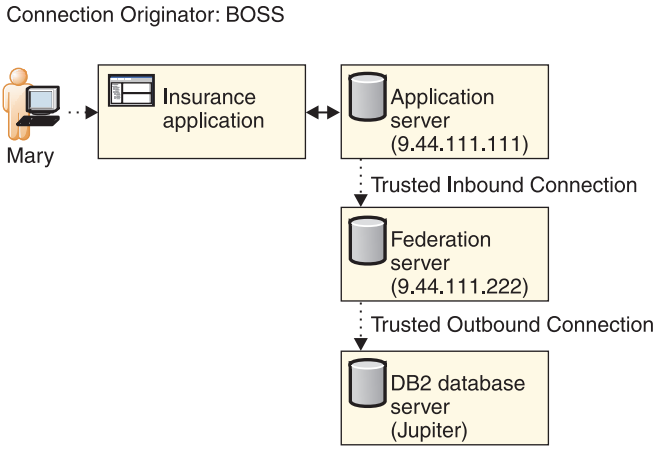
User ID and password requirements

To set up federated trusted contexts without using user mappings, the user IDs and passwords must meet these requirements:

- The user ID and password for the connection originator must be available to the federated server, and the connection originator's user ID and password must be the same on the federated server and on the data source. The credentials are identified in a CONNECT statement that the connection originator issues or in an API call that the application makes. If the CONNECT statement is used, an implicit trusted connection is created, and the connection cannot be reused. If the application makes an API call and explicitly requests a trusted connection, an explicit trusted connection is created, and the connection can be reused.
- If the remote trusted context allows connection reuse without authentication, connection re-users must have the same user ID on the federated server and on the remote data source.
- If the remote trusted context allows connection reuse with authentication, connection re-users must have the same user name and password on the federated server and on the remote data source.

The scenario

The following figure illustrates a typical multi-tier federated system that is configured to use end-to-end federated trusted connections. Although this scenario includes an application server, any database client can establish a trusted connection.



This scenario has two users, neither of whom require a user mapping:

- BOSS has the same user ID and password on the federated server and on the data source. Therefore, BOSS does not require a user mapping.
- Mary uses the same user ID on the federated server and on the data source, and the federated context allows her to connection without authenticating. Therefore, she does not require a user mapping.

The scenario has three servers:

- The application server, which hosts the insurance application and has the IP address 9.44.111.111.
- The federation server, which has the IP address 9.44.111.222.
- The remote DB2 database server, which is cataloged on the federated server as JUPITER.

The following steps describe the configuration of this scenario.

Note: In the commands, object names that are variable display in italics. When you implement trusted contexts, specify variable names that apply to your specific system configuration.

1. On the remote DB2 database server, create this trusted context object:

   ```
   CREATE TRUSTED CONTEXT MY_DB2_TCX
   BASED UPON CONNECTION USING
   SYSTEM AUTHID BOSS
   ATTRIBUTES (ADDRESS '9.44.111.222')
   WITH USE FOR PUBLIC WITHOUT AUTHENTICATION
   ENABLE
   ```

   This trusted context specifies that BOSS is the trusted connection originator and that the connection request must come from IP address 9.44.111.222, which identifies the federation server. The trusted context specifies WITH USE FOR PUBLIC WITHOUT AUTHENTICATION. Consequently, after the trusted connection is established, any valid user of the remote data source can reuse the connection by providing only a user ID.
2. On the federation server, create this trusted context object:

```
CREATE TRUSTED CONTEXT MY_WFS_TCX
BASED UPON CONNECTION USING
SYSTEM AUTHID BOSS
ATTRIBUTES (ADDRESS '9.44.111.111')
WITH USE FOR PUBLIC WITHOUT AUTHENTICATION
ENABLE
```

This trusted context specifies that BOSS is the trusted connection originator and that the request for the connection must come from the IP address 9.44.111.111, which identifies the application server. After the trusted connection is established, any valid user of the federated server can reuse the connection by providing only a user ID.

3. On the federation server, create this server definition:

```
CREATE SERVER JUPITER TYPE db2/udb
 VERSION 9.5 WRAPPER drda...
 OPTIONS(DBNAME 'remotedb', ...);
```

This server definition contains the information that the federated server requires to connect to the remote DB2 database named remotedb.

The scenario, step-by-step

This is a brief step-by-step description of how trusted connections are made and how users are switched in this scenario. The scenario code includes comments that describe how the application accomplishes these tasks.

1. The application server requests a trusted inbound connection for BOSS.
2. BOSS performs a task, and the federated server establishes an explicit outbound trusted connection for BOSS. User ID BOSS is propagated from the application server through the federation server to the DB2 database server, where the actions that BOSS performs can be audited.
3. Mary logs into the insurance application which is on her laptop. The application server switches the inbound connection to the federated server from BOSS to Mary.
4. Mary performs a task within the application.
5. The federation server switches the outbound connection from BOSS to Mary, and her ID is propagated through the federation server to the DB2 server, where the actions that Mary performs can be audited.

Sample code for end-to-end federated trusted connection scenarios

This sample code illustrates how to use APIs in an application that uses end-to-end federated trusted connections.

An application must use APIs to request an explicit trusted inbound connection and to switch the user on the connection. This sample code illustrates the parts of the application that perform these tasks. For both end-to-end trusted context scenarios, the application is the same.

This excerpt from an application uses the command line interface APIs. APIs are also available for applications that use Java or XA ODBC/CLI.

```
//Set the trusted connection attribute.
SQLSetConnectAttr(h1, SQL_ATTR_USE_TRUSTED_CONTEXT, SQL_TRUE, SQL_IS_INTEGER);

//Establish a trusted inbound connection for BOSS.
SQLConnect(h1, "testdb", SQL_NTS, "BOSS", SQL_NTS,"*****", SQL_NTS);

//Establish a trusted outbound connection for BOSS.
```

```
Perform some work under the user ID BOSS.
SQLExecDirect(hstmt, (unsigned char*)"INSERT INTO PATENTS_NN VALUES...", SQL_NTS);
...
//Commit the work.
SQLEndTran(SQL_HANDLE_DBC, h1, SQL_COMMIT);

//At the transaction boundary, switch the inbound user ID on the trusted
connection to Mary.
SQLSetConnectAttr(h1, SQL_ATTR_TRUSTED_CONTEXT_USERID,"Mary", SQL_IS_POINTER);

//Switch the outbound user ID on the trusted connection to Mary. Perform some
work under the user ID Mary.
SQLExecDirect(*hstmt, (unsigned char*)"INSERT INTO PATENTS_NN VALUES...", SQL_NTS)

...
//Commit the work.
SQLEndTran(SQL_HANDLE_DBC, h1, SQL_COMMIT);

//Disconnect from the database.
SQLDisconnect(h1);
```

Scenario: End-to-end federated trusted connections, with user mappings

User mappings are required when users do not have the same user ID and password on the federated server and on the remote data source. For this scenario, you create trusted user mappings, as well as configure trusted contexts.

User mapping requirements

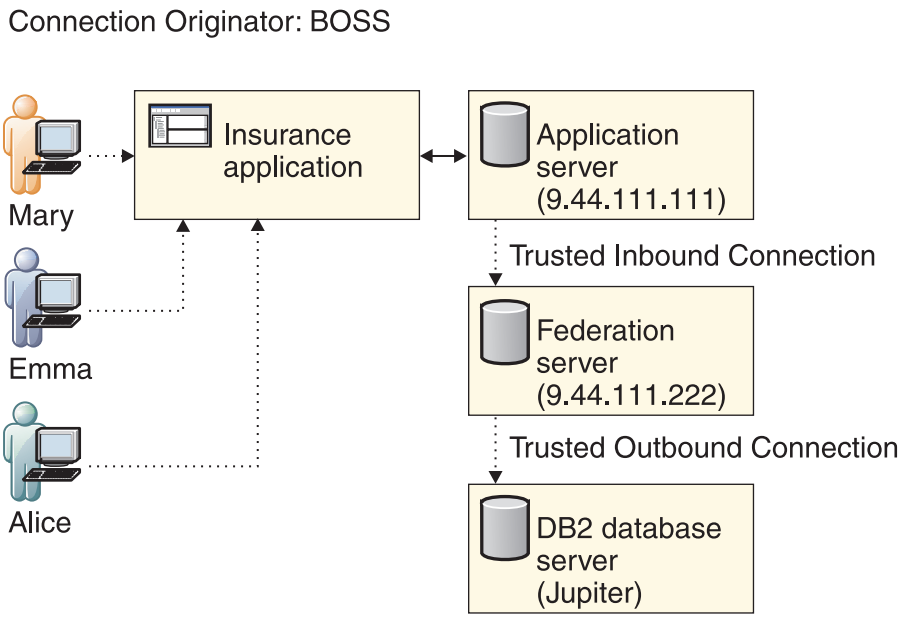
The federated server receives inbound connection requests and makes outbound connection requests to a remote data source. When users have the same user ID and password on the federated server and on the remote DB2 database server, user mappings are not required. However, when user credentials do not match, a user mapping is required. The user mapping maps the user's user ID on the federated server to the user's user ID, and to the user's password if one is specified, on the remote database server.

In a federated system that uses end-to-end trusted contexts, users whose name and password on the federated server and on the remote DB2 database server do not match require *trusted user mapping*. A trusted user mapping specifies that the user has permission to use a trusted context. To create a trusted user mapping or alter an existing user mapping, you set the USE_TRUSTED_CONTEXT user mapping option to 'Y'.

Who can create and alter a trusted user mapping is carefully controlled. Only a user who has SECADM authority can create or drop a trusted user mapping or alter an existing user mapping to add, set, or drop the USE_TRUSTED_CONTEXT user mapping option. A user who has a trusted user mapping can alter only the REMOTE_PASSWORD option of his own user mapping.

The scenario

Here is a simple graphic that illustrates a typical multi-tier federated system that uses user mappings. Although this scenario includes an application server, any database client can establish a trusted connection.



This scenario has four users:

- BOSS, who is the connection originator, has the same user ID and password on the federated server and on the remote DB2 database server. Therefore, BOSS does not have a user mapping .
- Mary does not have a user mapping. She uses the same user ID on both the federation server and on the DB2 database server. Because the trusted context specifies that PUBLIC can reuse the connection without authenticating, Mary's password is not required.
- Alice has a trusted user mapping that specifies the REMOTE_AUTHID option. Alice has a different user ID on the federated server and on the remote DB2 database server. Because the trusted context specifies that anyone can reuse the connection without authenticating, Alice's password is not required.
- On the federated server, Emma uses the user ID EMMA. This user ID maps to the user ID EGREENE and the password MYPASS on the DB2 database server. Because the trusted context specifies that Emma must authenticate, Emma has a trusted user mapping that specifies both the REMOTE_AUTHID option and the REMOTE_PASSWORD option.

This scenario has three servers:

- The application server, which hosts the insurance application and has the IP address 9.44.111.111.
- The federation server, which has the IP address 9.44.111.222.
- The remote DB2 database server, which is cataloged on the federated server as JUPITER.

To configure this scenario, the SECADM completes these steps:

1. On the remote DB2 database server, create the trusted context object:

```
CREATE TRUSTED CONTEXT MY_DB2_TCX
BASED UPON CONNECTION USING
SYSTEM AUTHID BOSS
ATTRIBUTES (ADDRESS '9.44.111.222')
WITH USE FOR EMMA WITH AUTHENTICATION,
PUBLIC WITHOUT AUTHENTICATION
ENABLE
```

This trusted context specifies that BOSS is the trusted connection originator and that the request for the connection must come from the federated server that has the IP address 9.44.111.222. After the trusted connection is established, any valid database user can reuse the connection by providing only a user ID.

2. On the federated server, create the trusted context object:

```
CREATE TRUSTED CONTEXT MY_WFS_TCX
BASED UPON CONNECTION USING
SYSTEM AUTHID BOSS
ATTRIBUTES (ADDRESS '9.44.111.111')
WITH USE FOR EMMA WITH AUTHENTICATION,
PUBLIC WITHOUT AUTHENTICATION
ENABLE
```

   This trusted context specifies that BOSS is the trusted connection originator and that the request for the connection must come from IP address 9.44.111.111, which identifies the application server. After the trusted connection is established, Emma can reuse the connection, but she must authenticate. Any valid database user must provide only a user ID to reuse the connection.

3. On the federation server, create this server definition:

```
CREATE SERVER JUPITER TYPE db2/udb
VERSION 9.5 WRAPPER drda
OPTIONS(DBNAME 'remotedb', ...);
```

   This server definition contains the information that the federated server requires to connect to the remote DB2 database named remotedb.

4. On the federation server, create this trusted user mapping for Alice:

```
CREATE MAPPING FOR USER ALICE
SERVER JUPITER
OPTIONS
(REMOTE_AUTHID 'AJACKSON', USE_TRUSTED_CONTEXT 'Y');
```

   This user mapping specifies that a trusted connection can be reused by user ALICE who maps to user ID AJACKSON and on the remote DB2 database server.

5. On the federation server, create this trusted user mapping for Emma:

```
CREATE MAPPING FOR USER EMMA
SERVER JUPITER
OPTIONS
(REMOTE_AUTHID 'EGREENE', REMOTE_PASSWORD 'MYPASS', USE_TRUSTED_CONTEXT 'Y');
```

   This user mapping specifies that a trusted connection can be reused by user EMMA who maps to user ID EGREENE and password MYPASS on the remote DB2 database server.

The scenario, step-by-step

1. The application server requests a trusted inbound connection for BOSS.
2. BOSS performs a task, and the user ID BOSS is propagated through the federated server to the DB2 database server, where the actions that BOSS performs can be audited.
3. Emma logs into the insurance application, which is hosted on the application server. The application server requests that the federated inbound connection be switched from BOSS to Emma, after authenticating Emma.
4. Emma performs a task within the application.
5. The federated server switches the federated outbound connection from BOSS to Emma, and her user mapped user ID and password are propagated through the federated server to the DB2 server, where the actions that EGREENE (Emma's remote user ID) performs can be audited.

6. Alice logs into the insurance application. The application server requests that the federated inbound connection be switched from Emma to Alice, without authenticating Alice.
7. Alice performs a task within the application.
8. The federated server switches the federated outbound connection from Emma to AJACKSON (Alice's mapped user ID), and her user ID is propagated through the federated server to the DB2 database server, where the actions that AJACKSON perform can be audited.
9. Mary logs into the insurance application. Mary does not require authentication; therefore, the application server switches the federated inbound trusted connection to Mary, without providing a password.
10. Mary performs a task within the application.
11. The federated server switches the federated outbound connection from AJACKSON to Mary, and Mary's user ID is propagated through the federated server to the DB2 database server, where the actions that Mary performs can be audited.

Scenario: Federated outbound trusted connections

A federated outbound trusted connection establishes a trusted environment between the federated server and a remote DB2 database server. Because this connection does not require authentication, you can eliminate the need to store and maintain user passwords.

For some configurations, the federation server must accommodate inbound connections that are not trusted. A connection is not trusted when either of the following statements are true:

- The attributes of the inbound connection request do not match the attributes of any trusted context object on the federated server.
- The federation server does not specify a trusted context for the server from which the connection request comes. All users get non-trusted inbound connections.

Trusted context, server definition, and user mapping requirements

When you create the trusted context on the remote DB2 data source server, you must specify WITH USE FOR PUBLIC WITHOUT AUTHENTICATION. Then all users who use the same user ID on the federated server and on the DB2 database server can use the outbound trusted context without authenticating.

To create a federated outbound trusted connection, you specify the FED_PROXY_USER option in the server definition for the remote DB2 data source server. This option identifies the authorization ID of the user who originates the outbound trusted connection. In addition, you create a user mapping for the federated proxy user. This user mapping must specify both the REMOTE_AUTHID and REMOTE_PASSWORD options because the trusted context on data source requires that the connection originator authenticate.

Only a user who has SECADM authority can create and alter a server definition that has the FED_PROXY_USER option defined. In addition, the SET SERVER OPTION statement is not valid for the FED_PROXY_USER server option.

There might be situations when you want to configure different sets of users to connect through different proxy users. For example, you may want to configure

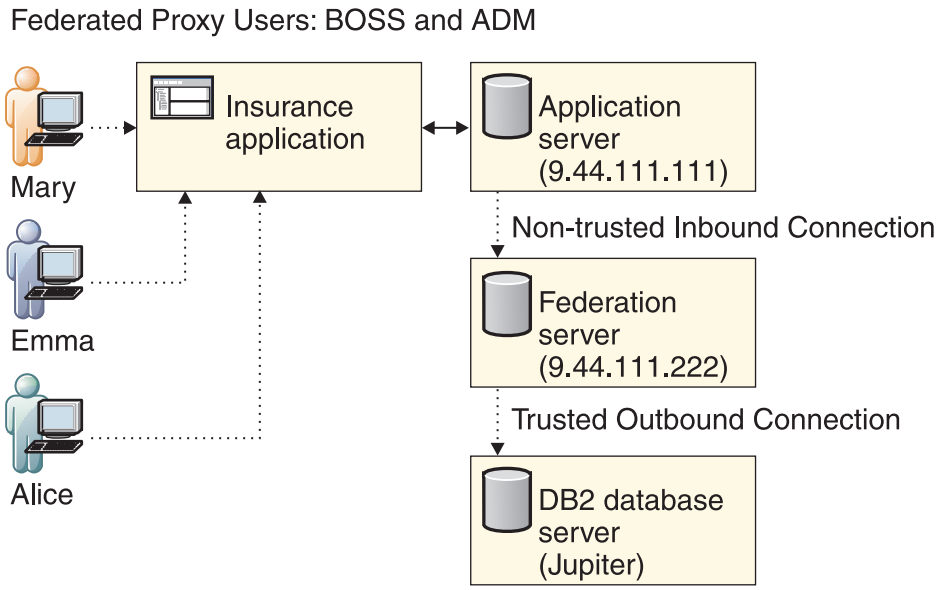
different roles for different users. To facilitate this, in each user mapping that does not use the default federated proxy user, you specify the FED_PROXY_USER option, and set the USE_TRUSTED_CONTEXT option to 'Y'. When the FED_PROXY_USER option is specified in both the server definition and the user mapping, the value in the user mapping overrides the value in the server definition.

Only a user who has SECADM authority can add, drop, or set the FED_PROXY_USER option and create or drop a user mapping that includes the FED_PROXY_USER option.

Important:
There are only two situations in which the user of a trusted outbound connection requires a user mapping: when the user uses a different user ID on the federated server and on the database server and when the user must connect through a federated proxy user who is not the default federated proxy user.

The scenario

The following figure illustrates a scenario that requires federated outbound trusted connections. Although this scenario includes an application server, any database client can establish a trusted connection.



This scenario has five users:

- BOSS, who is the default federated proxy user, has a user mapping.
- ADM, who is a federated proxy user, has a user mapping.
- Mary, who uses the insurance application, does not have a user mapping.
- Emma and Alice, each of whom use the insurance application, have user mappings.

This scenario has three servers:

- The application server, which hosts the insurance application and has the IP address 9.44.111.111. This scenario shows an application server, but any client can be used.
- The federation server, which has the IP address 9.44.111.222.

- The remote DB2 database server, which is cataloged on the federated server as JUPITER.

These steps describe the configuration:

Note: In the commands, object names that are variable display in italics. When you implement trusted contexts, specify variable names that apply to your specific system configuration.

1. On the remote DB2 database server, create these trusted context objects:

   ```
   CREATE TRUSTED CONTEXT MY_DB2_TXC
   BASED UPON CONNECTION USING
   SYSTEM AUTHID BOSS
   ATTRIBUTES (ADDRESS '9.44.111.222')
   WITH USE FOR PUBLIC WITHOUT AUTHENTICATION
   ENABLE
   ```

   This trusted context specifies that BOSS is the connection originator and that the request for the outbound trusted connection must come from the federated server that has the IP address 9.44.111.222. After the trusted outbound connection is established, any user can reuse the connection without authenticating.

   ```
   CREATE TRUSTED CONTEXT MY_DB2_TXC_ALICE
   BASED UPON CONNECTION USING
   SYSTEM AUTHID ADM
   ATTRIBUTES (ADDRESS '9.44.111.222')
   WITH USE FOR PUBLIC WITHOUT AUTHENTICATION ROLE Manager
   ENABLE
   ```

   This trusted context specifies that ADM is the connection originator and that the request for the outbound trusted connection must come from the federated server that has the IP address 9.44.111.222. After the trusted outbound connection is established, Alice can reuse the connection without authenticating, and she gains the role of Manager within the scope of the trusted connection.

2. On the federation server, create this server definition:

   ```
   CREATE SERVER JUPITER TYPE db2/udb
   VERSION 9.5 WRAPPER drda...
   OPTIONS(DBNAME 'remotedb',FED_PROXY_USER 'BOSS');
   ```

   This server definition contains the information that the federated server requires to connect to the remote DB2 database named remotedb. The definition specifies that if the inbound connection to the federated server is not trusted, then BOSS, the default federated proxy user, establishes the outbound trusted connection.

3. On the federation server, create user mappings for the federated proxy users:

   ```
   CREATE USER MAPPING FOR BOSS SERVER JUPITER
   OPTIONS(REMOTE_AUTHID 'BOSS',REMOTE_PASSWORD 'MYPASS');
   CREATE USER MAPPING FOR ADM SERVER JUPITER
   OPTIONS(REMOTE_AUTHID 'ADM', REMOTE_PASSWORD 'PWD');
   ```

4. On the federated server, create these trusted user mappings for Emma and Alice.

   ```
   CREATE USER MAPPING FOR EMMA SERVER JUPITER
   OPTIONS(REMOTE_ AUTHID 'EGREENE', USE_TRUSTED_CONTEXT 'Y')
   CREATE USER MAPPING FOR ALICE SERVER JUPITER
   OPTIONS (USE_TRUSTED_CONTEXT 'Y',FED_PROXY_USER 'ADM');
   ```

Both user mappings have the USE_TRUSTED_CONTEXT option is set to 'Y', so that the users can use the trusted context. In addition, Emma and Alice require these additional options:

- Emma requires a user mapping because she does not use the same user ID on the federated server and on the DB2 database server. Therefore, her user mapping specifies the REMOTE_AUTHID option.
- Alice requires a user mapping because the trusted context specifies that she uses ADM as the federated proxy user. Therefore, her user mapping specifies the FED_PROXY_USER option.

In this scenario, Mary does not require a user mapping. The user mappings for Emma and Alice do not store the remote password; therefore, those user mappings do not require constant updates to keep the remote password up-to-date.

The scenario, step-by-step

These steps describe how trusted outbound connections work in this scenario.

1. The application creates a non-trusted inbound connection to the federated server:

   ```
   CONNECT TO FEDSVR USER MARY USING '****'
   ```

2. The first federated request that accesses the server JUPITER creates an outbound connection to JUPITER:

   ```
   SELECT * FROM JUPITER_NN01
   CONNECT RESET
   ```

3. Because JUPITER specifies FED_PROXY_USER=BOSS, and Mary's user mapping does not specify a federated proxy user, the outbound connection is created for BOSS and then immediately switched to the current inbound user, who, in this case, is Mary.
4. Mary's federated request is completed and recorded in the audit log under the name MARY. Then the connection is reset:
5. Create a non-trusted inbound connection to the federated server for Emma:

   ```
   CONNECT TO FEDSVR USER EMMA USING '****'
   ```

6. The first federated request in the current connection that accesses JUPITER creates a federated outbound connection to JUPITER.

   ```
   SELECT * FROM JUPITER_NN01
   CONNECT RESET
   ```

7. The outbound connection is created using BOSS and then is immediately switched to the current inbound user, Emma, whose ID is mapped to EGREENE.
8. Emma's federated request is completed and recorded in the audit log under the name EGREENE. Then the connection is reset:
9. Create a non-trusted inbound connection to the federated server for Alice:

   ```
   CONNECT TO FEDSVR USER ALICE USING '****'
   ```

10. The first federated request in the current connection that accesses JUPITER creates a federated outbound connection to JUPITER. Because Alice's trusted user mapping specifies that she use federated proxy user ADM rather than the default federated proxy user BOSS, the outbound connection is created using ADM and then is immediately switched to Alice, who is the current inbound user.

    ```
    SELECT * FROM JUPITER_NN01
    CONNECT RESET
    ```

11. Alice's federated request is completed and can be recorded in the audit log. Then the connection is reset:

User mappings and federated trusted connections

These tables describe how user mappings, with and without remote IDs and remote passwords, are used in end-to-end federated trusted connections and in outbound federated trusted connections.

These tables describe the possible user mappings for BOSS, the connection originator, and for Mary, the connection re-user. In the tables, the term *non-trusted connection* refers to a regular connection that is not trusted on the inbound connection and not trusted on the outbound connection.

Some cells of the table use the term *if available* to describe a password. A password is available to the federated server if both a user ID and a password are explicitly specified as part of the connection. If you use API connect calls to connect to the federated server, then the password is explicitly passed in; for example, in CLI/ODBC, the password is specified through the SQL_ATTR_TRUSTED_CONTEXT_PASSWORD connection attribute. If you use the CONNECT statement to connect to the federated server, then use this syntax to pass in the password explicitly:

```
CONNECT TO database name USER user ID USING password
```

Mappings for connection originators and federated proxy users

Table 22. User mappings for BOSS, the connection originator

User mapping for BOSS	Non-trusted connection	End-to-end federated trusted connection (Where BOSS establishes the trusted inbound connection)	Outbound federated trusted connection (Where the server or user mapping option FED_PROXY_USER='BOSS' is set)
No user mapping	Pass BOSS's federated user ID and federated password (if available) to the remote data source.	Pass BOSS's federated user ID and federated password (if available) to the remote data source.	ERROR SQL1101N
User mapping that specifies only a remote user ID	Pass BOSS's remote user ID and federated password (if available) to the remote data source.	Pass BOSS's remote user ID and federated password (if available) to the remote data source.	ERROR SQL1101N
User mapping that specifies a remote user ID and a remote password	Pass BOSS's remote user ID and remote password to the remote data source.	Pass BOSS's remote user ID and remote password to the remote data source.	Pass BOSS's remote user ID and remote password to the remote data source.

Mappings for connection re-users

Table 23. User mappings for Mary, the connection re-user

User mapping for Mary	End-to-end federated trusted connection (Where BOSS establishes the trusted inbound connection, and the connection switches to Mary)	Outbound federated trusted connection (Where the server or user mapping option FED_PROXY_USER='BOSS' is set)
No user mapping	Pass Mary's federated user ID and federated password (if available) to the remote data source.	Pass Mary's federated user ID to the remote data source.
Non-trusted user mapping that specifies only a remote user ID	Pass Mary's federated user ID and federated password (if available) to the remote data source.	Pass Mary's federated user ID to the remote data source.
Non-trusted user mapping that specifies a remote user ID and a remote password	Pass Mary's federated user ID and federated password (if available) to the remote data source.	Pass Mary's federated user ID to the remote data source.
Trusted user mapping that specifies only a remote user ID	Pass Mary's remote user ID to the remote data source.	Pass Mary's remote user ID to the remote data source.
Trusted user mapping that specifies a remote user ID and a remote password	Pass Mary's remote user ID and remote password to the remote data source.	Pass Mary's remote user ID and remote password to the remote data source.

Chapter 37. Label-based access control (LBAC) and federated systems

Ensure that only users who have the appropriate authority can see the data in a table.

With label-based access control, you can apply security policies to the rows and columns of a table. Each security policy specifies the credentials that are granted to each user ID and session ID. For example, the table Prices has columns Wholesale, Retail, and Sale. If the user Alice is entitled to access columns Retail and Sale, then the query SELECT RETAIL, SALE FROM PRICES succeeds. But the query SELECT * WHOLESALE fails.

When you create a nickname on an object, the federated server automatically detects whether the data source uses label-based access control. If label-based access control is being used, the nickname is not cached. For nicknames that were created before label-based access control was available, use the ALTER NICKNAME statement to allow or disallow caching. For example, if you created a nickname on a data source object before federated support for label-based access control was available, you can alter the nickname to disallow caching.

Each security policy has a unique label that is stored in the Label column of the table. A database administrator can hide the column that contains the labels to prevent users from knowing that the column exists. Nicknames that have hidden label columns are not cached.

Chapter 38. External user mapping repositories

Store user mappings in an external repository for multiple federated servers to share, and reduce the administrative maintenance of managing user mappings that are stored on each federated server.

From a maintenance perspective, storing user mappings in an external repository is better than storing them in the catalog. Remote passwords expire on a regular basis. If you store user mappings in the catalog, when a remote password expires, it must be updated at the remote data source and in the catalog. With an external repository, when a remote password expires, you update it only once, in the external repository.

To use an external repository for user mappings, you must create a user mapping plug-in. The plug-in must use security settings that match the security settings that the external repository uses. Use Secure Sockets Layer (SSL) to secure communications between the federated server and the external repository. Then when you create the configuration file for the user mapping plug-in, specify that the plug-in uses SSL. Also, restrict access to the source code of the plug-in so that the information remains secure.

If auditing is turned on, each time the federated server uses the user mapping plug-in, a VALIDATE audit record is created. To capture VALIDATE records, configure the db2audit facility.

Note: To ensure security, public user mappings are not retrieved from external user mapping repositories.

These sample plug-ins and complete instructions for creating, testing, and deploying your own custom plug-in are provided:

User mapping plug-in (C programming language)

The C plug-in consists of five functions that provide an interface for retrieving user mappings from an external repository.

The sequence of the function calls is as follows

1. FSUMPluginInit
2. FSUMconnect
3. FSUMfetchUM
4. FSUMdisconnect
5. FSUMPluginTerm

Initialize – FSUMPluginInit

Immediately after the federated server loads the plug-in library, the server calls the FSUMPluginInit function, which initializes the plug-in and passes the pointers for the other functions to the federated server. These functions are global and must be externally resolvable. If the plug-in is written in C++, these functions must be declared with extern "C". The federated server passes in the pointers to a set of utility functions, which the plug-in can obtain, as necessary.

The plug-in is loaded into a threaded db2fmp process. All applications that use the plug-in share the same plug-in library, which must be thread-safe. Each application uses one thread in the db2fmp process. After the application successfully retrieves a user mapping and cleans up, the federated server returns the thread to the thread pool for future use. Because the federated server can serve multiple applications simultaneously, multiple threads that share the same plug-in library can be activated at the same time. Each thread has a separate connection handle to the user mapping repository. The FSUMPluginInit API also provides a way for each thread to handle global plug-in resources, for example to increase the reference count to the plug-in.

Connect to repository – FSUMconnect

The federated server calls the FSUMconnect function to connect to the user mapping repository. The plug-in must include a descriptor that stores all of the information that is required to make the connection to the repository. For example, the information might include a handle to an opened file or a connection handle to an LDAP server. Depending on how you implement security for the external repository, the information might also include a user ID and a password. If credentials are required, you must set up the way that the credentials are managed. For example, if you put credentials in a configuration file, when the plug-in tries to connect to the external repository, the plug-in reads the credentials from that file.

Retrieve user mapping – FSUMfetchUM

The plug-in calls the FSUMfetchUM function to retrieve the user mapping from the external repository and calls the FSUMaddUMOption utility function to send the remote ID and remote password to the federated server. In the repository, each user mapping is identified by the federated server instance name, the database name, the remote server name, and the local authorization ID. In addition, each user mapping must include the REMOTE_AUTHID and REMOTE_PASSWORD options. If a remote password is encrypted, the plug-in must decrypt it before sending it to the federated server.

Disconnect from repository – FSUMdisconnect

The federated server calls the FSUMdisconnect function to disconnect from the user mapping repository. To disconnect, this function disassociates the thread with the user mapping repository. For example, this function might close an opened file or close a connection to an LDAP server.

Release global resources – FSUMPluginTerm

As the last step, the federated server calls the FSUMPluginTerm function to release any global resources that the FSUMPluginInit function allocated. To terminate, this function disassociates the thread with the plug-in.

Supported platforms for the user mapping plug-in (C programming language)

Before you build the plug-in, confirm that you are using a supported platform.

The following table lists the supported platforms for the user mapping plug-in.

Platform	File name
AIX, 64 bit	plugin_file_name.a
Linux AMD 64, Power PC, 64, Power PC 390	plugin_file_name.so
Microsoft Windows, 32-bit and 64-bit	plugin_file_name.dll

Restrictions for developing a user mapping plug-in (C programming language)

When you develop user mapping plugs-in in C, keep these restrictions in mind.

C-linkage
The plug-in library must be linked with C-linkage. Header files that provide the prototypes, data structures needed to implement the plug-in, and error code definitions are provided for C/C++ only. Functions that will be resolved at load time must be declared with extern "C" if the plug-in library is compiled as C++.

.NET common language runtime is not supported
The .NET common language runtime (CLR) is not supported for compiling and linking the source code for the plug-in library.

Signal handlers
The plug-in library must not install signal handlers or change the signal mask because doing so interferes with the reporting and recovering from errors. The plug-in library must never raise C++ exceptions.

Thread-safe
The plug-in library must be thread-safe and re-entrant. Only the plug-in initialization is not required to be re-entrant. The plug-in initialization function can potentially be called multiple times from different threads, in which case, the plug-in cleans up all used resources and reinitializes itself.

Overriding standard C library and operating system calls
The plug-in library must not override standard C library and operating system calls.

32-bit and 64-bit applications
A 32-bit federated server must use a 32-bit plug-in. A 64-bit federated server must use a 64-bit plug-in. In a hybrid instance, where the client is 32-bit and the server is 64-bit, the plug-in must be 64-bit.

Text strings
Input strings are not guaranteed to be null-terminated, and output strings are not required to be null-terminated. Instead, integer lengths are given for all input strings, and pointers to integers are given for lengths to be returned.

Header file (fsumplugin.h) for the user mapping plug-in (C programming language)

The header file, fsumplugin.h, contains data structures, functions, and error code.

The user mapping plug-in must include this header file, which is in the `sqllib/include` directory.

```
/* Definition of user mapping option names. */
#define FSUM_REMOTE_AUTHID_OPTION "REMOTE_AUTHID"
#define FSUM_REMOTE_PASSWORD_OPTION "REMOTE_PASSWORD"
```

```
/* Definition of option value types. */
#define FSUM_OPTION_VALUE_BINARY_TYPE 1
#define FSUM_OPTION_VALUE_STRING_TYPE 2

/* Data structure to describe an user option. */

typedef struct _FSUMOption
{
   const char* optionName;
   size_t optionNameLen;
   char*  optionValue;
   size_t optionValueLen;
   size_t optionValueType;
   struct _FSUMOption* nextOption;
} FSUMOption;

/* Data structure to describe a user mapping entry. A user mapping entry might
   have multiple user mapping options, such as REMOTE_AUTHID and REMOTE_PASSWORD.
   These options are placed in a linked-list.  This data structure holds the pointer
   to the first option in the list. */

typedef struct _FSUMEntry
{
   const char* fsInstanceName;
   size_t      fsInstanceNameLen;
   const char* fsDatabaseName;
   size_t      fsDatabaseNameLen;
   const char* fsServerName;
   size_t      fsServerNameLen;
   const char* fsAuthID;
   size_t      fsAuthIDLen;
   FSUMOption* firstOption;
} FSUMEntry;

/* The functions to implement in addition to the FSUMPluginInit function,
   which is not in the FSUMPluginAPIs structure. */

typedef struct _FSUMPluginAPIs
{
   size_t version;
   SQL_API_RC (SQL_API_FN * FSUMconnect)
 (void** a_FSUMRepository,const char* a_cfgFilePath);

   SQL_API_RC (SQL_API_FN * FSUMfetchUM)
 (void* a_FSUMRepository,FSUMEntry* a_entry);

   SQL_API_RC (SQL_API_FN * FSUMdisconnect)
 (void* a_FSUMRepository);

   SQL_API_RC (SQL_API_FN * FSUMPluginTerm) ();
} FSUMPluginAPIs;

/* The federated server provides these utilities as callback functions
to the plug-in. */

typedef SQL_API_RC (SQL_API_FN FSUMallocateFP)
 (size_t a_blkSize, void** a_pblkPtr);

typedef void(SQL_API_FN FSUMdeallocateFP)
 (void* a_blkPtr);

typedef SQL_API_RC (SQL_API_FN FSUMloadFP)
 (const char* a_libName,void** a_lib);

typedef SQL_API_RC (SQL_API_FN FSUMgetFunctionFP)
 (const char* a_functionName,void* a_lib,void** a_pFuncAddress);
```

```
typedef SQL_API_RC (SQL_API_FN FSUMunloadFP)
 (void* a_lib);

typedef SQL_API_RC (SQL_API_FN FSUMlogErrorMsgFP)
(sqlint32 a_level, const char* a_msg, size_t a_length);

typedef SQL_API_RC (SQL_API_FN FSUMaddUMOptionFP)
 (FSUMEntry *a_entry,
 const char* optionName,
 size_t optionNameLen,
 const char* optionValue,
 size_t optionValueLen);

/* Structure to hold the utility functions that the federated server provides. */

typedef struct _FSUMPluginUtilities
{
   FSUMallocateFP      *allocate;
   FSUMdeallocateFP    *deallocate;
   FSUMloadFP          *load;
   FSUMgetFunctionFP   *getFunction;
   FSUMunloadFP        *unload;
   FSUMlogErrorMsgFP   *logErrorMsg;
   FSUMaddUMOptionFP   *addUMOption;
} FSUMPluginUtilities;

/* User mapping plug-in entry point type. */
typedef SQL_API_RC (SQL_API_FN *FSUMPluginInitType)
 (sqlint32, FSUMPluginAPIs*, FSUMPluginUtilities*);

/* User mapping plug-in C interface entry point type. */
typedef SQL_API_RC (*fsum_plugin_hook_type)
 (const char*, FSUMPluginInitType*, FSUMPluginUtilities*);

/* Definition of return codes for utility functions */
#define FSUM_PLUGIN_UTIL_OK 0
#define FSUM_PLUGIN_UTIL_FAILED -1

/* Definition for extern C. */
#define FSUM_PLUGIN_EXT_C extern "C"

/* Error severities that the logErrorMsg function uses.*/
#define FSUM_LOG_NONE      0 /* No logging */
#define FSUM_LOG_CRITICAL  1 /* Severe error encountered */
#define FSUM_LOG_ERROR     2 /* Error encountered */
#define FSUM_LOG_WARNING   3 /* Warning */
#define FSUM_LOG_INFO      4 /* Informational */


/* Error codes that the functions return.*/
#define FSUM_PLUGIN_OK                0
#define FSUM_INITIALIZE_ERROR         1
#define FSUM_PLUGIN_VERSION_ERROR     2
#define FSUM_CONNECTION_ERROR         3
#define FSUM_LOOKUP_ERROR             4
#define FSUM_DECRYPTION_ERROR         5
#define FSUM_DISCONNECT_ERROR         6
#define FSUM_INVALID_PARAMETER_ERROR  7
#define FSUM_UNAUTHORIZED_CALLER      8
#define FSUM_AUTHENTICATION_ERROR     9
#define FSUM_TERMINATION_ERROR        10

/* Maximum length of a name.*/
#define FSUM_MAX_NAME_LEN             128

/* Maximum length of a file path. */
#define FSUM_MAX_PATH_LEN             256
```

```
/* Maximum length of an option value. */
#define FSUM_MAX_OPTION_VALUE_LEN     (2048+1)

/* Maximum length of an error message. */
#define FSUM_MAX_ERROR_MSG_SIZE       2048
```

FSUMPluginInit function (C programming language)

FSUMPluginInit is the initialization function for the user mapping plug-in.

This function performs these tasks:

- Passes the pointers for the four other required functions to the federated server.
- Obtains utility functions that the federated server passes.
- Sets up global resources at the plug-in level.

Syntax

```
SQL_API_RC SQL_API_FN FSUMPluginInit
 (sqlint32 a_version,
 FSUMPluginAPIs* a_pluginAPIs,
 FSUMPluginUtilities* a_pluginUtils);
```

Inputs

sqlint32 a_version
The version number of the user mapping plug-in interface.

FSUMPluginUtilities *a_pluginUtils
The pointer to the structure that contains all the utility functions. The plug-in obtains a pointer to each utility function, as needed.

Outputs

FSUMPluginAPIs *a_pluginAPIs
The plug-in uses this structure to pass the function pointers for FSUMconnect, FSUMfetchUM, FSUMdisconnect, and FSUMPluginTerm to the federated server.

FSUMconnect function (C programming language)

To connect to the external user mapping repository, the federated server calls the FSUMconnect function.

Syntax

```
SQL_API_RC SQL_API_FN FSUMconnect
(void** a_FSUMRepository,
const char* a_cfgFilePath)
```

Inputs

const char* a_cfgFilePath
The full path to the plug-in library. If the configuration file is placed in the same directory as the plug-in library, the plug-in can use this path information to locate the configuration file.

Outputs

void a_FSUMRepository**
The pointer to the connection descriptor is cast to void before it is sent to federated server. In subsequent API function calls, the federated server

passes the pointer to the plug-in, and the plug-in must cast the pointer back the real structure of the connection descriptor.

FSUMfetchUM function (C programming language)

The federated server calls the FSUMfetchUM function to retrieve user mapping options from an external repository.

Each user mapping entry is identified by the federated instance name (fsInstanceName), database name (fsDatabaseName), remote server name (fsServerName), and the local user's authorization ID (fsAuthID). The user mapping can also include the REMOTE_AUTHID and REMOTE_PASSWORD options, which the plug-in retrieves.

Syntax

```
SQL_API_RC SQL_API_FN FSUMfetchUM (void* a_FSUMRepository,
FSUMEntry* a_entry);
```

Inputs

void* a_FSUMRepository
The connection handle to the external repository. This handle must be cast to the real structure that is defined in the plug-in.

FSUMEntry* a_entry
This parameter is both an input parameter and an output parameter.

As an input, this parameter passes the information that is required to identify the user mapping entry in the user mapping repository. This information includes fsInstanceName, fsDatabaseName, fsServerName, and fsAuthID.

As an output, the user mapping options that are retrieved from the user mapping repository are added to this parameter. The plug-in must use the FSUMaddUMOption utility function to add an option to the user mapping entry, a_entry. If the user mapping includes the REMOTE_PASSWORD option and if the password is encrypted, the plug-in must decrypt the password before calling the FSUMaddUMOption. The federated server is not responsible for freeing the storage for the strings that are passed to the FSUMaddUMoption function.

FSUMdisconnect function (C programming language)

To disconnect from the external user mapping repository, the federated server calls this function.

Syntax

```
SQL_API_RC SQL_API_FN FSUMdisconnect
(void* a_FSUMRepository);
```

Inputs

void* a_FSUMRepository
The connection handle to the external repository. This handle must be cast to the real structure that is defined in the plug-in.

FSUMPluginTerm function (C programming language)

The user mapping plug-in calls this function to release global resources that the FSUMPluginInit function allocates.

This function has no parameters.

```
SQL_API_RC SQL_API_FN FSUMPluginTerm ()
```

Developing a user mapping plug-in (C programming language)

The plug-in that you develop must connect to the external repository, retrieve user mappings, and decrypt remote passwords. The repository that you use determines how you code the plug-in.

About this task

Important: Be aware that as you develop and use the plug-in, you send sensitive user IDs and passwords between multiple sources. To protect this information, restrict access to the plug-in source code, and configure the DB2 audit facility to capture a VALIDATE record in the diagnostic log file each time that the federated server uses the plug-in. The diagnostic log file is useful not only for tracking usage but also for troubleshooting any problems that occur.

To develop a user mapping plug-in, complete these tasks:

Declaring external functions for the user mapping plug-in (C programming language)

The plug-in implements the functions. When the FSUMPluginInit function is called, it passes the function pointers to the federated server.

The initialization function, FSUMPluginInit, is the only function that must have this exact function name. For each of the other functions, you can use any valid C function name. The functions are global and must be resolved externally. If you write the plug-in in C++, you must use FSUM_PLUGIN_EXT_C to declare the functions.

The sample plug-in implements the following APIs:

- FSUMconnect API in the myConnect function
- FSUMfetchUM in the myFetchUM function
- FSUMdisconnect in the myDisconnect function
- FSUMPluginTerm in the myPluginTerm function

In the sample, when the FSUMPluginInit function is called, the function pointers for myConnect, myFetchUM, myDisconnect, and myPluginTerm are passed to the federated server.

The following code is in the `fsumplugin_file.c` file.

```
SQL_API_RC SQL_API_FN FSUMPluginInit
(sqlint23 version,
FSUMPluginAPIs *pluginAPIs,
FSUMPluginUtilities* pluginUtils)

SQL_API_RC SQL_API_FN myConnect
(void** FSUMRepository)

SQL_API_RC SQL_API_FN myFetchUM
(void* FSUMRepository,
FSUMEntry* entry)

SQL_API_RC SQL_API_FN myDisconnect
(void* FSUMRepository)

SQL_API_RC SQL_API_FN myPluginTerm ()
```

Declaring utility functions for the user mapping plug-in (C programming language)

You must declare four required utility options for the user mapping plug-in.

FSUMlogErrorMSg, FSUMaddUMOption, FSUMallocate, and FSUMdeallocate are required utility functions for the user mapping plug-in. The optional utility functions can be useful when you develop a user mapping plug-in. You obtain the pointers of the utility functions from the FSUMPluginUtilities structure in the FSUMPluginInit function.

Required utility functions

```
FSUMlogErrorMsgFP *FSUMlogErrorMsg=NULL;
FSUMaddUMOptionFP *FSUMaddUMOption=NULL;
FSUMallocateFP *FSUMallocate=NULL;
FSUMdeallocateFP *FSUMdeallocate=NULL;
```

Optional utility functions

```
FSUMgetFunctionFP *FSUMgetFunction=NULL;
FSUMloadFP *FSUMload=NULL;
FSUMunloadFP *FSUMunload=NULL;
```

Setting function pointers for the user-mapping plug-in (C programming language)

Pass the pointers to the required functions to the federated server, and obtain the pointers to the utility functions from the federated server.

```
/* The plug-in initialization function. Do not change the name. */

SQL_API_RC SQL_API_FN FSUMPluginInit
 (sqlint32 a_version,
 FSUMPluginAPIs* a_pluginAPIs,
 FSUMPluginUtilities* a_pluginUtils)
{
   SQL_API_RC rc = FSUM_PLUGIN_OK ;

   /* Pass the function pointers to federated server */

   a_pluginAPIs->FSUMconnect = &myConnect
   a_pluginAPIs->FSUMfetchUM = &myFetchUM
   a_pluginAPIs->FSUMdisconnect = &myDisconnect
   a_pluginAPIs->FSUMPluginTerm = &myPluginTerm

   /* Get the pointers of the required utility functions */

   FSUMallocate = a_pluginUtils->allocate;
   FSUMdeallocate = a_pluginUtils->deallocate;
   FSUMlogErrorMsg = a_pluginUtils->logErrorMsg;
   FSUMaddUMOption = a_pluginUtils->addUMOption;
   /*You can also get the pointers of the optional utility functions */

   /** If there is an error, do the following:
     1. Optional --- Call FSUMlogErrorMsg to log the error
     2. Mandatory --- Return error code rc = FSUM_INITIALIZE_ERROR
   **/

   return rc;
}
```

Error handling in the user mapping plug-in (C programming language)

The user mapping plug-in uses the FSUMlogErrorMsg function to log all error and informational messages in the `db2diag.log` file.

If a function call is successful, the plug-in returns FSUM_PLUGIN_OK. If the call is not successful, the plug-in returns an error code. The following table describes the errors.

Table 24. Errors for the user mapping plug-in

Error number	Constant name	Error message	Functions that return the error	Error code
0	FSUM_PLUGIN_OK	The plug-in ran successfully.	• FSUMPluginInit • FSUMconnect • FSUMfetchUM • FSUMdisconnect • FSUMPluginTerm	SQL_SUCCESS (No error)
1	FSUM_INITIALIZE_ERROR	The plug-in failed to initialize.	• FUMPluginInit	SQL20349N , reason code 1
2	FSUM_PLUGIN_VERSION_ ERROR	The plug-in version is wrong.	• FUMPluginInit	SQL20349N, reason code 2
3	FSUM_CONNECTION_ ERROR	Unable to connect to the external repository.	• FSUMPluginInit	SQL20349N, reason code 3
4	FSUM_LOOKUP_ERROR	Lookup on the repository failed.	• FSUMfetchUM	SQL20349N, reason code 4
5	FSUM_DECRYPTION_ERROR	Decryption failed.	• FSUMfetchUM	SQL20349N, reason code 5
6	FSUM_DISCONNECT_ERROR	Unable to disconnect from the repository.	• FSUMdisconnect	SQL20349N, reason code 6
7	FSUM_INVALID_ PARAMETER_ERROR	Invalid parameter.	• FSUMfetchUM • FSUMdisconnect • FSUMPluginTerm	SQL20349N, reason code 7
8	FSUM_UNAUTHORIZED_ CALLER	The caller is not authorized to call the plug-in.	• FSUMPluginInit	SQL20349N, reason code 8
9	FSUM_AUTHENTICATION_ ERROR	Unable to authenticate with the repository.	• FSUMconnect	SQL20350N, no reason code
10	FSUM_TERMINATION_ ERROR	Failure to clean up a resource at the plug-in level.	• FSUMPluginTerm	SQL20349N, reason code 9
Not a number from 0 to 10		Unknown error occurred.	• FSUMPluginInit • FSUMconnect • FSUMfetchUM • FSUMdisconnect • FSUMPluginTerm	SQL20349N, reason code 10

The plug-in can also use the FSUMlogErrorMsg utility function to log messages to the `db2diag.log` file. To specify the severity of the messages, use FSUM_LOG_CRITICAL, FSUM_LOG_ERROR, FSUM_LOG_WARNING, or FSUM_LOG_INFO.

When you use the test program to test the plug-in, messages are printed directly to the screen.

Building the user mapping plug-in (C)

Use the bldplugin script to build the user mapping plug-in.

About this task

To build the user mapping plug-in:

Procedure

1. Examine the bldplugin script and update the path information to match your specific installation.
2. Issue this command to build the plug-in:

   ```
   bldplugin plugin_file_name
   ```

   For more information, see the bldplugin file.
3. Set up the external repository to include user mapping information and to enforce an encryption scheme. If you are building the sample user mapping plug-in, open the fsumplugin_file.txt file and create a new user mapping entry for your system. The sample plug-in uses a simple encryption scheme that reverses the bytes of the remote password.

Testing the user mapping plug-in (C programming language)

Use the test program to load and test the sample user mapping plug-in or your own custom user mapping plug-in before you deploy it to the federated server.

About this task

The setup program fsumsetup_file (fsumsetup_file.exe on Microsoft Windows) and the test program fsumlookup (fsumlookup.exe on Windows) are installed in the same directory where the sample plug-in files are installed.

The fsumsetup_file program generates a configuration file named fsumplugin_file.cfg. This file stores the full path to the text file that contains the user mapping entries. The fsumlookup program (fsumlookup.exe on Microsoft Windows) tests the plug-in.

Procedure

1. Run the fsumsetup_file program to set up the full path and file name of the file that stores the user mappings.
2. Run the fsumlookup program to test the plug-in:

   ```
   fsumlookup pluginName fsInstance fsDatabase fsRemoteServer fsAuthid
   ```

   where:

   - *pluginName* is the name of the plug-in.
   - *fsInstance* is the name of the federated server instance.
   - *fsDatabase* is the name of the federated database.
   - *fsRemoteServer* is the name of the remote data source server, as specified in the CREATE SERVER statement.
   - *fsAuthid* is the local user ID.

   If the plug-in is functioning properly, the fsumlookup program prints the parameters that identify the user mapping entry, as well as the option names and option values, to the screen. The following is an example of the output.

   ```
   fsumlookup fsumplugin_file.a FSinst1 FSdb remoteDB localID
   fsInstanceName: FSinst1
   fsDatabaseName: FSdb
   ```

```
fsServerName: remoteDB
fsAuthID: localID
optionName:REMOTE_PASSWORD
optionValue:p4ssw0rd
optionName:REMOTE_AUTHID
optionValue:remoteID
```

If an error occurs, the test program prints the error message to the screen.

Deploying the user mapping plug-in (C programming language)

You can build and compile a user mapping plug-in in any directory. Then to deploy the plug-in, copy it to the correct directory on the federated server.

Procedure

To deploy the plug-in on the federated server, copy the plug-in and the configuration file to the correct directory on the federated server:

- On UNIX and Linux, *inst_home*/sqllib/function, where *inst_home* is the instance home directory
- On Windows, *%DB2PATH%*\function, where *%DB2PATH%* is the directory where the DB2 database system is installed

Enabling access to the external user mapping repository

After you create a user mapping plug-in, you must set the DB2_UM_PLUGIN and DB2_UM_PLUGIN_LANG options to enable the federated server to access the user mappings that are stored in the external repository.

About this task

You can set the options at the wrapper or server level, but you must set both options at the same level. DB2_UM_PLUGIN specifies the name of the plug-in, and DB2_UM_PLUGIN_LANG specifies the language in which the plug-in is written. The two options are dependent on each other. Therefore, you must add DB2_UM_PLUGIN before you add DB2_UM_PLUGIN_LANG. If you add both options in the same statement, the order in which you specify them does not matter. To drop the options, you must drop DB2_UM_PLUGIN_LANG before you drop DB2_UM_PLUGIN.

For example, the following statements use the name of the sample plug-in to create wrapper w1, alter wrapper w2,, create server s1 and alter server s2 to use the user mapping plug-in:

```
CREATE WRAPPER w1 LIBRARY 'libdb2drda.a'
 OPTIONS
 (DB2_UM_PLUGIN 'fsumplugin_file.a', DB2_UM_PLUGIN_LANG 'C');

ALTER WRAPPER w2
 OPTIONS
 (ADD DB2_UM_PLUGIN 'fsumplugin_file.a', ADD DB2_UM_PLUGIN_LANG 'C');

CREATE SERVER s1 TYPE db2/cs VERSION 10 WRAPPER w2
 AUTHORIZATION remoteID PASSWORD p4ssw0rd
 OPTIONS
 (NODE 'node1',
 DBNAME 'db1',
 DB2_UM_PLUGIN 'fsumplugin_file.a',
 DB2_UM_PLUGIN_LANG 'C');

ALTER SERVER s2
 OPTIONS
 (ADD DB2_UM_PLUGIN 'fsumplugin_file.a', ADD DB2_UM_PLUGIN_LANG 'C');
```

Updating the user mapping plug-in (C programming language)

After you modify an existing user mapping plug-in, you must copy the updated version to the federated server.

About this task

To update the user mapping plug-in:

Procedure

1. Copy the updated plug-in to the sqllib/function directory.
2. Enter these commands to stop and then restart the federated server instance:

```
db2stop
db2start
```

Sample user mapping plug-in (C programming language)

Review the sample source code to learn how to develop a plug-in that provides a C interface to an external user mapping repository.

This sample user mapping plug-in uses a file as the external repository. The sample source files are automatically installed on your system. Where they are installed depends on the operating system that you use:

- On Windows, *%DB2PATH%*\samples\federated\umplugin\file, where *%DB2PATH%*is the directory where the DB2 database system is installed
- On Linux and UNIX, *inst_home*/sqllib/samples/federated/umplugin/file, where *inst_home* is the instance home directory

The sample includes these files:

fsumplugin_file.h
: The header file that contains data structures and function definitions for the sample plug-in.

fsumplugin_file.c
: The source code for the sample plug-in, which uses a file as the external repository.

fsumplugin_file.txt
: A file that contains sample user mapping entries that are in plain text.

fsumlookup
: A stand-alone program that tests the sample plug-in outside of the federated server environment. On Microsoft Windows, the file is `fsumlookup.exe`.

fsumsetup_file
: A stand-alone program that sets the full path and file name of the file that stores the user mappings. On Microsoft Windows, the file is `fsumsetup_file.exe`

bldplugin
: The build script that compiles and links the plug-in. On Microsoft Windows, the file is `bldplugin.bat`.

README A file that contains instructions for compiling, building, testing, and deploying the sample plug-in.

In the file `fsumplugin_file.txt`, each user mapping entry occupies one line, and the passwords are encrypted by simply reversing the bytes. This is a very simple

encryption method. If you modify the sample plug-in for use in a production environment, you must use an encryption method that meets the security requirements of the environment.

The following syntax shows the format for a sample user mapping entry.

Note: Although the code displays on multiple lines here, you must enter all of this code on one line in your application. Note that a semicolon separates the parts of the user mapping and that a colon separates the REMOTE_AUTHID and REMOTE_PASSWORD option names from their corresponding settings.

```
fsInstance;fsDatabase;fsRemoteServer;
fsAuthid;REMOTE_AUTHID:remote_authID;
REMOTE_PASSWORD:remote_password
```

For example:

```
DB2INST1;TESTDB;MSSQL2K;ALICE;REMOTE_AUTHID:TESTUSR1;REMOTE_PASSWORD:drowssap
```

User mapping plug-in (Java programming language)

The architecture of the Java plug-in includes two interface classes and three utility classes. Use these to develop a plug-in that retrieves user mappings from an external repository.

By default, user mappings for a data source are stored locally on each federated server. An LDAP server stores objects, such as user mapping entries, in a directory tree. These objects can have attributes, such as passwords. In addition, the LDAP server often stores additional information about users, for example their e-mail addresses and phone numbers.

The following figure illustrates the architecture of the sample the user mapping plug-in:

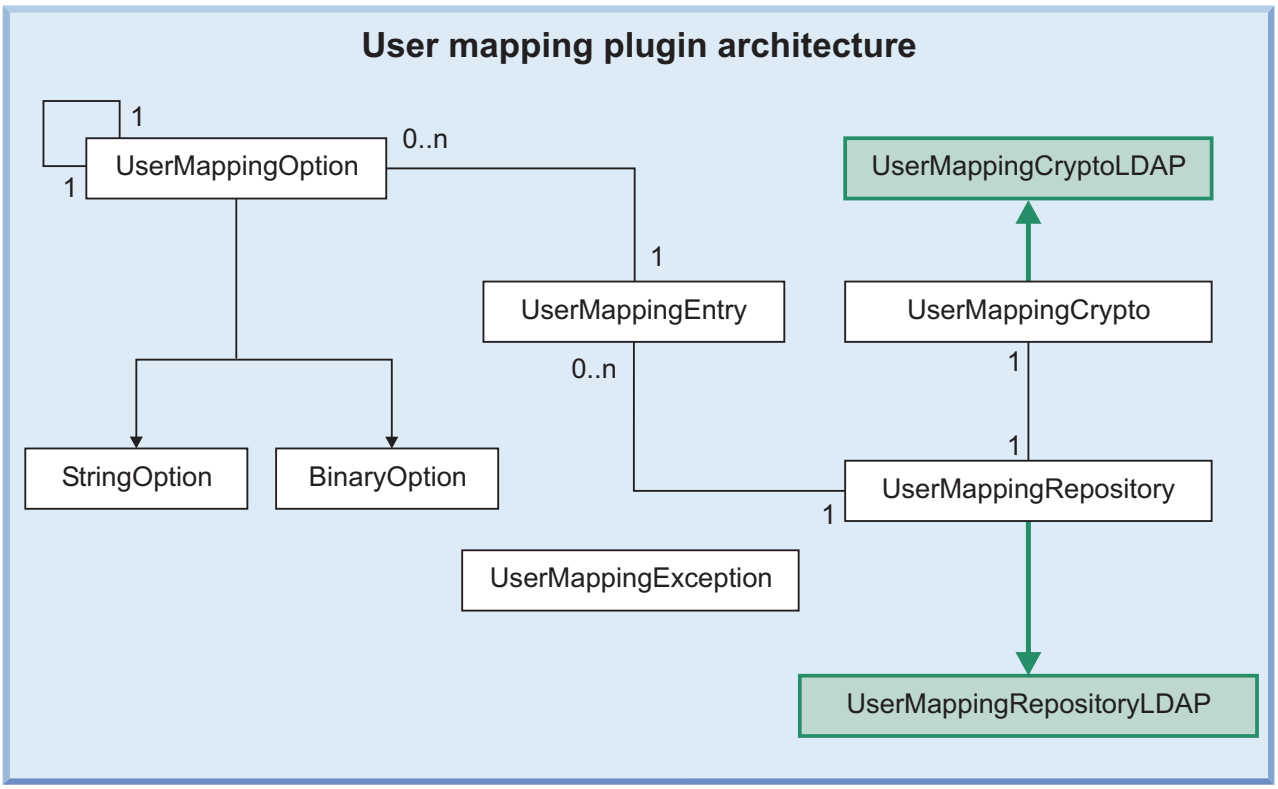


Figure 15. The architecture of the user mapping plug-in

UserMappingCrypto and UserMappingRepository are interface classes. To create a user mapping plug-in that retrieves user mappings from the external repository that your organization uses, you must extend these interface classes. UserMappingEntry, UserMappingOption, and UserMappingException are utility classes. You can use the utility classes without modification. Some functions and methods that are in the interface classes act as utility functions. For example, you can use the getChars() function and getBytes() function, which are in the UserMappingCrypto class, without modification.

In the figure, the *0..n* term means there can be zero or more of those objects. For example, a UserMappingEntry object can have multiple UserMappingOption objects, but there can be only be one UserMappingRepository object. The UserMappingCryptoLDAP class and UserMappingRepositoryLDAP class are extended from their parent classes.

Classes for the user mapping plug-in (Java programming language)

The architecture of the Java plug-in includes two interface classes and three utility classes. Use these to develop a plug-in that retrieves user mappings from an external repository.

UserMappingRepository class (Java programming language)

The UserMappingRepository class is an abstract class that does not have a constructor. To create your own user mapping plug-in, you must create a subclass of the UserMappingRepository class or modify the subclass in the sample Java plug-in.

The UserMappingRepository class contains the following public methods: getVersionNumber(), getCrypto(), connect(), disconnect(), fetchUM(), and lookupUM(). You must create your own subclass of the UserMappingRepository that contains these functions. In these functions, write the code that the plug-in uses to interact with the external repository.

You can view the implementation of these functions in a sample Java plug-in that retrieves user mappings from an LDAP server. The files are in the `sqllib/samples/federated/umplugin/ldap/` directory. The functions from this class are used in the UserMappingRepositoryLDAP.java and UserMappingLookupLDAP.java files.

Public methods

`int getVersionNumber()`
Returns the version number of the plug-in development kit that is used by the plug-in.

`UserMappingCrypto getCrypto()`
Returns the UserMappingCrypto object that is associated with this UserMappingRepository object.

`abstract void connect()`
You must implement your own method for connecting to your repository within this function.

`abstract void disconnect()`
You must implement your own method for disconnecting from your repository within this function.

`abstract voidfetchUM(``UserMappingEntry` **`um)`**
You must implement your own method for retrieving the user mapping from the repository within this function. The **um** parameter contains the detailed query information that is used to determine which user mapping to retrieve.

`UserMappingEntrylookupUM(UserMappingRepository repository,` `String` **`iiInstanceName,`** `String` **`iiDatabaseName,`** `String` **`iiRemoteServerName,`** `String` **`iiAuthid)`**
This function is primarily user to test the plug-in. The function uses the **iiInstanceName**, **iiDatabaseName**, **iiRemoteServerName**, and **iiAuthid** parameters as input to create and initialize the UserMappingEntry class. The function calls the `connect`, `fetchUM`, and `disconnect` methods.

UserMappingCrypto class (Java programming language)

If your external repository encrypts or encodes remote passwords, you must create your own subclass of the UserMappingCrypto class. The constructor of the subclass that you create is used to construct the cryptography object. The methods of the cryptography class are called by other classes when the user mapping passwords need to be encrypted, decrypted, encoded, or decoded.

The UserMappingCrypto class contains the following public methods: encrypt(), decrypt(), encode(), and decode(). In these functions, you must write your code for encrypting, decrypting, encoding, and decoding the remote password. The getBytes() and getChars() functions are utility functions that are inherited and can be used without modification. The encryption, decryption, encoding, and decoding methods that you code must match the encryption and encoding methods that the external repository uses to protect the stored passwords.

You can view the implementation of these functions in a sample Java plug-in that retrieves user mappings from an LDAP server. The files are located in the `sqllib/samples/federated/umplugin/ldap/` directory. The functions from this class are used in the UserMappingRepositoryLDAP.java and UserMappingSetupLDAP.java sample files.

Public methods

abstract byte[] encrypt(byte[] **plainValue)**
Implement the encryption algorithm that matches the encryption algorithm that the external repository uses.

abstract byte[] decrypt(byte[] **encryptedValue)**
Implement the decryption algorithm that reverses the encryption algorithm that the external repository uses and then returns the password.

abstract string encode(byte[] **bytes)**
Write or implement a function that encodes the **bytes** parameter into a string. This function encodes the encrypted value, which is in bytes, into a string.

abstract byte[] decode(String[] **string)**
Write or implement a function that decodes the **string** parameter into bytes. This function decodes the retrieved password, which is a string, into bytes so that the value can be decrypted.

byte[] getBytes(char[] **chars)**
This function is inherited and can be used without modification. The function transforms each character of a string into a byte.

char[] getChars(byte[] **bytes)**
This function is inherited and can be used without modification. The function transforms each byte into a character.

Protected attributes

SecretKey key
The secret key that is used to encrypt and decrypt the remote passwords.

Cipher cipher
The algorithm that the secret key uses to encrypt the password.

UserMappingEntry class (Java programming language)

The UserMappingEntry class is a utility class that creates and holds the user mapping options. The methods in the UserMappingEntry class are called by the fetchUM() and lookupUM() functions from the UserMappingRepository class.

Public methods

You can view the use of these functions in a sample Java plug-in that retrieves user mappings from an LDAP server. The files are in the `sqllib/samples/federated/umplugin/ldap/` directory. The functions from this class are used in the UserMappingRepositoryLDAP.java and UserMappingLookupLDAP.java files.

UserMappingEntry(UserMappingRepository **repository,** String **iiInstanceName,** String **iiDatabaseName,** String **iiRemoteServerName,** String **iiAuthID**)
This constructor is used to instantiate the UserMappingEntry object with the following input parameters:

- **iiInstance** - Instance name of the federated server
- **iiDatabase** - Database name on the federated server
- **iiRemoteServerName** - Remote server name for the data source

- **iiAuthid** - Local user ID that is associated with the user mapping

UserMappingRepository getRepository()
Returns the UserMappingRepository object that is associated with this UserMappingEntry.

String getiiInstanceName()
Returns the name of the instance.

String getiiDatabaseName()
Returns the name of the database.

String getiiRemoteServerName()
Returns the name of the remote server.

String getiiAuthID()
Returns the name of the local user ID that is associated with the user mapping.

UserMappingOption getFirstOption()
Returns the first UserMappingOption object that belongs to this UserMappingEntry object.

void addOption(UserMappingOption **newOption)**
Adds a new UserMappingOption object to this UserMappingEntry object.

UserMappingOption class (Java programming language)

The UserMappingOption class is a utility class that contains the functions for providing the federated server with the remote user ID and the remote password.

Public methods

You can view the use of these functions in a sample Java plug-in that retrieves user mappings from an LDAP server. The files are in the `sqllib/samples/federated/umplugin/ldap/` directory. The functions from this class are used in the UserMappingRepositoryLDAP.java and UserMappingLookupLDAP.java files.

UserMappingEntry getEntry()
Returns the UserMappingEntry object that is associated with this UserMappingOption object.

String getName()
Returns the name of the option.

void setName()
Sets the name of the option.

UserMappingOption getNextOption()
Returns the next option.

void setNextOption(UserMappingOption **nextOption)**
Sets the next option.

abstract object getValue()
Returns the value of the option. This function must return either a string value or binary value. See the StringOption and BinaryOption methods that are listed below.

StringOption class

The StringOption class is extended from the UserMappingOptions class and contains the following public methods: getValue() and setValue().

Public methods

object getValue()
Returns the value of the string option when the option is a string.

void setValue(String value)
Sets the value of the string option.

BinaryOption class

The BinaryOption class is extended from the UserMappingOption class and contains the following public methods: getValue() and void setValue().

Public methods

object getValue()
Returns the value of the binary option.

void setValue(byte[] value)
Sets the value of the binary option.

UserMappingException class (Java programming language)

The Java user-mapping plug-in uses the UserMappingException class, which is a subclass of the java.lang.Exception class, to report errors.

Public methods

You can view the use of these functions in a sample Java plug-in that retrieves user mappings from an LDAP server. The files are in the `sqllib/samples/federated/umplugin/ldap/` directory. The functions from this class are used in each of the sample Java files.

UserMappingException(`int` **errorNumber)**
The constructor that is used for instantiating the UserMappingException object that is used for reporting errors. The **errorNumber** parameter that is sent to the UserMappingException object defines the type of error that is reported.

int getErrorNumber()
Returns the error number of the exception.

The UserMappingRepositoryLDAP.java file contains this function for catching and reporting errors for testing an LDAP plugin.

String getErrorMessage()
Returns the error message of the exception.

The UserMappingRepositoryLDAP.java file contains a method for catching and reporting errors for testing an LDAP plugin.

Error messages

The following table lists error numbers, constant names, and error messages.

Table 25. Error numbers and messages

Error number	Constant name	Error message
1	INITIALIZE_ERROR	The plug-in failed to initialize.
2	CONNECTION_ERROR	Unable to connect to the repository.

Table 25. Error numbers and messages (continued)

Error number	Constant name	Error message
3	AUTHENTICATION_ERROR	Unable to authenticate with the repository.
4	LOOKUP_ERROR	Lookup on the repository failed.
5	DECRYPTION_ERROR	Decryption failed.
6	DISCONNECT_ERROR	Unable to disconnect from the repository.
7	INVALID_PARAMETER_ERROR	Invalid parameter.
8	UNAUTHORIZED_CALLER	The caller is not authorized to call the plug-in.

Sample user mapping plug-in (Java programming language)

The sample Java plug-in retrieves user mapping entries from an LDAP server. To use the plug-in in your environment, modify the sample plug-in to match the settings that your LDAP server uses.

The files for the plug-in are in the `sqllib/samples/federated/umplugin/ldap/` directory. The following table describes each file. Before you modify the files, copy them to an empty working directory. Then work with the copies.

Table 26. Description of the files in the sample user mapping plug-in (Java)

File name	Description
`README.txt`	This file contains a condensed version of the instructions and documentation for testing and using the sample plug-in.
`UserMappingCryptoLDAP.java`	This Java class contains the code that implements the security measures for encrypting, decrypting, encoding, and decoding the user mappings that are retrieved from the LDAP server. You must modify this file so that it works with your LDAP server.
`UserMappingSetupLDAP.java`	This Java class creates the configuration file that stores LDAP connection information and other configuration parameters, including: IP address or host name, SSL or non-SSL, user ID, and password.
`UserMappingRepositoryLDAP.java`	This Java class contains the code for connecting, disconnecting, and fetching user mappings from the LDAP server. The code for this file uses the schema that is defined in the `schema.ldif` file. If you change the schema, you must also change a section in this file.
`UserMappingLookupLDAP.java`	This Java class contains the code to perform an LDAP lookup test. Use this file to test your plug-in outside of the federated server. Then test the plug-in on the federated server.
`schema.ldif`	This file is loaded into the LDAP server to define the schema. The LDIF file contains objects and attributes that are added to the LDAP server.

Table 26. Description of the files in the sample user mapping plug-in (Java) (continued)

File name	Description
`entry.ldif`	The `entry.ldif` file adds user entries to the LDAP server. The user entries use the objects and attributes from the `schema.ldif` file to store the user mappings.

Developing a user mapping plug-in (Java programming language)

Use the sample code as a starting point to develop a Java plug-in that retrieves user mappings from an external repository. The sample code retrieves mappings from an LDAP repository, but you can modify the code to access any external repository.

Before you begin

Verify the following:

- The Java Development Kit (JDK) version 1.4 or later is installed.
- The `db2umplugin.jar` file. This Java Archive (JAR) file is installed as part of the DB2 server installation or the DB2 client installation.
- The sample user mapping plug-in files are installed. These files, which are installed as part of the DB2 client installation, are in the `sqllib/samples/federated/umplugin/ldap/` directory.
- The **`java_heap_sz`** parameter is set to 2048.
-

About this task

The plug-in that you develop must be able to connect to the external repository, retrieve user mappings, and decrypt remote passwords. The repository that you use determines how you code the plug-in. For example, if you use an LDAP repository that stores encrypted passwords, the plug-in must contain the encryption schema and the secret key that is required to decode the passwords.

Be aware that as you develop and use the plug-in, you send sensitive user IDs and passwords between multiple sources. To protect this information, restrict access to the plug-in source code, and configure the db2audit facility to capture a VALIDATE record in the diagnostic log file, `db2diag.log`, each time that the federated server uses the plug-in. The diagnostic log file is useful not only for tracking usage but also for troubleshooting any problems that occur.

To develop a plug-in, complete these tasks:

Modifying the sample user mapping plug-in files (Java programming language)

The sample Java plug-in retrieves user mappings from an LDAP server. You can modify the plug-in to retrieve user mappings from other types of external repositories. Use the sample plug-in as the starting point for developing a customized plug-in that works with your specific external repository.

Each sample file accomplishes a different task in the process of retrieving user mappings. You modify many of the functions and classes in the sample code to work with your specific external repository. The following table lists the important functions.

Table 27. Functions and classes to modify

File name	Element to modify	Class to reference

`UserMappingCryptoLDAP.java`	`UserMappingCryptoLDAP()`
	`encrypt()`
	`decrypt()`
	`getKey()`
	`decode()`
	`encode()`
		UserMappingCrypto class
		UserMappingException class

`UserMappingRepositoryLDAP.java`	`class UserMappingRepositoryLDAP(String configFilePath)`
	`connect()`
	`disconnect()`
	`fetchUM()`
		UserMappingRepository class
		UserMappingEntry class
		UserMappingOption class
		UserMappingException class

`UserMappingSetupLDAP.java`	Modify this file to create a configuration file that stores the values that your plug-in requires for connecting to the external repository and retrieving user mappings from it. If you manually create a configuration file, ensure that the passwords that you store in the file are encrypted. The configuration file name must match the name of the repository class, for example, UserMappingRepository*XXXX* class and `UserMappingRepository`*`XXXX`*`.cfg` file.	

Modifying the UserMappingCryptoLDAP sample file (Java programming language)

To implement the security methods that the LDAP server uses, modify functions that encrypt, decrypt, encode, and decode remote passwords.

About this task

Because encryption methods must be secret and unique, this task specifies the sections and functions to modify. You customize the code to implement the security methods that your LDAP server uses.

To modify the security functions in the UserMappingCryptoLDAP file:

Procedure

1. Open the `UserMappingCryptoLDAP.java` file with a text editor.
2. Below the IBM copyright and legal disclaimer, import the packages that your code will reference. The sample plug-in uses the javax.crypto and javax.crypto.spec Java packages, which provide the classes for the cipher (encrypting and decrypting), as well the key and algorithm parameters. Replace these Java packages with your own.
3. Update the following functions:

   `public UserMappingCryptoLDAP()`

   Replace the code for the cipher with the code for the cipher that matches the password encryption that your LDAP server uses.

   `public byte[] encrypt(byte[] plainValue)`

   This function provides the code that encrypts the passwords so that they can be stored on the LDAP server. This function also encrypts the LDAP connection password that is stored in the configuration file.

   Replace the code for this function with your own code that encrypts the **`plainValue`** parameter.

public byte[] decrypt(byte[] encryptedValue)
Replace the code for this function with your own code that decrypts the **encryptedValue** parameter.

private SecretKey getKey()

Replace the code for this function with the code to provide the plug-in with the key that is used to encrypt and decrypt your passwords.

public byte[] decode(String string)
The passwords are first encrypted and then encoded. This function provides the code for decoding the passwords before the passwords are decrypted. The encrypted passwords are encoded to transform the binary output of the encrypted password into ASCII characters.

Replace the code for this function with your own code that decodes the **string** parameter.

public String encode(byte[] bytes)
The passwords are first encrypted and then encoded. This function provides the code for encoding the binary output of the encrypted passwords. The encrypted passwords are encoded to transform the binary output of the encrypted password into ASCII characters.

Replace the code for this function with your own code that encodes the **bytes** parameter.

Modifying the UserMappingRepositoryLDAP sample file (Java programming language)

The `UserMappingRepositoryLDAP.java` file for the Java plug-in contains the functions for connecting to the LDAP server, retrieving user mappings, and disconnecting. Modify the sample code to match the object classes that are defined in the directory structure of your LDAP server.

About this task

To retrieve the user mappings from the LDAP server, the plug-in must search the directory for the user entries that have the attributes that define the user mapping. The following information is stored as attributes of a user:

- Remote server name
- Instance name
- Database name
- Remote user name
- Remote user password

The sample code assumes that the user entry is identified by the *inetOrgPerson* object class and that the user mapping entry is identified by the *IIUserMapping* object class. The Lightweight Directory Interchange Format (LDIF) sample files, `schema.ldif` and `entry.ldif`, load the sample schema and sample entries into the LDAP server.

The code in the `UserMappingRepositoryLDAP.java` file assumes that the `schema.ldif` file contains the LDIF code for objects and attributes that have the following names:

- *IIUserMapping* object
  - *IIRemoteServerName* attribute
  - *IIInstanceName* attribute

- *IIDatabaseName* attribute
- *IIRemotePassword* attribute
- *uid* attribute

You modify the `UserMappingRepositoryLDAP.java` file to match the schema that the LDAP server uses. The `UserMappingRepositoryLDAP.java` file searches for and retrieves the user mapping entries from the LDAP server. The LDIF files are provided as a sample schema and as a sample method of storing user mapping entries.

To modify the schema that the `UserMappingRepositoryLDAP.java` file uses:

Procedure

1. Locate the code: `private String UserObjectClassName = "inetOrgPerson"` and replace the *inetOrgPerson* value with the name of the object class that your LDAP server uses for user entries.
2. Optional: Change the attribute names that the plug-in uses. Replace the values for the *IIRemoteServerAttrName*, *IIInstanceAttrName*, *IIDatabaseAttrName*, and *IIRemotePasswordAttrName* variables with the attribute names that you choose.
3. Optional: If you use the LDIF files, ensure that the schema in the LDIF files matches the structure that the `UserMappingRepositoryLDAP.java` file searches.

Compiling the files for the user mapping plug-in (Java programming language)

After you modify the source files for the Java plug-in, you must compile them.

About this task

In the commands below, if the full path contains spaces, you must enclose the full path in quotation marks, for example `"C:\program files\sqllib\java\db2umplugin.jar"`. *%DB2PATH%* is the directory where DB2 is installed, for example, `C:\ProgramFiles\IBM\sqllib`. *inst_home* is the instance home directory.

The commands below assume that you are using the file naming convention that is based on the names of the classes. Replace *xxxx* with the name that you chose.

To compile the source files:

Procedure

1. Issue the compile command:

   Windows:

   ```
   javac -classpath" %DB2PATH%\java\db2umplugin.jar; ^
          %CLASSPATH%" -d . ^
          .\UserMappingRepositoryxxxx.java ^
          .\UserMappingCryptoxxxx.java ^
          .\UserMappingSetupxxxx.java ^
          .\UserMappingLookupXXXX.java
   ```

   UNIX:

   ```
   javac -classpath inst_home/sqllib/java/db2umplugin.jar:\
          $CLASSPATH -d . \
          ./UserMappingRepositoryxxxx.java \
          ./UserMappingCryptoxxxx.java \
          ./UserMappingSetupxxxx.java \
          ./UserMappingLookupxxxx.java
   ```

2. Archive the Java class files into a single Java Archive (JAR) file: The period symbol after the output file name, directs the command to find and place the files in the same directory. If you change the directory, use the appropriate file path for your operating system (for example, /home/user/folder or C:\test\folder).

   ```
   jar -cfM0 UserMappingRepository XXXX.jar .
   ```

What to do next

Creating the configuration file for the user mapping plug-in (Java programming language)

The configuration file stores the connection information that the Java plug-in uses to connect to the LDAP server.

About this task

To create the configuration file, run the configuration program, which prompts you for this information:

- Host name or IP address of the LDAP server
- Port number of the LDAP server (default is 389)
- Distinguished name of the LDAP subtree (for example, ou=ii,o=ibm,c=us)
- User ID for connecting to the LDAP server
- Password for connecting to the LDAP server
- SSL configuration

The input that you provide is used to create the `UserMappingRepositoryLDAP.cfg` file, which stores configuration information. If a password is required to connect to the LDAP server, the password is encrypted with the algorithm that you specify in the `UserMappingCryptoLDAP.java` file.

In the commands below, if the full path contains spaces, then you must enclose the full path in quotation marks, for example, `"C:\program files\sqllib\java\db2umplugin.jar"`. *%DB2PATH%* is the directory where DB2 is installed, for example, `C:\ProgramFiles\IBM\sqllib`. *inst_home* is the instance home directory.

To create the configuration file for the sample LDAP plug-in:

Windows:

```
java -classpath "%DB2PATH%\java\db2umplugin.jar; ^
 .\UserMappingRepositoryLDAP.jar;%CLASSPATH%" UserMappingSetupLDAP
```

UNIX:

```
java -classpath inst_home/sqllib/java/db2umplugin.jar: \
 ./UserMappingRepositoryLDAP.jar:$CLASSPATH UserMappingSetupLDAP
```

Testing the user mapping plug-in (Java programming language)

To test the Java plug-in outside of the federated server, develop an application that connects to and retrieves user mappings from the external repository.

About this task

You can develop a simple program that calls the `lookupUM()` method that your UserMappingRepository*XXXX* class inherited from the UserMappingRepository class to connect to the external repository and retrieve user mappings. You can

view the UserMappingLookupLDAP.java file that is located in the sqllib/samples/federated/umplugin/ldap/ directory.

In the commands below, if the full path contains spaces, then you must enclose the full path in quotation marks, for example "C:\program files\sqllib\java\db2umplugin.jar". *%DB2PATH%* is the directory where DB2 is installed, for example, C:\ProgramFiles\IBM\sqllib. *inst_home* is the instance home directory.

The test program must take the parameters that are required for identifying the user mapping:

- *remoteServerName* - Remote server name for the data source
- *iiAuthid* - Local user ID that is associated with the user mapping
- *iiInstance* - Instance name of the federated server
- *iiDatabase* - Database name on the federated server

To test the user mapping plug-in:

Windows:

```
java -classpath "%DB2PATH%\java\db2umplugin.jar; ^
  .\UserMappingRepositoryXXXX.jar;%CLASSPATH%" ^
  UserMappingLookupXXXX -server remoteServerName ^
   -authid iiAuthid ^
  -instance iiInstance -database iiDatabase
```

UNIX:

```
java -classpath inst_home/sqllib/java/db2umplugin.jar: ^
  ./UserMappingRepositoryXXXX.jar:$CLASSPATH \
  UserMappingLookupXXXX -server remoteServerName \
  -authid iiAuthid \
  -instance iiInstance -database iiDatabase
```

What to do next

Deploying the user mapping plug-in files (Java programming language)

After you compile and test the Java plug-in, deploy the files on the federated server.

About this task

In the commands below, *sqllib* is the full path to your DB2 installation. The commands assume that you use a file naming convention that is based on the names of the classes. Replace *xxxx* with the name that you chose.

To deploy the compiled files:

Procedure

Copy the UserMappingRepository*xxxx*.jar and UserMappingRepository*xxxx*.cfg files to the *sqllib*/function/directory.

Results

With the files in this directory, the federated server can load and call the classes contained in the files.

Configuring access to the user mapping plug-in (Java programming language)

Set the DB2_UM_PLUGIN option to configure the federated server to use a Java plug-in to retrieve user mappings.

Before you begin

Before you begin

Before you configure the federated server to access the user mappings in an external repository, you must perform these tasks:

- Develop a user mapping plug-in
- Deploy the plug-in on the federated server
- Update the database manager configuration:

```
db2 update dbm cfg using JDK_PATH your_jdk_path
db2 terminate
db2stop
db2start
```

About this task

The DB2_UM_PLUGIN option must contain the full path of the class, including the package name. If you develop a package, you must include the package name before the class name for example, `'package.classname'`. If you use the LDAP sample plug-in, specify the value `'UserMappingRepositoryLDAP'` in the DB2_UM_PLUGIN option. The sample plug-in is not developed as a package.

To configure the federated server to access the external repository:

Procedure

Choose how you want to implement the user mapping plug-in:

Method	SQL statement

Specify the user mapping plug-in when you create a wrapper	`CREATE WRAPPER wrapper-name`
	`  OPTIONS (`
	`  DB2_UM_PLUGIN 'UserMappingRepositoryLDAP'`
	`  );`

Alter an existing wrapper to specify the user mapping plug-in	`ALTER WRAPPER wrapper-name (`
	`  ADD DB2_UM_PLUGIN 'UserMappingRepositoryLDAP'`
	`  );`

Specify the user mapping plug-in when you create a server definition	`CREATE SERVER server_name`
	`  TYPE date_source_type`
	`  VERSION version_number`
	`  WRAPPER wrapper-name OPTIONS (`
	`  DB2_UM_PLUGIN 'UserMappingRepositoryLDAP'`
	`  );`

Alter an existing server definition to specify the user mapping plug-in	`ALTER SERVER server_name`
	`  OPTIONS (`
	`  ADD DB2_UM_PLUGIN 'UserMappingRepositoryLDAP'`
	`  );`

What to do next

After you set the DB2_UM_PLUGIN option, the federated server uses the connection information that you specify in the UserMappingRepository*XXXX*.cfg file to retrieve user mappings from the external repository. *XXXX* is the name of the plug-in.

To alter the wrapper to use a different plug-in

```
ALTER WRAPPER wrapper-name (
  SET DB2_UM_PLUGIN 'com.package_name.um.UserMappingRepositoryXXXX'
  );
```

To alter a server definition to use a different plug-in

```
ALTER SERVER server_name
  OPTIONS (
     SET DB2_UM_PLUGIN 'UserMappingRepositoryXXXX'
     );
```

Chapter 39. Oracle security in a federated system

Federation supports a set of Oracle security features.

If you implement these features in Oracle, the federated server can take advantage of them:

Oracle label security

The federated server supports Oracle Label Security, which you can use to secure data and ensure that only users with the appropriate authority can see it.

Administrators can use Oracle Label Security to apply security policies to each row in a table. The security policies determine a user's level of access to the data, based on the authorities that are granted to the user ID or session ID. When you create a nickname on an Oracle data source object, the federated server automatically detects whether the data source uses Oracle Label Security. If Oracle Label Security is being used, the nickname is not cached.

You can use the ALTER NICKNAME statement to allow or disallow caching. For example, if you created a nickname on a data source object with Oracle Label Security before federated support for this feature was available, you can alter the nickname to disallow caching. If you created a nickname on a data source object with Oracle Label Security and Oracle Label Security is removed, you can alter the nickname to allow caching.

A database administrator can choose to hide a label to prevent some users from knowing that a particular row exists. In this case, the column is hidden in the table. Nicknames with hidden label columns are not cached.

Oracle proxy authentication and federated trusted contexts

Create a physical connection to the Oracle data source, and then switch the connection to a different user on the same connection.

Using Oracle proxy authentication and federated trusted contexts reduces the network overhead of creating a separate network connection from the federated server to the Oracle database for each user, while still positively asserting the identity of the connected user to the Oracle data source. The application can switch from user to user, as required, to process transactions on the behalf of the users.

To configure this scenario, you perform these tasks:

1. On the Oracle server, use the Oracle ALTER USER statement to register each proxy user. Here Mary, who is the proxy user, is granted permission to use the proxy named BOSS and gains the role CLERK for the duration of the connection:

   ```
   ALTER USER MARY GRANT
   CONNECT THROUGH BOSS
   WITH ROLE CLERK
   ```

2. On the federated server, create the trusted context object:

```
CREATE TRUSTED CONTEXT MY_FED_TCX
BASED UPON CONNECTION USING SYSTEM AUTHID BOSS
ATTRIBUTES (ENCRYPTION 'NONE')
WITH USE FOR MARY WITHOUT AUTHENTICATION
ENABLE
```

With this configuration, the federated server can set up an end-to-end trusted connection from the client through the federated server to the Oracle data source. BOSS can establish a trusted connection, and MARY can reuse it.

To establish and reuse trusted connections, use the API that DB2 provides.

Chapter 40. Data source support for federated features

Refer to this table when you want to know whether or not a data source supports a specific federated feature.

Before you can use some of these features, you may need to set specific wrapper or server options or perform other tasks to enable the functionality. For more information, see the specific topics on each feature.

Table 28. Features and supported data sources

Feature	Data sources
Application savepoints with WRITE operations against nicknames	DB2 for Linux, UNIX, and Windows
Asynchrony optimization	All data sources
Cache tables	DB2 family Informix Microsoft SQL Server Oracle Sybase
Data import into nicknames	DB2 family Informix Microsoft SQL Server Oracle Sybase Teradata
Error tolerance in nested table expressions	DB2 family Informix JDBC Microsoft SQL Server ODBC Oracle Sybase Teradata
External user mapping repository	All data sources
Federated health indicators	DB2 family Excel Informix JDBC Microsoft SQL Server ODBC Oracle Sybase Table-structured files Teradata XML (root nicknames only)
Federated procedures	DB2 family, in trusted mode Oracle, in trusted mode Microsoft SQL Server, in trusted mode Sybase, in fenced mode, with the federated server installed on UNIX Sybase, in fenced mode or trusted mode, with the federated server installed on Linux or Microsoft Windows

Table 28. Features and supported data sources (continued)

Feature	Data sources
Federated trusted contexts	DB2 for Linux, UNIX, and Windows Version 9.5 DB2 for z/OS Version 9 Oracle
HTTP proxy	Web services XML
Connection-level isolation	DB2 family Informix JDBC Microsoft SQL Server ODBC Oracle Sybase
Label-based access control	DB2 for Linux, UNIX, and Windows Version 9.1 and 9.5 Oracle
LOB read and write operations	DB2 for z/OS DB2 for Linux, UNIX, and Windows DB2 for System i Oracle Teradata
LOB read-only operations	BioRS Informix JDBC Microsoft SQL Server ODBC Script Sybase Web services XML
Materialized query tables	All data sources, with specific restrictions
Nickname statistics update facility	BioRS DB2 family Excel Informix JDBC Microsoft SQL Server ODBC Oracle Sybase Table-structured files Teradata XML (root nicknames only)
Pass-through sessions	DRDA Informix Oracle Microsoft SQL Server Sybase Teradata
Remote XML data type	DB2 for Linux, UNIX, and Windows XML wrapper

Table 28. Features and supported data sources (continued)

Feature	Data sources
SOCKS proxy	BioRS Script Web services XML All relational data sources **Note:** To connect to relational data sources, you must set `DB2ENVLIST=SOCKS5C_CONFIG`.
Secure Socket Layer (SSL)	Web services XML
Statement-level isolation	DB2 family Microsoft SQL Server
Two-phase commit transactions	DB2 for Linux, UNIX, and Windows, in trusted mode DB2 for System i, in trusted mode DB2 for z/OS, in trusted mode Informix, in trusted mode Microsoft SQL Server, in trusted mode, with the federated server installed on Microsoft Windows Oracle, in trusted mode Sybase, in trusted mode, with the federated server installed on Microsoft Windows Sybase, in fenced mode, with the federated server installed on UNIX
Unicode support	All data sources

Chapter 41. Data source options reference

Each data source supports specific wrapper, server, user mapping, nickname, and column options.

BioRS options reference

To configure how the federated server and its users interact with a data source, set and modify wrapper, server, user mapping, nickname, and column options.

Wrapper options

The following tables list the options that apply to this data source and identify the required options that you must specify in the CREATE WRAPPER and CREATE SERVER statements.

Table 29. Wrapper options for BioRS

Name	Description
DB2_FENCED	Required. Specifies whether the wrapper runs in fenced mode or in trusted mode. Valid values are Y and N. The default is N; the wrapper runs in trusted mode.
DB2_UM_PLUGIN	Specifies the implementation of the user mapping plug-in. For a plug-in written in Java, specifies a case-sensitive string for the class name that corresponds to the user mapping repository class. For example, `"UserMappingRepositoryLDAP"`. For a plug-in written in C, specifies any valid C library name.
DB2_UM_PLUGIN_LANG	Specifies the language of the user mapping plug-in. Valid values are Java and C. The default is Java.
PROXY_SERVER_NAME	Specifies the name or IP address of the proxy server. This option is required if the value of PROXY_TYPE is HTTP or SOCKS. Valid IP addresses are in IPv4 (dot-separated) format or in IPv6 (colon-separated) format. Use IPv6 format only if IPv6 is configured.
PROXY_SERVER_PORT	Specifies the port or service name for the proxy service on the proxy server. This option is required if the value of PROXY_TYPE is HTTP or SOCKS. Valid values are a decimal port number from 1 to 32760 or a service name.

Table 29. Wrapper options for BioRS (continued)

Name	Description
PROXY_TYPE	Specifies the proxy type to use to access the Internet when the federated server is behind a firewall. Valid values are NONE, HTTP, and SOCKS. The default value is NONE. If you set this option to HTTP or SOCKS, you must also specify the PROXY_SERVER_NAME and PROXY_SERVER_PORT.

Server options

Table 30. Server options for BioRS

Name	Description
CASE_SENSITIVE	Specifies whether the BioRS server treats names in a case-sensitive manner. Valid values are Y and N. The default is Y; names are treated in a case-sensitive manner. In the BioRS product, a configuration parameter controls the case sensitivity of the data that is stored on the BioRS server. The CASE_SENSITIVE option and the configuration parameter must specify the same case sensitivity. If you need to change the value of the CASE_SENSITIVE option after you create the server definition, you must drop the server definition and create the definition and all nicknames again.
DB2_MAX_ASYNC_REQUESTS_ PER_QUERY	Specifies the maximum number of concurrent asynchronous requests from a query. Valid values are from -1 to 64000. The default is 1. -1 specifies that the federated query optimizer determines the number of requests. 0 specifies that the data source cannot accommodate additional asynchronous requests.
DB2_UM_PLUGIN	Specifies the implementation of the user mapping plug-in. For a plug-in written in Java, specifies a case-sensitive string for the class name that corresponds to the user mapping repository class. For example, `"UserMappingRepositoryLDAP"`. For a plug-in written in C, specifies any valid C library name.
DB2_UM_PLUGIN_LANG	Specifies the language for the user mapping plug-in. Valid values are Java and C. The default is Java.
NODE	Required. Specifies the DNS host name or IP address of the system on which the BioRS query tool is available. Valid IP addresses are in IPv4 (dot-separated) format or IPv6 (colon-separated) format. Use IPv6 format only if IPv6 is configured. The default value is `localhost`.

Table 30. Server options for BioRS (continued)

Name	Description
PORT	Specifies the port for connections to the BioRS server. Valid values are a numeric port or a TCP/IP service name. The default is 5014.
PROXY_AUTHID	Specifies the user name for proxy server authentication.
PROXY_PASSWORD	Specifies the password for proxy server authentication.
PROXY_SERVER_NAME	Specifies the name or IP address of the proxy server. Valid IP addresses are in IPv4 (dot-separated) format or in IPv6 (colon-separated) format. Use IPv6 format only if IPv6 is configured.
PROXY_SERVER_PORT	Specifies the port or service name for the proxy service on the proxy server. Valid values are a decimal port number from 1 to 32760 or a service name.
PROXY_TYPE	Specifies the proxy type to use to access the Internet when the federated server is behind a firewall. Valid values are NONE, HTTP, and SOCKS. The default value is NONE.
TIMEOUT	Specifies the maximum time, in minutes, that the federated server waits for a response from the remote server. The default is 10.

User mapping options

Table 31. User mapping options for BioRS

Option	Description
GUEST	Specifies that the GUEST BioRS authentication ID is used to perform operations. Valid values are Y and N. The default is N; the GUEST BioRS ID is not used. This option is not valid if you specify the REMOTE_AUTHID and REMOTE_PASSWORD options.
PROXY_AUTHID	Specifies the user name for proxy server authentication.
PROXY_PASSWORD	Specifies the password for proxy server authentication. The password is encrypted when it is stored in the federated database catalog.
REMOTE_AUTHID	Specifies the remote user ID to which the local user ID is mapped. If you do not specify this option, the ID that is used to connect to the federated database is used.
REMOTE_PASSWORD	Specifies the remote password for the remote user ID. If you do not specify this option, the password that is used to connect to the federated database is used.

Nickname options

Table 32. Nickname options for BioRS

Option	Description
REMOTE_OBJECT	Specifies the name of the BioRS databank that is associated with the nickname. This name determines the schema and the BioRS databank for the nickname. The name also specifies the relationship of the nickname to other nicknames. The case sensitivity of this option depends on the case sensitivity of the BioRS server and on the value of the CASE_SENSITIVE server option. You cannot use the ALTER NICKNAME statement to change or delete this name. If the name of the BioRS databank changes, you must delete the nickname and then create it again.
TIMEOUT	Specifies the maximum time, in minutes, to wait for a response from the data source server. The default is 10.

Column options

Table 33. Column options for BioRS

Option	Description
ELEMENT_NAME	Specifies the BioRS element name. The case sensitivity of this name depends on the case sensitivity of the BioRS server and on the value of the CASE_SENSITIVE server option. You must specify the BioRS element name only if it is different from the column name.
IS_INDEXED	Specifies whether the corresponding column is indexed and, therefore, can be referenced in a predicate. Valid values are Y and N. The default is N; the column is not indexed.
REFERENCED_OBJECT	Specifies the name of the BioRS databank that is referenced by the current column. The case sensitivity of this name depends on the case sensitivity of the BioRS server and on the value of the CASE_SENSITIVE server option. This option is valid only for columns that have the BioRS data type of Reference.

DB2 database options reference

To configure how the federated server and its users interact with a data source, set and modify wrapper, server, user mapping, nickname, and column options.

Wrapper options

The following tables list the options that apply to DB2 data sources and identify the required options that you must specify.

Table 34. Wrapper options for DB2 data sources

Name	Description
DB2_FENCED	Required. Specifies whether the wrapper runs in fenced mode or in trusted mode. Valid values are Y and N. The default is N; the wrapper runs in trusted mode.
DB2_UM_PLUGIN	Specifies the implementation of the user mapping plug-in. For a plug-in written in Java, specifies a case-sensitive string for the class name that corresponds to the user mapping repository class. For example, `"UserMappingRepositoryLDAP"`. For a plug-in written in C, specifies any valid C library name.
DB2_UM_PLUGIN_LANG	Specifies the language of the user mapping plug-in. Valid values are Java and C. The default is Java.

Server options

Table 35. Server options for DB2 data sources

Name	Description
APP_ISOLATION_ENABLE	Specifies the isolation level for a client connection. Valid values are Y and N. The default is N; the federated server does not set the isolation level for a connection to the client. In this case, the client can use the isolation level set in configuration file, for example db2cli.ini. Y specifies that the federated server obtains the isolation level from the application and sets the application isolation as a connection attribute.
COLLATING_SEQUENCE	Specifies whether the data source uses the same default collating sequence as the federated database. Valid values are Y, N, and I. I specifies it is not case-sensitive. The default is Y. The collating sequence specified for the federated server must match the collating sequence on the remote data source.
COMM_RATE	Specifies the communication rate, in megabytes per second, between the federated server and the data source server. Valid values are whole numbers the are greater than 0 and less than 2147483648. The default is 2.

Table 35. Server options for DB2 data sources (continued)

Name	Description
CPU_RATIO	Specifies how much faster or slower the data source CPU is when compared to federated server CPU. Valid values are greater than 0 and less than $1x10^{23}$. The default is 1.0. Values can be expressed in any valid double notation, for example, 123E10, 123, or 1.21E4. A setting of 1 indicates that the federated server and the data source server have the same CPU speed; a 1:1 ratio. A setting of 0.5 indicates that the federated server CPU speed is 50% slower than the data source CPU. A setting of 2 indicates that the federated CPU is twice as fast as the data source CPU.
DATE_COMPAT	Specifies whether the date_compat parameter is applied to the database. Valid values are Y and N. The default is N. This server option is only valid for DB2 Database for Linux, UNIX, and Windows, Version 9.7 or later. In InfoSphere Federation Server Version 9.7 Fix Pack 2 and later, when you run the CREATE SERVER statement, this server option is automatically configured based on the configuration of your data source. If you attempt to manually configure this server option, you receive the SQL1841N message.
DBNAME	Required. Specifies the specific database to use for the initial remote DB2 database connection. This specific database is the database alias for the remote DB2 database that is cataloged on the federated server by using the CATALOG DATABASE command or the DB2 Configuration Assistant.
DB2_MAXIMAL_PUSHDOWN	Specifies the primary criteria that the query optimizer uses to choose an access plan. Valid values are Y and N. The default is N; the query optimizer chooses the plan that has the lowest estimated cost. Y specifies that the query optimizer choose the access plan that pushes down the most query operations to the data source.
DB2_MAX_ASYNC_REQUESTS_ PER_QUERY	Specifies the maximum number of concurrent asynchronous requests from a query. Valid values are from -1 to 64000. The default is 1. -1 specifies that the federated query optimizer determines the number of requests. 0 specifies that the data source cannot accommodate additional asynchronous requests.

Table 35. Server options for DB2 data sources (continued)

Name	Description
DB2_TWO_PHASE_COMMIT	Specifies whether the federated server connects to the data source in two-phase commit protocol or one-phase commit protocol. Valid values are Y and N. The default is N; the federated server uses the one-phase commit protocol to connect. Y specifies that the federated server uses two-phase commit protocol to connect.
DB2_UM_PLUGIN	Specifies the implementation of the user mapping plug-in. For a plug-in written in Java, specifies a case-sensitive string for the class name that corresponds to the user mapping repository class. For example, `"UserMappingRepositoryLDAP"`. For a plug-in written in C, specifies any valid C library name.
DB2_UM_PLUGIN_LANG	Specifies the language of the user mapping plug-in. Valid values are Java and C. The default is Java.
FED_PROXY_USER	Specifies the authorization ID to use to establish all outbound trusted connections when the inbound connection is non-trusted. The user whose ID is specified in this option must have a user mapping that specifies both the REMOTE_AUTHID and REMOTE_PASSWORD options. **Restriction:** This server option is only valid for DB2 Database for Linux, UNIX, and Windows Version 9.5 and later and DB2 for z/OS Version 9 and later.
FOLD_ID	Specifies the case for the user ID that is sent to the data source. There is no default value. The federated server sends the user ID in uppercase; then if the uppercase user ID fails, the server sends the user ID in lowercase. Valid values are U (uppercase), L (lowercase), and N (null). Avoid using the null setting, which might result in poor performance.
FOLD_PW	Specifies the case for the password that is sent to the data source. There is no default value; the federated server sends the password in uppercase; then if the uppercase password fails, the server sends the password in lowercase. Valid values are U (uppercase), L (lowercase), and N (null). Avoid using the null setting, which might result in poor performance.

Table 35. Server options for DB2 data sources (continued)

Name	Description
IO_RATIO	Specifies how much faster or slower the data source I/O system runs when compared to the federated server I/O system. Valid values are greater than 0 and less than $1x10^{23}$. The default is 1.0. Values can be expressed in any valid double notation, for example, 123E10, 123, or 1.21E4. A setting of 1 indicates that the federated server and the data source server have the same I/O speed; a 1:1 ratio. A setting of 0.5 indicates that the federated server speed is 50% slower than the data source speed. A setting of 2 indicates that the federated speed is twice as fast as the data source speed.
NO_EMPTY_STRING	Specifies whether the remote data source server can contain empty strings. Valid values are Y and N. The default value varies depending on your remote data source. For remote Oracle data sources, the default is Y; all empty string values are converted to NULL values. For all other remote data sources, the default is N; the data source can contain empty strings. You can improve your systems performance by setting this option to Y in system configurations where the federated server is in VARCHAR2 compatible mode but the remote data source is not VARCHAR2 compatible.
NUMBER_COMPAT	Specifies whether the data source server supports the NUMBER data type. Valid values are Y and N. The default is N; the data source server does not support the NUMBER data type. In systems where the federated server does not support the NUMBER data type but the data source server does, you must set the NUMBER_COMPAT option to Y because the data source server can return DECFLOAT results that are outside of the range of the DECIMAL data type and cause the SQLSTATE 560BD error. **Restriction:** This server option is only valid for DB2 Database for Linux, UNIX, and Windows Version 9.7 and later. In InfoSphere Federation Server Version 9.7 Fix Pack 2 and later, when you run the CREATE SERVER statement, this server option is automatically configured based on the configuration of your data source. If you attempt to manually configure this server option, you receive the SQL1841N message.

Table 35. Server options for DB2 data sources (continued)

Name	Description
OLD_NAME_GEN	Specifies how to convert the column names and index names that are in the data source into nickname column names and local index names for the federated server. Valid values are Y and N. The default is N; the generated names closely match the names in the data source. Y specifies that the generated names are the same as the names that were created in IBM WebSphere Federation Server Version 9 and earlier. Thus, the names might not closely match the data source names.
PUSHDOWN	Specifies whether the federated server allows the data source to evaluate operations. Valid values are Y and N. The default is Y; the data source evaluates operations. N specifies that the federated server send SQL statements that include only SELECT with column names. Predicates, such as WHERE=; column and scalar functions, such as MAX and MIN; sorts, such as ORDER BY OR GROUP BY; and joins are not included in any SQL that the federated server sends to the data source.
SAME_DECFLT_ROUNDING	Specifies whether the rounding mode of both the federated server and data source server are using the same DECFLOAT rounding mode settings. Valid values are Y and N. The default is N; the federated server and remote server have different DECFLOAT rounding mode settings. **Important:** If you set this option to Y when the rounding modes are different between the federated server and data source server, you might receive incorrect DECFLOAT rounding results. To configure an existing federated server and data source server that use the same DECFLOAT rounding mode setting, use the ALTER SERVER statement. **Restriction:** This server option is only valid for DB2 Database for Linux, UNIX, and Windows Version 9.5 and later. In InfoSphere Federation Server Version 9.7 Fix Pack 2 and later, when you run the CREATE SERVER statement, this server option is automatically configured based on the configuration of your data source. If you attempt to manually configure this server option, you receive the SQL1841N message.

Table 35. Server options for DB2 data sources (continued)

Name	Description
VARCHAR2_COMPAT	Specifies whether the remote data source is VARCHAR2 compatible. The valid values Y and N. The default value varies depending on the remote data source. For remote Oracle data sources, the default is Y; the data source is VARCHAR2 compatible. For all other remote data sources, the default is N; the data source is not set to VARCHAR2 compatible mode. You must set this server option to Y if your DB2 Database for Linux, UNIX, and Windows, ODBC, or JDBC data source is configured in VARCHAR2 compatible mode. In InfoSphere Federation Server Version 9.7 Fix Pack 2 and later, when you run the CREATE SERVER statement, this server option is automatically configured based on the configuration of your data source. If you attempt to manually configure this server option, you receive the SQL1841N message.

User mapping options

Table 36. User mapping options for DB2 data sources

Option	Description
FED_PROXY_USER	Specifies the authorization ID to use to establish all outbound trusted connections when the inbound connection is non-trusted. The user whose ID is specified in this option must have a user mapping that specifies both a REMOTE_AUTHID and a REMOTE_PASSWORD. If you specify the FED_PROXY_USER user mapping option, you must also specify the FED_PROXY_USER server option. **Restriction:** This server option is only valid for DB2 Database for Linux, UNIX, and Windows Version 9.5 and later and DB2 for z/OS Version 9 and later.
ACCOUNTING_STRING	Required if accounting information must be passed. Specifies a DRDA accounting string. Valid values include any string that has 255 characters or fewer.
REMOTE_AUTHID	Specifies the remote user ID to which the local user ID is mapped. If you do not specify this option, the ID that is used to connect to the federated database is used.
REMOTE_PASSWORD	Specifies the remote password for the remote user ID. If you do not specify this option, the password that is used to connect to the federated database is used.

Table 36. User mapping options for DB2 data sources (continued)

Option	Description
USE_TRUSTED_CONTEXT	Specifies whether the user mapping is trusted. Valid values are Y and N. The default is N; the user mapping is not trusted and can be used only in non-trusted federated outbound connections. Y specifies that the user mapping is trusted and can be used in both trusted and non-trusted outbound federated connections. **Restriction:** This server option is only valid for DB2 Database for Linux, UNIX, and Windows Version 9.5 and later and DB2 for z/OS Version 9 and later.

Column options

Table 37. Column options for DB2 data sources

Option	Description
NUMERIC_STRING	Specifies how to treat numeric strings. The default is N. If the data source string column contains only numeric strings and no other characters, including blanks, set the NUMERIC_STRING option to Y. When NUMERIC_STRING is set to Y for a column, the query optimizer recognizes that the column contains no blanks that could interfere with the sorting of the data in the column. Use this option when the collating sequence of a data source is different from the collating sequence that the federated server uses. Columns that use this option are not excluded from remote evaluation because of a different collating sequence.
NO_EMPTY_STRING	Specifies whether the remote data source server can contain empty strings. Valid values are Y and N. The default value varies depending on your remote data source. For remote Oracle data sources, the default is Y; all empty string values are converted to NULL values. For all other remote data sources, the default is N; the data source can contain empty strings.
XML_ROOT	Specifies the XML root element to add to the values of an XML column that references an XML sequence. This option ensures that the values of the XML column are a well-formed XML document.

Excel options reference

To configure how the federated server and its users interact with a data source, set and modify wrapper, server, and nickname options.

Wrapper options

The following tables list the options that apply to this data source and identify the required options that you must specify.

Table 38. Wrapper options for Excel

Name	Description
DB2_FENCED	Required. Specifies whether the wrapper runs in fenced mode or in trusted mode. Valid values are Y and N. The default is N; the wrapper runs in trusted mode.

Server options

Table 39. Server options for Excel

Name	Description
DB2_MAX_ASYNC_REQUESTS_PER_ QUERY	Specifies the maximum number of concurrent asynchronous requests from a query. Valid values are from -1 to 64000. The default is 1. -1 specifies that the federated query optimizer determines the number of requests. 0 specifies that the data source cannot accommodate additional asynchronous requests.

Nickname options

Table 40. Nickname options for Excel

Option	Description
FILE_PATH	Required. Specifies the fully qualified directory path and file name of the Excel spreadsheet that you want to access.
RANGE	Specifies the range of cells to use, for example, A1:C100. The value before the colon specifies the top left cell of the range. The value after the colon specifies the bottom right cell of the range.

Informix options reference

To configure how the federated server and its users interact with a data source, set and modify wrapper, server, user mapping, and column options.

Wrapper options

The following tables list the options that apply to this data source and identify the required options that you must specify.

Table 41. Wrapper options for Informix

Name	Description
DB2_FENCED	Required. Specifies whether the wrapper runs in fenced mode or in trusted mode. Valid values are Y and N. The default is N; the wrapper runs in trusted mode.
DB2_UM_PLUGIN	Specifies the implementation of the user mapping plug-in. For a plug-in written in Java, specifies a case-sensitive string for the class name that corresponds to the user mapping repository class. For example, `"UserMappingRepositoryLDAP"`. For a plug-in written in C, specifies any valid C library name.
DB2_UM_PLUGIN_LANG	Specifies the language of the user mapping plug-in. Valid values are Java and C. The default is Java.

Server options

Table 42. Server options for Informix

Name	Description
COLLATING_SEQUENCE	Specifies whether the data source uses the same default collating sequence as the federated database. Valid values are Y, N, and I. I specifies a case-insensitive. The default is Y. The collating sequence specified for the federated server must match the collating sequence on the remote data source.
COMM_RATE	Specifies the communication rate, in megabytes per second, between the federated server and the data source server. Valid values are whole numbers the are greater than 0 and less than 2147483648. The default is 2.
CPU_RATIO	Specifies how much faster or slower the data source CPU is when compared to federated server CPU. Valid values are greater than 0 and less than $1x10^{23}$. The default is 1.0. Values can be expressed in any valid double notation, for example, 123E10, 123, or 1.21E4. A setting of 1 indicates that the federated server and the data source server have the same CPU speed; a 1:1 ratio. A setting of 0.5 indicates that the federated server CPU speed is 50% slower than the data source CPU. A setting of 2 indicates that the federated CPU is twice as fast as the data source CPU.
DBNAME	Required. Specifies the name of the Informix database that you want to access.

Table 42. Server options for Informix (continued)

Name	Description
DB2_MAXIMAL_PUSHDOWN	Specifies the primary criteria that the query optimizer uses to choose an access plan. Valid values are Y and N. The default is N; the query optimizer chooses the plan that has the lowest estimated cost. Y specifies that the query optimizer choose the access plan that pushes down the most query operations to the data source. If more than one access plan meets this criteria, plan that has the lowest cost is chosen.
DB2_MAX_ASYNC_REQUESTS_ PER_QUERY	Specifies the maximum number of concurrent asynchronous requests from a query. Valid values are from -1 to 64000. The default is 1. -1 specifies that the federated query optimizer determines the number of requests. 0 specifies that the data source cannot accommodate additional asynchronous requests.
DB2_TWO_PHASE_COMMIT	Specifies whether the federated server connects to the data source in two-phase commit protocol or one-phase commit protocol. Valid values are Y and N. The default is N; the federated server uses the one-phase commit protocol to connect. Y specifies that the federated server uses two-phase commit protocol to connect.
DB2_UM_PLUGIN	Specifies the implementation of the user mapping plug-in. For a plug-in written in Java, specifies a case-sensitive string for the class name that corresponds to the user mapping repository class. For example, `"UserMappingRepositoryLDAP"`. For a plug-in written in C, specifies any valid C library name.
DB2_UM_PLUGIN_LANG	Specifies the language of the user mapping plug-in. Valid values are Java and C. The default is Java.
FOLD_ID	Specifies the case for the user ID that is sent to the data source. There is no default value; the federated server sends the user ID in uppercase; then if the uppercase user ID fails, the server sends the user ID in lowercase. Valid values are U (uppercase), L (lowercase), and N (null). Avoid using the null setting, which might result in poor performance.
FOLD_PW	Specifies the case for the password that is sent to the data source. There is no default value; the federated server sends the password in uppercase; then if the uppercase password fails, the server sends the password in lowercase. Valid values are U (uppercase), L (lowercase), and N (null). Avoid using the null setting, which might result in poor performance.

Table 42. Server options for Informix (continued)

Name	Description
INFORMIX_CLIENT_LOCALE	Specifies the CLIENT_LOCALE to use for the connection between the federated server and the data source server. The value is any valid Informix locale. If you do not specify this option, the CLIENT_LOCALE environment variable is set to the value that is specified in the `db2dj.ini` file. If the `db2dj.ini` does not specify the CLIENT_LOCALE environment variable, the INFORMIX_CLIENT_LOCALE is set to the Informix locale that most closely matches the code page and territory of the federated database.
INFORMIX_DB_LOCALE	Specifies the Informix DB_LOCALE to use for the connection between the federated server and the data source server. If the INFORMIX_DB_LOCALE option is not specified, the Informix DB_LOCALE environment variable is set to the value specified in the `db2dj.ini` file. If the `db2dj.ini` file does not specify a value, the Informix DB_LOCALE environment variable is not set.

INFORMIX_LOCK_MODE	Specifies the lock mode to set for an Informix data source. The Informix wrapper issues the SET LOCK MODE command immediately after connecting to an Informix data source. Valid values are W, N, and a number. The default is W; the wrapper waits an unlimited amount of time for the lock to be released. N specifies not to wait; an error is returned immediately. Use a number to specify the maximum amount of time, in seconds, to wait. If a deadlock or timeout occurs, use the ALTER SERVER statement to change the value of the INFORMIX_LOCK_MODE option. For example:
	`ALTER SERVER TYPE informix`
	`VERSION 9`
	`WRAPPER informix`
	`OPTIONS`
	`(ADD informix_lock_mode '60')`

Table 42. Server options for Informix (continued)

Name	Description
IO_RATIO	Specifies how much faster or slower the data source I/O system runs when compared to the federated server I/O system. Valid values are greater than 0 and less than $1x10^{23}$. The default is 1.0. Values can be expressed in any valid double notation, for example, 123E10, 123, or 1.21E4. A setting of 1 indicates that the federated server and the data source server have the same I/O speed; a 1:1 ratio. A setting of 0.5 indicates that the federated server speed is 50% slower than the data source speed. A setting of 2 indicates that the federated speed is twice as fast as the data source speed.
IUD_APP_SVPT_ENFORCE	Specifies whether the federated server enforces the use of application savepoint statements. Valid values are Y and N. The default is Y; if the data source does not enforce application savepoint statements and an error occurs during an insert, update, or delete operation, the federated server rolls back the transaction and SQL error code SQL1476N is returned. It is recommended that you use the default setting.
NODE	Required. Specifies the name by which the data source is defined as an instance to its relational database management system.
OLD_NAME_GEN	Specifies how to convert the column names and index names that are in the data source into nickname column names and local index names for the federated server. Valid values are Y and N. The default is N; the generated names closely match the names in the data source. Y specifies that the generated names are the same as the names that were created in IBM WebSphere Federation Server Version 9 and earlier. Thus, the names might not closely match the data source names.
PUSHDOWN	Specifies whether the federated server allows the data source to evaluate operations. Valid values are Y and N. The default is Y; the data source evaluates operations. N specifies that the federated server send SQL statements that include only SELECT with column names. Predicates, such as WHERE=; column and scalar functions, such as MAX and MIN; sorts, such as ORDER BY OR GROUP BY; and joins are not included in any SQL that the federated server sends to the data source.

User mapping options

Table 43. User mapping options for Informix

Name	Description
REMOTE_AUTHID	Specifies the remote user ID to which the local user ID is mapped. If you do not specify this option, the ID that is used to connect to the federated database is used.
REMOTE_PASSWORD	Specifies the remote password for the remote user ID. If you do not specify this option, the password that is used to connect to the federated database is used.

Column options

Table 44. Column options for Informix

Name	Description
BINARY_REP	Identifies specific binary data type columns. You can set the value of the BINARY_REP column option to Y to force the federated server to push down the SQL_BINARY data type. Enabling the query optimizer to perform the comparison of binary type columns at the remote data source can improve query performance.
NUMERIC_STRING	Specifies how to treat numeric strings. The default is N. If the data source string column contains only numeric strings and no other characters, including blanks, set the NUMERIC_STRING option to Y. When NUMERIC_STRING is set to Y for a column, the query optimizer recognizes that the column contains no blanks that could interfere with the sorting of the data in the column. Use this option when the collating sequence of a data source is different from the collating sequence that the federated server uses. Columns that use this option are not excluded from remote evaluation because of a different collating sequence.

JDBC options reference

To configure how the federated server and its users interact with a data source, set and modify wrapper, server, user mapping, and column options.

Wrapper options

The following tables list the options that apply to this data source and identify the required options that you must specify.

Table 45. Wrapper options for JDBC

Name	Description
DB2_FENCED	Required. Specifies whether the wrapper runs in fenced mode or in trusted mode. The only valid value is Y because the DB2 server only supports loading the JVM in fenced mode. The default is Y; the wrapper runs in fenced mode.
DB2_UM_PLUGIN	Specifies the implementation of the user mapping plug-in. For a plug-in written in Java, specifies a case-sensitive string for the class name that corresponds to the user mapping repository class. For example, `"UserMappingRepositoryLDAP"`. For a plug-in written in C, specifies any valid C library name.
DB2_UM_PLUGIN_LANG	Specifies the language of the user mapping plug-in. Valid values are Java and C. The default is Java.

Server options

Table 46. Server options for JDBC

Name	Description
COLLATING_SEQUENCE	Specifies whether the data source uses the same default collating sequence as the federated database. Valid values are Y, N, and I. I specifies a case-insensitive. The default is Y. The collating sequence specified for the federated server must match the collating sequence on the remote data source.
COMM_RATE	Specifies the communication rate, in megabytes per second, between the federated server and the data source server. Valid values are whole numbers the are greater than 0 and less than 2147483648. The default is 2.
CPU_RATIO	Specifies how much faster or slower the data source CPU is when compared to federated server CPU. Valid values are greater than 0 and less than $1x10^{23}$. The default is 1.0. Values can be expressed in any valid double notation, for example, 123E10, 123, or 1.21E4. A setting of 1 indicates that the federated server and the data source server have the same CPU speed; a 1:1 ratio. A setting of 0.5 indicates that the federated server CPU speed is 50% slower than the data source CPU. A setting of 2 indicates that the federated CPU is twice as fast as the data source CPU.

Table 46. Server options for JDBC (continued)

Name	Description
DATEFORMAT	Specifies the date format that the data source uses. Use 'DD', 'MM', and 'YY' or 'YYYY' to specify the date format. You can specify a delimiter such as a space, a hyphen, or a comma. For example, the format `'YYYY-MM-DD'` specifies a date such as 1958-10-01. The value can contain null values.
DB2_MAXIMAL_PUSHDOWN	Specifies the primary criteria that the query optimizer uses to choose an access plan. Valid values are Y and N. The default is N; the query optimizer chooses the plan that has the lowest estimated cost. Y specifies that the query optimizer choose the access plan that pushes down the most query operations to the data source.
DB2_MAX_ASYNC_REQUESTS_PER_ QUERY	Specifies the maximum number of concurrent asynchronous requests from a query. Valid values are from -1 to 64000. The default is 0. -1 specifies that the federated query optimizer determines the number of requests. 0 specifies that the data source cannot accommodate additional asynchronous requests.
DB2_UM_PLUGIN	Specifies the implementation of the user mapping plug-in. For a plug-in written in Java, specifies a case-sensitive string for the class name that corresponds to the user mapping repository class. For example, `"UserMappingRepositoryLDAP"`. For a plug-in written in C, specifies any valid C library name.
DB2_UM_PLUGIN_LANG	Specifies the language of the user mapping plug-in. Valid values are Java and C. The default is Java.
DRIVER_CLASS	Specifies the JDBC driver library. You can register the server against multiple JDBC data sources if the JDBC driver conforms to the JDBC specification version 3.0 and later. See your JDBC driver documentation for the JDBC specifications and information on how to set the DRIVER_CLASS server option. **Example** In this example, the `com.ibm.db2.jcc.DB2Driver` JDBC driver library is specified: `DRIVER_CLASS 'com.ibm.db2.jcc.DB2Driver'` **Important:** If you specify this option, you must also specify the URL server option.

Table 46. Server options for JDBC (continued)

Name	Description
DRIVER_PACKAGE	Specifies the JDBC driver packages. Use a path separator to specify multiple driver class packages. Use a semicolon in the Windows operating systems and a colon in Linux and Unix operating systems. **Example** In this example, you specify multiple driver packages with a colon on Linux operating systems: `DRIVER_PACKAGE '/path1/file1.jar: /path2/file2.jar'`
FOLD_ID	Specifies the case for the user ID that is sent to the data source. There is no default value; the federated server sends the user ID in uppercase; then if the uppercase user ID fails, the server sends the user ID in lowercase. Valid values are U (uppercase), L (lowercase), and N (null). Avoid using the null setting, which might result in poor performance.
FOLD_PW	Specifies the case for the password that is sent to the data source. There is no default value; the federated server sends the password in uppercase; then if the uppercase password fails, the server sends the password in lowercase. Valid values are U (uppercase), L (lowercase), and N (null). Avoid using the null setting, which might result in poor performance.
IO_RATIO	Specifies how much faster or slower the data source I/O system runs when compared to the federated server I/O system. Valid values are greater than 0 and less than $1x10^{23}$. The default is 1.0. Values can be expressed in any valid double notation, for example, 123E10, 123, or 1.21E4. A setting of 1 indicates that the federated server and the data source server have the same I/O speed; a 1:1 ratio. A setting of 0.5 indicates that the federated server speed is 50% slower than the data source speed. A setting of 2 indicates that the federated speed is twice as fast as the data source speed.

Table 46. Server options for JDBC (continued)

Name	Description
JDBC_LOG	Specifies whether the JDBC wrapper creates log files for error tracing. Valid values are Y and N. The default is N; log file are not created. If this server option is set to Y, the JDBC wrapper writes JDBC log files to the `jdbc_wrapper_prod_id.log` file, where *prod_id* is the product ID. The log file is stored in the directory specified by the DB2 database manager configuration parameter DIAGPATH. The default directory on UNIX systems is `inst_home/sqllib/db2dump`. **Recommendation:** Setting this server option to YES will impact the performance of your system and you should not enable logging in production systems.
OLD_NAME_GEN	Specifies how to convert the column names and index names that are in the data source into nickname column names and local index names for the federated server. Valid values are Y and N. The default is N; the generated names closely match the names in the data source. Y specifies that the generated names are the same as the names that were created in IBM WebSphere Federation Server Version 9 and earlier. Thus, the names might not closely match the data source names.
PUSHDOWN	Specifies whether the federated server allows the data source to evaluate operations. Valid values are Y and N. The default is Y; the data source evaluates operations. N specifies that the federated server send SQL statements that include only SELECT with column names. Predicates, such as WHERE=; column and scalar functions, such as MAX and MIN; sorts, such as ORDER BY OR GROUP BY; and joins are not included in any SQL that the federated server sends to the data source.
TIMEFORMAT	Specifies the time format that the data source uses. Use 'hh12', 'hh24', 'mm', 'ss', 'AM', and 'A.M.' to specify the time format. For example, the format `'hh24:mm:22'` specifies a time such as 16:00:00. The format `'hh12:mm:ss AM'` specifies a time such as 8:00:00 AM. The value can contain null values.

Table 46. Server options for JDBC (continued)

Name	Description
TIMESTAMPFORMAT	Specifies the timestamp format that the data source uses. Valid values are in the format that the DATEFORMAT option and the TIMEFORMAT option use. Specify 'n' for a tenth of a second, 'nn' for a hundredth of a second, 'nnn' for milliseconds, and so on, up to 'nnnnnn' for microseconds. For example, the format `'YYY-MM-DD-hh24:mm:ss.nnnnnn'` specifies a timestamp such as 1994-01-01-24:00:00.000000. The value can contain null values.
URL	Specifies the JDBC connection string of the remote server. The JDBC connection string consists of three parts that are all separated by a colon: • Database protocol • Database type name or connectivity driver name • Database identity through an alias or sub-name **Example** In this example, the JDBC connection string is `jdbc:db2://cn.ibm.com:50471/testdb`: `URL 'jdbc:db2://cn.ibm.com:50471/testdb'` You can register the server against multiple JDBC data sources if the JDBC driver conforms to the JDBC specification version 3.0 and above. See your JDBC driver documentation for the JDBC specifications and information on how to set the URL server option. **Important:** If you specify this option, you must also specify the DRIVER_CLASS server option.
VARCHAR_NO_TRAILING_BLANKS	Specifies whether the data source contains VARCHAR columns that contain at least one trailing blank character. The default is N; VARCHAR columns contain at least one trailing blank character.

User mapping options

Table 47. User mapping options for JDBC

Name	Description
REMOTE_AUTHID	Specifies the remote user ID to which the local user ID is mapped. If you do not specify this option, the ID that is used to connect to the federated database is used.

Table 47. User mapping options for JDBC (continued)

Name	Description
REMOTE_PASSWORD	Specifies the remote password for the remote user ID. If you do not specify this option, the password that is used to connect to the federated database is used.

Column options

Table 48. Column options for JDBC

Name	Description
NUMERIC_STRING	Specifies whether the column contains strings of numeric characters that include blanks. Valid values are Y and N. The default is N; the column does not contain numeric strings that include blanks. If the column contains only numeric strings followed by trailing blanks, do not specify Y. When NUMERIC_STRING is set to Y for a column, the query optimizer recognizes that the column contains no blanks that could interfere with the sorting of the data in the column. Use this option when the collating sequence of a data source is different from the collating sequence that the federated server uses. Columns that use this option are not excluded from remote evaluation because of a different collating sequence.
VARCHAR_NO_TRAILING_BLANKS	Specifies whether there is at least one trailing blank in the VARCHAR column.

Microsoft SQL Server options reference

To configure how the federated server and its users interact with a data source, set and modify wrapper, server, user mapping, and column options.

Wrapper options

The following tables list the options that apply to this data source and identify the required options that you must specify.

Table 49. Wrapper options for Microsoft SQL Server

Name	Description
DB2_FENCED	Required. Specifies whether the wrapper runs in fenced mode or in trusted mode. Valid values are Y and N. The default is N; the wrapper runs in trusted mode.

Table 49. Wrapper options for Microsoft SQL Server (continued)

Name	Description
DB2_UM_PLUGIN	Specifies the implementation of the user mapping plug-in. For a plug-in written in Java, specifies a case-sensitive string for the class name that corresponds to the user mapping repository class. For example, `"UserMappingRepositoryLDAP"`. For a plug-in written in C, specifies any valid C library name.
DB2_UM_PLUGIN_LANG	Specifies the language of the user mapping plug-in. Valid values are Java and C. The default is Java.

Server options

Table 50. Server options for Microsoft SQL Server

Name	Description
CODEPAGE	Specifies the code page that corresponds to the coded character set of the data source client configuration. On UNIX and Microsoft Windows systems that use a non-Unicode federated database, the default is the code page of the federated database. On UNIX systems that use a Unicode federated database, the default is 1208. On Windows systems that use a Unicode federated database, the default is 1202.
COLLATING_SEQUENCE	Specifies whether the data source uses the same default collating sequence as the federated database. Valid values are Y, N, and I. I specifies a case-insensitive. The default is Y. The collating sequence specified for the federated server must match the collating sequence on the remote data source.
COMM_RATE	Specifies how much faster or slower the data source CPU is when compared to federated server CPU. Valid values are greater than 0 and less than $1x10^{23}$. The default is 1.0. Values can be expressed in any valid double notation, for example, 123E10, 123, or 1.21E4. A setting of 1 indicates that the federated server and the data source server have the same CPU speed; a 1:1 ratio. A setting of 0.5 indicates that the federated server CPU speed is 50% slower than the data source CPU. A setting of 2 indicates that the federated CPU is twice as fast as the data source CPU.

Table 50. Server options for Microsoft SQL Server (continued)

Name	Description
CPU_RATIO	Specifies how much faster or slower the CPU of the data source runs when compared to the speed of the federated server's CPU. Valid values are greater than 0 and less than $1x10^{23}$. The default is 1.0. Values can be expressed in any valid double notation, for example, 123E10, 123, or 1.21E4.
DBNAME	Required. Specifies the alias for the database that you want to access. The value is case-sensitive.
DB2_MAXIMAL_PUSHDOWN	Specifies the primary criteria that the query optimizer uses to choose an access plan. Valid values are Y and N. The default is N; the query optimizer chooses the plan that has the lowest estimated cost. Y specifies that the query optimizer choose the access plan that pushes down the most query operations to the data source. If more than one access plan meets this criteria, plan that has the lowest cost is chosen.
DB2_MAX_ASYNC_REQUESTS_PER_ QUERY	Specifies the maximum number of concurrent asynchronous requests from a query. Valid values are from -1 to 64000. The default is 1. -1 specifies that the federated query optimizer determines the number of requests. 0 specifies that the data source cannot accommodate additional asynchronous requests.
DB2_TWO_PHASE_COMMIT	Specifies whether the federated server connects to the data source in two-phase commit protocol or one-phase commit protocol. Valid values are Y and N. The default is N; the federated server uses the one-phase commit protocol to connect. Y specifies that the federated server uses two-phase commit protocol to connect. **Important:** If you set this option to Y, you must also specify the XA_OPEN_STRING_OPTION.
DB2_UM_PLUGIN	Specifies the implementation of the user mapping plug-in. For a plug-in written in Java, specifies a case-sensitive string for the class name that corresponds to the user mapping repository class. For example, `"UserMappingRepositoryLDAP"`. For a plug-in written in C, specifies any valid C library name.
DB2_UM_PLUGIN_LANG	Specifies the language of the user mapping plug-in. Valid values are Java and C. The default is Java.

Table 50. Server options for Microsoft SQL Server (continued)

Name	Description
FOLD_ID	Specifies the case for the user ID that is sent to the data source. There is no default value; the federated server sends the user ID in uppercase; then if the uppercase user ID fails, the server sends the user ID in lowercase. Valid values are U (uppercase), L (lowercase), and N (null). Avoid using the null setting, which might result in poor performance.
FOLD_PW	Specifies the case for the password that is sent to the data source. There is no default value; the federated server sends the password in uppercase; then if the uppercase password fails, the server sends the password in lowercase. Valid values are U (uppercase), L (lowercase), and N (null). Avoid using the null setting, which might result in poor performance.
IO_RATIO	Specifies how much faster or slower the data source I/O system runs when compared to the federated server I/O system. Valid values are greater than 0 and less than $1x10^{23}$. The default is 1.0. Values can be expressed in any valid double notation, for example, 123E10, 123, or 1.21E4. A setting of 1 indicates that the federated server and the data source server have the same I/O speed; a 1:1 ratio. A setting of 0.5 indicates that the federated server speed is 50% slower than the data source speed. A setting of 2 indicates that the federated speed is twice as fast as the data source speed.
NODE	Required. If the federated server uses Microsoft Windows, the value for NODE is the system DSN name that you specify for the Microsoft SQL Server. If the federated server uses UNIX or Linux, the value for NODE is defined in the `odbc.ini` file. The value is case-sensitive.
OLD_NAME_GEN	Specifies how to convert the column names and index names that are in the data source into nickname column names and local index names for the federated server. Valid values are Y and N. The default is N; the generated names closely match the names in the data source. Y specifies that the generated names are the same as the names that were created in IBM WebSphere Federation Server Version 9 and earlier. Thus, the names might not closely match the data source names.
PASSWORD	Specifies whether or not passwords are sent to a data source. The default is Y.

Table 50. Server options for Microsoft SQL Server (continued)

Name	Description
PUSHDOWN	Specifies whether the federated server allows the data source to evaluate operations. Valid values are Y and N. The default is Y; the data source evaluates operations. N specifies that the federated server send SQL statements that include only SELECT with column names. Predicates, such as WHERE=; column and scalar functions, such as MAX and MIN; sorts, such as ORDER BY OR GROUP BY; and joins are not included in any SQL that the federated server sends to the data source.
XA_OPEN_STRING_OPTIONS	Required when DB2_TWO_PHASE_COMMIT is set to Y. Specifies the resource manager ID of the Microsoft SQL Server Registry.

User mapping options

Table 51. User mapping options for Microsoft SQL Server

Name	Description
REMOTE_AUTHID	Specifies the remote user ID to which the local user ID is mapped. If you do not specify this option, the ID that is used to connect to the federated database is used.
REMOTE_PASSWORD	Specifies the remote password for the remote user ID. If you do not specify this option, the password that is used to connect to the federated database is used.

Column options

Table 52. Column options for Microsoft SQL Server

Name	Description
BINARY_REP	Identifies specific binary data type columns. You can set the value of the BINARY_REP column option to Y to force the federated server to push down the SQL_BINARY data type. Enabling the query optimizer to perform the comparison of binary type columns at the remote data source can improve query performance.

Table 52. Column options for Microsoft SQL Server (continued)

Name	Description
NUMERIC_STRING	Specifies how to treat numeric strings. The default is N. If the data source string column contains only numeric strings and no other characters, including blanks, set the NUMERIC_STRING option to Y. When NUMERIC_STRING is set to Y for a column, the query optimizer recognizes that the column contains no blanks that could interfere with the sorting of the data in the column. Use this option when the collating sequence of a data source is different from the collating sequence that the federated server uses. Columns that use this option are not excluded from remote evaluation because of a different collating sequence.

ODBC options reference

To configure how the federated server and its users interact with a data source, set and modify wrapper, server, user mapping, and column options.

Wrapper options

The following tables list the options that apply to this data source and identify the required options that you must specify.

Table 53. Wrapper options for ODBC

Name	Description
DB2_FENCED	Required. Specifies whether the wrapper runs in fenced mode or in trusted mode. Valid values are Y and N. The default is N; the wrapper runs in trusted mode. **Important:** If you set this option to Y for a UNIX system, you must also set the DB2_SOURCE_CLIENT_MODE wrapper option.
DB2_SOURCE_CLIENT_MODE	Specifies that the client for the data source is 32-bit and that the database instance on the federated server is 64-bit. The only valid value is 32-bit. This option is valid only for UNIX. **Important:** If you set this option, you must also set the DB2_FENCED wrapper option to Y.
DB2_UM_PLUGIN	Specifies the implementation of the user mapping plug-in. For a plug-in written in Java, specifies a case-sensitive string for the class name that corresponds to the user mapping repository class. For example, `"UserMappingRepositoryLDAP"`. For a plug-in written in C, specifies any valid C library name.

Table 53. Wrapper options for ODBC (continued)

Name	Description
DB2_UM_PLUGIN_LANG	Specifies the language of the user mapping plug-in. Valid values are Java and C. The default is Java.
MODULE	Required for federated servers that run on a UNIX system. Specifies the full path of the library that contains the ODBC Driver Manager implementation or the SQL/CLI implementation. There is no default for UNIX. On a Microsoft Windows system, the default is `odbc32.dll`.

Server options

Table 54. Server options for ODBC

Name	Description
CODEPAGE	Specifies the code page that corresponds to the coded character set of the client configuration for the data source. On UNIX and Windows systems that use a non-Unicode federated database, the default is the code page that the federated database uses. On UNIX systems that use a Unicode federated database, the default is 1208. On Windows systems that use a Unicode federated database, the default is 1202.
COLLATING_SEQUENCE	Specifies whether the data source uses the same default collating sequence as the federated database. Valid values are Y, N, and I. I specifies a case-insensitive. The default is Y. The collating sequence specified for the federated server must match the collating sequence on the remote data source.
COMM_RATE	Specifies the communication rate, in megabytes per second, between the federated server and the data source server. Valid values are whole numbers the are greater than 0 and less than 2147483648. The default is 2.
CPU_RATIO	Specifies how much faster or slower the data source CPU is when compared to federated server CPU. Valid values are greater than 0 and less than $1x10^{23}$. The default is 1.0. Values can be expressed in any valid double notation, for example, 123E10, 123, or 1.21E4. A setting of 1 indicates that the federated server and the data source server have the same CPU speed; a 1:1 ratio. A setting of 0.5 indicates that the federated server CPU speed is 50% slower than the data source CPU. A setting of 2 indicates that the federated CPU is twice as fast as the data source CPU.

Table 54. Server options for ODBC (continued)

Name	Description
DATEFORMAT	Specifies the date format that the data source uses. Use 'DD', 'MM', and 'YY' or 'YYYY' to specify the date format. You can specify a delimiter such as a space, a hyphen, or a comma. For example, the format `'YYYY-MM-DD'` specifies a date such as 1958-10-01. The value can contain null values. If you specify a value with single quotes, you need to preserve the single quotes. Specify additional single quotes, `''`, to preserve the single quotes. These are two additional single quotes, not a double quote. For example: `'''YYYY-MM-DD'''`.
DBNAME	Specifies the name of the data source database that you want to access.
DB2_AUTHID_QUOTE_CHAR	Specifies the quote characters that are used for authid names such as schema and user names. If you do not specify this option, double quotes are used by default.
DB2_ID_QUOTE_CHAR	Specifies the quote characters that are used for delimited identifiers such as column names. If you do not specify this option, double quotes are used by default.
DB2_MAX_ASYNC_REQUESTS_PER_ QUERY	Specifies the maximum number of concurrent asynchronous requests from a query. Valid values are from -1 to 64000. The default is 0. -1 specifies that the federated query optimizer determines the number of requests. 0 specifies that the data source cannot accommodate additional asynchronous requests.
DB2_MAXIMAL_PUSHDOWN	Specifies the primary criteria that the query optimizer uses to choose an access plan. Valid values are Y and N. The default is N; the query optimizer chooses the plan that has the lowest estimated cost. Y specifies that the query optimizer choose the access plan that pushes down the most query operations to the data source.
DB2_TABLE_QUOTE_CHAR	Specifies the quote characters that are used for table names. If you do not specify this option, double quotes are used by default.
DB2_UM_PLUGIN	Specifies the implementation of the user mapping plug-in. For a plug-in written in Java, specifies a case-sensitive string for the class name that corresponds to the user mapping repository class. For example, `"UserMappingRepositoryLDAP"`. For a plug-in written in C, specifies any valid C library name.

Table 54. Server options for ODBC (continued)

Name	Description
DB2_UM_PLUGIN_LANG	Specifies the language of the user mapping plug-in. Valid values are Java and C. The default is Java.
ENABLE_BULK_INSERT	Specifies whether bulk insert processing is enabled for the Netezza data source. Valid values are Y and N. The default is N.
FOLD_ID	Specifies the case for the user ID that is sent to the data source. There is no default value; the federated server sends the user ID in uppercase; then if the uppercase user ID fails, the server sends the user ID in lowercase. Valid values are U (uppercase), L (lowercase), and N (null). Avoid using the null setting, which might result in poor performance.
FOLD_PW	Specifies the case for the password that is sent to the data source. There is no default value; the federated server sends the password in uppercase; then if the uppercase password fails, the server sends the password in lowercase. Valid values are U (uppercase), L (lowercase), and N (null). Avoid using the null setting, which might result in poor performance.
IO_RATIO	Specifies how much faster or slower the data source I/O system runs when compared to the federated server I/O system. Valid values are greater than 0 and less than $1x10^{23}$. The default is 1.0. Values can be expressed in any valid double notation, for example, 123E10, 123, or 1.21E4. A setting of 1 indicates that the federated server and the data source server have the same I/O speed; a 1:1 ratio. A setting of 0.5 indicates that the federated server speed is 50% slower than the data source speed. A setting of 2 indicates that the federated speed is twice as fast as the data source speed.
NODE	Required. Specifies the name of the node or the system DSN name that is assigned to the ODBC data source when the DSN is defined. The value is case-sensitive.
OLD_NAME_GEN	Specifies how to convert the column names and index names that are in the data source into nickname column names and local index names for the federated server. Valid values are Y and N. The default is N; the generated names closely match the names in the data source. Y specifies that the generated names are the same as the names that were created in IBM WebSphere Federation Server Version 9 and earlier. Thus, the names might not closely match the data source names.

Table 54. Server options for ODBC (continued)

Name	Description
PUSHDOWN	Specifies whether the federated server allows the data source to evaluate operations. Valid values are Y and N. The default is Y; the data source evaluates operations. N specifies that the federated server send SQL statements that include only SELECT with column names. Predicates, such as WHERE=; column and scalar functions, such as MAX and MIN; sorts, such as ORDER BY OR GROUP BY; and joins are not included in any SQL that the federated server sends to the data source.
TIMEFORMAT	Specifies the time format that the data source uses. Use 'hh12', 'hh24', 'mm', 'ss', 'AM', and 'A.M.' to specify the time format. For example, the format `'hh24:mm:22'` specifies a time such as 16:00:00. The format `'hh12:mm:ss AM'` specifies a time such as 8:00:00 AM. The value can contain null values.
TIMESTAMPFORMAT	Specifies the timestamp format that the data source uses. Valid values are in the format that the DATEFORMAT option and the TIMEFORMAT option use. Specify 'n' for a tenth of a second, 'nn' for a hundredth of a second, 'nnn' for milliseconds, and so on, up to 'nnnnnn' for microseconds. For example, the format `'YYY-MM-DD-hh24:mm:ss.nnnnnn'` specifies a timestamp such as 1994-01-01-24:00:00.000000. The value can contain null values.
VARCHAR_NO_TRAILING_BLANKS	Specifies whether the data source contains VARCHAR columns that contain at least one trailing blank character. The default is N; VARCHAR columns contain at least one trailing blank character.

User mapping options

Table 55. User mapping options for ODBC

Name	Description
REMOTE_AUTHID	Specifies the remote user ID to which the local user ID is mapped. If you do not specify this option, the ID that is used to connect to the federated database is used.
REMOTE_PASSWORD	Specifies the remote password for the remote user ID. If you do not specify this option, the password that is used to connect to the federated database is used.

Column options

Table 56. Column options for ODBC

Name	Description
NUMERIC_STRING	Specifies whether the column contains strings of numeric characters that include blanks. Valid values are Y and N. The default is N; the column does not contain numeric strings that include blanks. If the column contains only numeric strings followed by trailing blanks, do not specify Y. When NUMERIC_STRING is set to Y for a column, the query optimizer recognizes that the column contains no blanks that could interfere with the sorting of the data in the column. Use this option when the collating sequence of a data source is different from the collating sequence that the federated server uses. Columns that use this option are not excluded from remote evaluation because of a different collating sequence.
VARCHAR_NO_TRAILING_BLANKS	Specifies whether there is at least one trailing blank in the VARCHAR column.

Oracle options reference

To configure how the federated server and its users interact with a data source, set and modify wrapper, server, user mapping, and column options.

Wrapper options

The following tables list the options that apply to this data source and identify the required options that you must specify.

Table 57. Wrapper options for Oracle

Name	Description
DB2_FENCED	Required. Specifies whether the wrapper runs in fenced mode or in trusted mode. Valid values are Y and N. The default is N; the wrapper runs in trusted mode.
DB2_UM_PLUGIN	Specifies the implementation of the user mapping plug-in. For a plug-in written in Java, specifies a case-sensitive string for the class name that corresponds to the user mapping repository class. For example, `"UserMappingRepositoryLDAP"`. For a plug-in written in C, specifies any valid C library name.
DB2_UM_PLUGIN_LANG	Specifies the language of the user mapping plug-in. Valid values are Java and C. The default is Java.

Server options

Table 58. Server options for Oracle

Name	Description
COLLATING_SEQUENCE	Specifies whether the data source uses the same default collating sequence as the federated database. Valid values are Y, N, and I. I specifies a case-insensitive. The default is Y. The collating sequence specified for the federated server must match the collating sequence on the remote data source.
COMM_RATE	Specifies the communication rate, in megabytes per second, between the federated server and the data source server. Valid values are whole numbers the are greater than 0 and less than 2147483648. The default is 2.
CPU_RATIO	Specifies how much faster or slower the data source CPU is when compared to federated server CPU. Valid values are greater than 0 and less than $1x10^{23}$. The default is 1.0. Values can be expressed in any valid double notation, for example, 123E10, 123, or 1.21E4. A setting of 1 indicates that the federated server and the data source server have the same CPU speed; a 1:1 ratio. A setting of 0.5 indicates that the federated server CPU speed is 50% slower than the data source CPU. A setting of 2 indicates that the federated CPU is twice as fast as the data source CPU.
DB2_MAXIMAL_PUSHDOWN	Specifies the primary criteria that the query optimizer uses to choose an access plan. Valid values are Y and N. The default is N; the query optimizer chooses the plan that has the lowest estimated cost. Y specifies that the query optimizer choose the access plan that pushes down the most query operations to the data source.
DB2_MAX_ASYNC_REQUESTS_PER_QUERY	Specifies the maximum number of concurrent asynchronous requests from a query. Valid values are from -1 to 64000. The default is 1. -1 specifies that the federated query optimizer determines the number of requests. 0 specifies that the data source cannot accommodate additional asynchronous requests.
DB2_TWO_PHASE_COMMIT	Specifies whether the federated server connects to the data source in two-phase commit protocol or one-phase commit protocol. Valid values are Y and N. The default is N; the federated server uses the one-phase commit protocol to connect. Y specifies that the federated server uses two-phase commit protocol to connect.

Table 58. Server options for Oracle (continued)

Name	Description
DB2_UM_PLUGIN	Specifies the implementation of the user mapping plug-in. For a plug-in written in Java, specifies a case-sensitive string for the class name that corresponds to the user mapping repository class. For example, `"UserMappingRepositoryLDAP"`. For a plug-in written in C, specifies any valid C library name.
DB2_UM_PLUGIN_LANG	Specifies the language of the user mapping plug-in. Valid values are Java and C. The default is Java.
FED_PROXY_USER	Specifies the authorization ID to use to establish all outbound trusted connections when the inbound connection is non-trusted. **Important:** The user whose ID is specified in this option must have a user mapping that specifies both a REMOTE_AUTHID and a REMOTE_PASSWORD.
FOLD_ID	Specifies the case for the user ID that is sent to the data source. There is no default value; the federated server sends the user ID in uppercase; then if the uppercase user ID fails, the server sends the user ID in lowercase. Valid values are U (uppercase), L (lowercase), and N (null). Avoid using the null setting, which might result in poor performance.
FOLD_PW	Specifies the case for the password that is sent to the data source. There is no default value; the federated server sends the password in uppercase; then if the uppercase password fails, the server sends the password in lowercase. Valid values are U (uppercase), L (lowercase), and N (null). Avoid using the null setting, which might result in poor performance.
IO_RATIO	Specifies how much faster or slower the data source I/O system runs when compared to the federated server I/O system. Valid values are greater than 0 and less than $1x10^{23}$. The default is 1.0. Values can be expressed in any valid double notation, for example, 123E10, 123, or 1.21E4. A setting of 1 indicates that the federated server and the data source server have the same I/O speed; a 1:1 ratio. A setting of 0.5 indicates that the federated server speed is 50% slower than the data source speed. A setting of 2 indicates that the federated speed is twice as fast as the data source speed.
NODE	Required. Specifies the name of the node where the Oracle database server resides. Obtain the name of the node from the `tnsnames.ora` file.

Table 58. Server options for Oracle (continued)

Name	Description
OLD_NAME_GEN	Specifies how to convert the column names and index names that are in the data source into nickname column names and local index names for the federated server. Valid values are Y and N. The default is N; the generated names closely match the names in the data source. Y specifies that the generated names are the same as the names that were created in IBM WebSphere Federation Server Version 9 and earlier. Thus, the names might not closely match the data source names.
PLAN_HINTS	Specifies whether or not plan hints are to be enabled. Plan hints are statement fragments that provide additional information that the data source optimizer uses to improve query performance. The data source optimizer uses the plan hints to decide whether or not to use an index and which index or which table join sequence to use. Valid values are Y and N. The default is N; plan hints are not to be enabled. Y specifies that plan hints are to be enabled at the data source if it supports plan hints.
PUSHDOWN	Specifies whether the federated server allows the data source to evaluate operations. Valid values are Y and N. The default is Y; the federated server allows the data source to evaluate operations. N specifies that the federated server retrieves columns from the remote data source.
VARCHAR_NO_TRAILING_BLANKS	Specifies for a particular server if VARCHAR columns contain any trailing blanks. To apply the option to a single column, use the VARCHAR_NO_TRAILING_BLANKS column option.
XA_OPEN_STRING_OPTIONS	Specifies additional information to append to the string. For example, the information might be the directory for trace files.

User mapping options

Table 59. User mapping options for Oracle

Name	Description
FED_PROXY_USER	Specifies the authorization ID to use to establish all outbound trusted connections when the inbound connection is not trusted. **Important:** The user whose ID is specified in this option must have a user mapping that specifies both a REMOTE_AUTHID and a REMOTE_PASSWORD. If you specify the FED_PROXY_USER user mapping option, you must also specify the FED_PROXY_USER server option.

Table 59. User mapping options for Oracle (continued)

Name	Description
REMOTE_AUTHID	Specifies the remote user ID to which the local user ID is mapped. If you do not specify this option, the ID that is used to connect to the federated database is used.
REMOTE_PASSWORD	Specifies the remote password for the remote user ID. If you do not specify this option, the password that is used to connect to the federated database is used.
USE_TRUSTED_CONTEXT	Specifies whether the user mapping is trusted. Valid values are Y and N. The default is N; the user mapping is not trusted and can be used only in non-trusted outbound connections. Y specifies that the user mapping is trusted and can be used in both trusted and non-trusted outbound connections.

Column options

Table 60. Column options for Oracle

Name	Description
NUMERIC_STRING	Specifies whether the column contains strings of numeric characters that include blanks. Valid values are Y and N. The default is N; the column does not contain numeric strings that include blanks. If the column contains only numeric strings followed by trailing blanks, do not specify Y. When NUMERIC_STRING is set to Y for a column, the query optimizer recognizes that the column contains no blanks that could interfere with the sorting of the data in the column. Use this option when the collating sequence of a data source is different from the collating sequence that the federated server uses. Columns that use this option are not excluded from remote evaluation because of a different collating sequence.
VARCHAR_NO_TRAILING_BLANKS	Specifies if the column contains any trailing blanks.

Script options reference

To configure how the federated server and its users interact with a data source, set and modify wrapper, server, user mapping, nickname, and column options.

Wrapper options

The following tables list the options that apply to this data source and identify the required options that you must specify.

Table 61. Wrapper options for scripts

Name	Description
DB2_FENCED	Required. Specifies whether the wrapper runs in fenced mode or in trusted mode. Valid values are Y and N. The default is N; the wrapper runs in trusted mode.
PROXY_TYPE	Specifies the proxy type to use to access the Internet when the federated server is behind a firewall. Valid values are NONE and SOCKS. The default value is NONE.
PROXY_SERVER_NAME	Specifies the name or IP address of the proxy server. Valid IP addresses are in IPv4 (dot-separated) format or in IPv6 (colon-separated) format. Use IPv6 format only if IPv6 is configured.
PROXY_SERVER_PORT	Specifies the port or service name for the proxy service on the proxy server. Valid values are a decimal port number from 1 to 32760 or a service name.

Server options

Table 62. Server options for scripts

Name	Description
DAEMON_PORT	Specifies the port number on which the Script daemon listens for Script job requests. The default is 4099. The port number must be the same as the DAEMON_PORT option in the daemon configuration file. If a service name is configured for the Script daemon, use a TCP/IP service name.
DB2_MAX_ASYNC_REQUESTS_PER_ QUERY	Specifies the maximum number of concurrent asynchronous requests from a query. Valid values are from -1 to 64000. The default is 1. -1 specifies that the federated query optimizer determines the number of requests. 0 specifies that the data source cannot accommodate additional asynchronous requests.
NODE	Required. Specifies the DNS host name or IP address of the system on which the Script daemon runs. Valid IP addresses are in IPv4 (dot-separated) format or in IPv6 (colon-separated) format. Use IPv6 format only if IPv6 is configured.
PROXY_AUTHID	Specifies the user name for proxy server authentication.
PROXY_PASSWORD	Specifies the password for proxy server authentication.
PROXY_SERVER_NAME	Specifies the name or IP address of the proxy server. Valid IP addresses are in IPv4 (dot-separated) format or in IPv6 (colon-separated) format. Use IPv6 format only if IPv6 is configured.

Table 62. Server options for scripts (continued)

Name	Description
PROXY_SERVER_PORT	Specifies the port or service name for the proxy service on the proxy server. Valid values are a decimal port number from 1 to 32760 or a service name.
PROXY_TYPE	Specifies the proxy type to use to access the Internet when the federated server is behind a firewall. Valid values are NONE and SOCKS. The default value is NONE.

User mapping options

Table 63. User mapping options for scripts

Name	Description
PROXY_AUTHID	Specifies the user name for proxy server authentication.
PROXY_PASSWORD	Specifies the password for proxy server authentication. The password is encrypted when it is stored in the federated database catalog.

Nickname options

Table 64. Nickname options for scripts

Name	Description
DATASOURCE	Required for the root nickname. Specifies the name of the script to invoke. The script that you specify as the value of this option must also be specified in the Script daemon configuration file. **Important:** This option valid only for the root nickname.

NAMESPACES	Specifies the namespaces that are associated with the namespace prefixes that are used in the XPATH and TEMPLATE options for each column. Use this syntax:
	`NAMESPACES'prefix1= "actual_namespace1", prefix2="actual_namespace2"'`
	
	Use a comma to separate multiple namespaces. For example:
	`NAMESPACES='http://www.myweb.com/cust", i='http://www.myweb.com/cust/id", n='http://www.myweb.com/cust/name"'`

Table 64. Nickname options for scripts (continued)

Name	Description
STREAMING	Specifies whether the source document should be separated into logical fragments for processing. The fragments correspond to the node that matches the XPath expression of the nickname. The wrapper then parses and processes source data fragment-by-fragment. This type of parsing minimizes memory use. Valid values are Y and N. The default is N; the documents are not parsed. This option is valid only on the root nickname. Do not set both the STREAMING and VALIDATE options to Y.
TIMEOUT	Specifies the maximum time, in minutes, to wait for results from the daemon. The default is 60. This option is valid only for the root nickname.
VALIDATE	Specifies whether the source document is validated to ensure that it conforms to an XML schema or document type definition (DTD) before data is extracted from it. The default is N; validation does not occur. Before you set the value to Y, the schema file or DTD file is in the location that the source document specifies. This option is valid only for the root nickname. Do not set both the STREAMING and VALIDATE options to Y.
XPATH	Specifies the XPath expression that identifies the XML elements that represent individual tuples. The XPATH nickname option for a child nickname is evaluated in the context of the path that is specified by the XPATH nickname option of its parent. This XPath expression is used as a context for evaluating column values that are identified by the presence of the XPATH column option.

Column options

Table 65. Column options for scripts

Name	Description
DEFAULT	Specifies a default value for a script input column. If the SQL query does not provide a value, this default value is used.

Table 65. Column options for scripts (continued)

Name	Description
FOREIGN_KEY	Indicates that this nickname is a child nickname and specifies the name of the corresponding parent nickname. A nickname can have at most one FOREIGN_KEY column option. The value for the option is case-sensitive. Do not specify the XPATH option for this column. The column can be used only to join a parent nickname and a child nickname. A CREATE NICKNAME statement that includes a FOREIGN_KEY option fails if the parent nickname has a different schema name. Unless the nickname that is referred to in a FOREIGN_KEY clause was explicitly defined as lowercase or mixed case in the CREATE NICKNAME statement, you must specify the nickname in uppercase when you refer to this nickname in the FOREIGN_KEY clause. When this option is set on a column, no other option can be set on the column.
INPUT_MODE	Specifies the input mode for the column. Valid values are CONFIG and FILE_INPUT. CONFIG treats the value as the input mode for a column. FILE_INPUT specifies a file that stores the value. The wrapper passes the specified value to the Script daemon.
POSITION	Specifies an integer value for positional parameters. If the positional value is set to an integer, then this input must be in this position in the command line. If this option is set, the switch is inserted into the appropriate location when the query runs. If POSITION is set to -1, the option is added as the last command line option. A POSITION integer value cannot be used twice in the same nickname. This option applies only to input columns.
PRIMARY_KEY	Required for a parent nickname that has one or more child nicknames. Specifies that this nickname is a parent nickname. The column data type must be VARCHAR(16). A nickname can have only one PRIMARY_KEY column option. Yes is the only valid value. Do not specify the XPATH option for this column. The column can be used only to join parent nicknames and child nicknames. When this option is set on a column, no other option can be set on the column.

Table 65. Column options for scripts (continued)

Name	Description
SWITCH	Specifies a flag for the script on the command line. The value of this option preceded the column value that is supplied by WSSCRIPT.ARGS or by the default value, if any. If you do not specify a value for this option and a default value exists for the column, the default value is added without any switch information. This option is required for input columns.
SWITCH_ONLY	Enables the use of switches without a command line argument. Valid values are Y and N. Set this option to Y and the valid input values are Y and N. For an input value of Y, only the switch is added to the command line. For an input value of N, no value is added to the command line.
VALID_VALUES	Specifies a set of valid values for a column. Use a semicolon to separate multiple values.
XPATH	Specifies the Xpath expression in the XML document that contains the data that corresponds to this column. The wrapper evaluates this XPath expression after the CREATE NICKNAME statement applies the XPath expression from the XPATH nickname option.

Sybase options reference

To configure how the federated server and its users interact with a data source, set and modify wrapper, server, user mapping, and column options.

Wrapper options

The following tables list the options that apply to this data source and identify the required options that you must specify.

Table 66. Wrapper options for Sybase

Name	Description
DB2_FENCED	Required. Specifies whether the wrapper runs in fenced mode or in trusted mode. On Microsoft Windows, valid values are Y and N. The default is N; the wrapper runs in trusted mode. On UNIX, the default and the only valid value is Y; the wrapper must run in fenced mode.
DB2_UM_PLUGIN	Specifies the implementation of the user mapping plug-in. For a plug-in written in Java, specifies a case-sensitive string for the class name that corresponds to the user mapping repository class. For example, `"UserMappingRepositoryLDAP"`. For a plug-in written in C, specifies any valid C library name.

Table 66. Wrapper options for Sybase (continued)

Name	Description
DB2_UM_PLUGIN_LANG	Specifies the language of the user mapping plug-in. Valid values are Java and C. The default is Java.

Server options

Table 67. Server options for Sybase

Name	Description
COLLATING_SEQUENCE	Specifies whether the data source uses the same default collating sequence as the federated database. Valid values are Y, N, and I. I specifies a case-insensitive. The default is Y. The collating sequence specified for the federated server must match the collating sequence on the remote data source.
COMM_RATE	Specifies the communication rate, in megabytes per second, between the federated server and the data source server. Valid values are whole numbers the are greater than 0 and less than 2147483648. The default is 2.
CPU_RATIO	Specifies how much faster or slower the data source CPU is when compared to federated server CPU. Valid values are greater than 0 and less than $1x10^{23}$. The default is 1.0. Values can be expressed in any valid double notation, for example, 123E10, 123, or 1.21E4. A setting of 1 indicates that the federated server and the data source server have the same CPU speed; a 1:1 ratio. A setting of 0.5 indicates that the federated server CPU speed is 50% slower than the data source CPU. A setting of 2 indicates that the federated CPU is twice as fast as the data source CPU.
CONV_EMPTY_STRING	Specifies whether the federated server converts an empty string into a space during replication tasks. Valid values are Y and N. The default is N; the federated server does not convert empty strings. Set this option to Y when the data source has a non-nullable character column that stores an empty string.
DBNAME	Required. Specifies the name of the database that you want to access. Obtain the name of the database from the Sybase server.
DB2_ID_QUOTE_CHAR	Specifies the quote characters that are used for delimited identifiers such as column names. If you do not specify this option, double quotes are used by default.

Table 67. Server options for Sybase (continued)

Name	Description
DB2_MAXIMAL_PUSHDOWN	Specifies the primary criteria that the query optimizer uses to choose an access plan. Valid values are Y and N. The default is N; the query optimizer chooses the plan that has the lowest estimated cost. Y specifies that the query optimizer choose the access plan that pushes down the most query operations to the data source.
DB2_MAX_ASYNC_REQUESTS_PER_ QUERY	Specifies the maximum number of concurrent asynchronous requests from a query. Valid values are from -1 to 64000. The default is 1. -1 specifies that the federated query optimizer determines the number of requests. 0 specifies that the data source cannot accommodate additional asynchronous requests.
DB2_TWO_PHASE_COMMIT	Specifies whether the federated server connects to the data source in two-phase commit protocol or one-phase commit protocol. Valid values are Y and N. The default is N; the federated server uses the one-phase commit protocol to connect. Y specifies that the federated server uses two-phase commit protocol to connect.
DB2_UM_PLUGIN	Specifies the implementation of the user mapping plug-in. For a plug-in written in Java, specifies a case-sensitive string for the class name that corresponds to the user mapping repository class. For example, `"UserMappingRepositoryLDAP"`. For a plug-in written in C, specifies any valid C library name.
DB2_UM_PLUGIN_LANG	Specifies the language of the user mapping plug-in. Valid values are Java and C. The default is Java.
FOLD_ID	Specifies the case for the user ID that is sent to the data source. There is no default value; the federated server sends the user ID in uppercase; then if the uppercase user ID fails, the server sends the user ID in lowercase. Valid values are U (uppercase), L (lowercase), and N (null). Avoid using the null setting, which might result in poor performance.
FOLD_PW	Specifies the case for the password that is sent to the data source. There is no default value; the federated server sends the password in uppercase; then if the uppercase password fails, the server sends the password in lowercase. Valid values are U (uppercase), L (lowercase), and N (null). Avoid using the null setting, which might result in poor performance.

Table 67. Server options for Sybase (continued)

Name	Description
IFILE	Specifies the path and name of the Sybase interface file to use instead of the default interface file. The Sybase wrapper searches for the interface file in the following places, in the order specified: On Microsoft Windows, in the IFILE server option, then in the %DB2PATH%\interfaces directory, and finally in the %SYBASE%\ini\sql.ini directory. On UNIX, in the IFILE server option, then in the sqllib/interfaces directory, and finally in the $SYBASE/interfaces directory.
IO_RATIO	Specifies how much faster or slower the data source I/O system runs when compared to the federated server I/O system. Valid values are greater than 0 and less than $1x10^{23}$. The default is 1.0. Values can be expressed in any valid double notation, for example, 123E10, 123, or 1.21E4. A setting of 1 indicates that the federated server and the data source server have the same I/O speed; a 1:1 ratio. A setting of 0.5 indicates that the federated server speed is 50% slower than the data source speed. A setting of 2 indicates that the federated speed is twice as fast as the data source speed.
LOGIN_TIMEOUT	Specifies the amount of time, in seconds, that the federated server waits before abandoning a login request. The default is 0; the federated server waits an unlimited amount of time.
NODE	Required. Specifies the name of the node where the Sybase server resides. The node name is in the Sybase interfaces file.
OLD_NAME_GEN	Specifies how to convert the column names and index names that are in the data source into nickname column names and local index names for the federated server. Valid values are Y and N. The default is N; the generated names closely match the names in the data source. Y specifies that the generated names are the same as the names that were created in IBM WebSphere Federation Server Version 9 and earlier. Thus, the names might not closely match the data source names.
PACKET_SIZE	Specifies the packet size, in bytes, that the client library uses to send tabular data stream (TDS) packets. If the Sybase wrapper needs to send or receive large amounts of text and image data, increase the PACKET_SIZE.

Table 67. Server options for Sybase (continued)

Name	Description
PLAN_HINTS	Specifies whether or not plan hints are to be enabled. Plan hints are statement fragments that provide additional information that the data source optimizer uses to improve query performance. The data source optimizer uses the plan hints to decide whether or not to use an index and which index or which table join sequence to use. Valid values are Y and N. The default is N; plan hints are not to be enabled. Y specifies that plan hints are to be enabled at the data source if it supports plan hints.
PUSHDOWN	Specifies whether the federated server allows the data source to evaluate operations. Valid values are Y and N. The default is Y; the data source evaluates operations. N specifies that the federated server send SQL statements that include only SELECT with column names. Predicates, such as WHERE=; column and scalar functions, such as MAX and MIN; sorts, such as ORDER BY OR GROUP BY; and joins are not included in any SQL that the federated server sends to the data source.
SERVER_PRINCIPAL_NAME	Specifies the principal name of the Sybase server when the principal name is different from node name. The default value is the node name.
TIMEOUT	Specifies the maximum time, in seconds, that the federated server waits for the remote server to respond to a command. The default is 0, which specifies an unlimited amount of time.
XA_OPEN_STRING_OPTIONS	Specifies open strings for the Sybase DTM XA interface. These strings are in addition to the LRM name, user name, and password.

User mapping options

Table 68. User mapping options for Sybase

Option	Description
ENABLE_KERBEROS_CONNECTION	Specifies the Kerberos security mechanism. The default value N indicates use of a default security mechanism. The value Y indicates that the Kerberos security mechanism is enabled.
REMOTE_AUTHID	Specifies the remote user ID to which the local user ID is mapped. If you do not specify this option, the ID that is used to connect to the federated database is used.

Table 68. User mapping options for Sybase (continued)

Option	Description
REMOTE_PASSWORD	Specifies the remote password for the remote user ID. If you do not specify this option, the password that is used to connect to the federated database is used.

Column options

Table 69. Column options for Sybase

Option	Description
NUMERIC_STRING	Specifies how to treat numeric strings. The default is N. If the data source string column contains only numeric strings and no other characters, including blanks, set the NUMERIC_STRING option to Y. When NUMERIC_STRING is set to Y for a column, the query optimizer recognizes that the column contains no blanks that could interfere with the sorting of the data in the column. Use this option when the collating sequence of a data source is different from the collating sequence that the federated server uses. Columns that use this option are not excluded from remote evaluation because of a different collating sequence.

Teradata options reference

To configure how the federated server and its users interact with a data source, set and modify wrapper, server, user mapping, and column options.

Wrapper options

The following tables list the options that apply to this data source and identify the required options that you must specify.

Table 70. Wrapper options for Teradata

Name	Description
DB2_FENCED	Required. Specifies whether the wrapper runs in fenced mode or in trusted mode. On Microsoft Windows, valid values are Y and N. The default is N; the wrapper runs in trusted mode. On UNIX, the default value is Y; the wrapper must run in fenced mode.
DB2_UM_PLUGIN	Specifies the implementation of the user mapping plug-in. For a plug-in written in Java, specifies a case-sensitive string for the class name that corresponds to the user mapping repository class. For example, `"UserMappingRepositoryLDAP"`. For a plug-in written in C, specifies any valid C library name.

Table 70. Wrapper options for Teradata (continued)

Name	Description
DB2_UM_PLUGIN_LANG	Specifies the language of the user mapping plug-in. Valid values are Java and C. The default is Java.

Server options

Table 71. Server options for Teradata

Name	Description
COLLATING_SEQUENCE	Specifies whether the data source uses the same default collating sequence as the federated database. Valid values are Y, N, and I. I specifies a case-insensitive. The default is Y. The collating sequence specified for the federated server must match the collating sequence on the remote data source.
COMM_RATE	Specifies the communication rate, in megabytes per second, between the federated server and the data source server. Valid values are whole numbers the are greater than 0 and less than 2147483648. The default is 2.
CPU_RATIO	Specifies how much faster or slower the data source CPU is when compared to federated server CPU. Valid values are greater than 0 and less than $1x10^{23}$. The default is 1.0. Values can be expressed in any valid double notation, for example, 123E10, 123, or 1.21E4. A setting of 1 indicates that the federated server and the data source server have the same CPU speed; a 1:1 ratio. A setting of 0.5 indicates that the federated server CPU speed is 50% slower than the data source CPU. A setting of 2 indicates that the federated CPU is twice as fast as the data source CPU.
DB2_MAXIMAL_PUSHDOWN	Specifies the primary criteria that the query optimizer uses to choose an access plan. Valid values are Y and N. The default is N; the query optimizer chooses the plan that has the lowest estimated cost. Y specifies that the query optimizer choose the access plan that pushes down the most query operations to the data source.
DB2_MAX_ASYNC_REQUESTS_PER_ QUERY	Specifies the maximum number of concurrent asynchronous requests from a query. Valid values are from -1 to 64000. The default is 1. -1 specifies that the federated query optimizer determines the number of requests. 0 specifies that the data source cannot accommodate additional asynchronous requests.

Table 71. Server options for Teradata (continued)

Name	Description
DB2_UM_PLUGIN	Specifies the implementation of the user mapping plug-in. For a plug-in written in Java, specifies a case-sensitive string for the class name that corresponds to the user mapping repository class. For example, `"UserMappingRepositoryLDAP"`. For a plug-in written in C, specifies any valid C library name.
DB2_UM_PLUGIN_LANG	Specifies the language of the user mapping plug-in. Valid values are Java and C. The default is Java.
IO_RATIO	Specifies how much faster or slower the data source I/O system runs when compared to the federated server I/O system. Valid values are greater than 0 and less than $1x10^{23}$. The default is 1.0. Values can be expressed in any valid double notation, for example, 123E10, 123, or 1.21E4. A setting of 1 indicates that the federated server and the data source server have the same I/O speed; a 1:1 ratio. A setting of 0.5 indicates that the federated server speed is 50% slower than the data source speed. A setting of 2 indicates that the federated speed is twice as fast as the data source speed.
NODE	Required. Specifies the alias name or IP address of the Teradata server.
OLD_NAME_GEN	Specifies how to convert the column names and index names that are in the data source into nickname column names and local index names for the federated server. Valid values are Y and N. The default is N; the generated names closely match the names in the data source. Y specifies that the generated names are the same as the names that were created in IBM WebSphere Federation Server Version 9 and earlier. Thus, the names might not closely match the data source names.
PUSHDOWN	Specifies whether the federated server allows the data source to evaluate operations. Valid values are Y and N. The default is Y; the data source evaluates operations. N specifies that the federated server send SQL statements that include only SELECT with column names. Predicates, such as WHERE=; column and scalar functions, such as MAX and MIN; sorts, such as ORDER BY OR GROUP BY; and joins are not included in any SQL that the federated server sends to the data source.

User mapping options

Table 72. User mapping options for Teradata

Name	Description
REMOTE_AUTHID	Specifies the remote user ID to which the local user ID is mapped. If you do not specify this option, the ID that is used to connect to the federated database is used.
REMOTE_PASSWORD	Specifies the remote password for the remote user ID. If you do not specify this option, the password that is used to connect to the federated database is used.

Column options

Table 73. Column options for Teradata

Name	Description
NUMERIC_STRING	Specifies how to treat numeric strings. The default is N. If the data source string column contains only numeric strings and no other characters, including blanks, set the NUMERIC_STRING option to Y. When NUMERIC_STRING is set to Y for a column, the query optimizer recognizes that the column contains no blanks that could interfere with the sorting of the data in the column. Use this option when the collating sequence of a data source is different from the collating sequence that the federated server uses. Columns that use this option are not excluded from remote evaluation because of a different collating sequence.

Table-structured file options reference

To configure how the federated server and its users interact with a data source, set and modify wrapper, server, nickname, and column options.

The following tables list the options that apply to this data source and identify the required options that you must specify.

Wrapper options

Table 74. Wrapper options for table-structured files

Name	Description
DB2_FENCED	Required. Specifies whether the wrapper runs in fenced mode or in trusted mode. Valid values are Y and N. The default is N; the wrapper runs in trusted mode.

Server options

Table 75. Server options for table-structured files

Name	Description
DB2_MAX_ASYNC_REQUESTS_PER_ QUERY	Specifies the maximum number of concurrent asynchronous requests from a query. Valid values are from -1 to 64000. The default is 1. -1 specifies that the federated query optimizer determines the number of requests. 0 specifies that the data source cannot accommodate additional asynchronous requests.

Nickname options

Table 76. Nickname options for table-structured files

Name	Description
COLUMN_DELIMITER	Specifies a single character to use as the delimiter that separates columns in the table-structured file. The default is a comma (,). A single quotation mark cannot be used as a delimiter. The column delimiter must be consistent throughout the file. A null value is represented by two delimiters next to each other or by a delimiter followed by a line terminator, if the NULL field is the last one on the line. The column delimiter cannot exist as valid data for a column.
CODEPAGE	Specifies the code page of the file at the data source. This option is valid only for federated databases that use Unicode. The source data is converted from the specified code page to Unicode.
FILE_PATH	Specifies the fully qualified path to the table-structured file. Enclose the file name in single quotation marks. The data file must be a standard file or a symbolic link, rather than a pipe or another non-standard file type. **Important:** If you specify the FILE_PATH option, do not specify a DOCUMENT column.
KEY_COLUMN	Specifies the name of the column on which the file is sorted. A column that has the DOCUMENT column option cannot be the key column. Only single-column keys are supported. The value must be the name of a column that is defined in the CREATE NICKNAME statement. The column must be sorted in ascending order. The key column must be designated as non-nullable by adding the NOT NULL option to its definition in the nickname statement. This value is case-sensitive.

Table 76. Nickname options for table-structured files (continued)

Name	Description
SORTED	Specifies whether the file at the data source is or is not sorted in ascending order. Valid values are Y and N. The default is N; the file at the data source is not sorted in ascending order. Sorted data sources must be sorted in ascending order according to the collation sequence for the current locale, as defined by the settings in the LC_COLLATE National Language Support category. If you specify that the data source is sorted, set the VALIDATE_DATA_FILE option to Y.
VALIDATE_DATA_FILE	For sorted files, this option specifies whether the wrapper verifies that the key column is sorted in ascending order and checks for null keys. This validation occurs only once when the nickname is first created. The default is N; sort order is not verified. This option is valid only when the SORTED option is set to Y and the DOCUMENT option is not specified.

Column options

Table 77. Column options for table-structured files

Option	Description
DOCUMENT	Allows you to specify the file path when the query runs, instead of when you create the nickname. The only valid value is FILE. Only one column of each nickname can be specified with the DOCUMENT option. The column that is associated with the DOCUMENT option must be a data type of VARCHAR or CHAR. Using the DOCUMENT nickname column option instead of the FILE_PATH nickname option implies that the file that corresponds to this nickname will be supplied when the query runs. If the DOCUMENT option has the FILE value, the value that is supplied when the query runs is the full path of the file whose schema matches the nickname definition for this nickname.

Web services options reference

To configure how the federated server and its users interact with a data source, set and modify wrapper, server, user mapping, nickname, and column options.

Wrapper options

The following tables list the options that apply to this data source and identify the required options that you must specify.

Table 78. Wrapper options for Web services

Name	Description
DB2_FENCED	Required. Specifies whether the wrapper runs in fenced mode or in trusted mode. Valid values are Y and N. The default is N; the wrapper runs in trusted mode.
DB2_UM_PLUGIN	Specifies the implementation of the user mapping plug-in. For a plug-in written in Java, specifies a case-sensitive string for the class name that corresponds to the user mapping repository class. For example, `"UserMappingRepositoryLDAP"`. For a plug-in written in C, specifies any valid C library name.
DB2_UM_PLUGIN_LANG	Specifies the language of the user mapping plug-in. Valid values are Java and C. The default is Java.
PROXY_TYPE	Specifies the proxy type to use to access the Internet when the federated server is behind a firewall. Valid values are NONE, HTTP, and SOCKS. The default value is NONE.
PROXY_SERVER_NAME	Specifies the name or IP address of the proxy server. Valid IP addresses are in IPv4 (dot-separated) format or in IPv6 (colon-separated) format. Use IPv6 format only if IPv6 is configured.
PROXY_SERVER_PORT	Specifies the port or service name for the proxy service on the proxy server. Valid values are a decimal port number from 1 to 32760 or a service name.
SSL_KEYSTORE_FILE	Specifies the certificate storage file for communications that use SSL or TSL. A valid value is a fully qualified path name that is accessible by the federated database agent or by a fenced-mode process. The default is *install path*`/cfg/WSWrapperKeystore.kdb`.
SSL_KEYSTORE_PASSWORD	Specifies the password to use to access the file in the SSL_KEYSTORE_FILE option. Valid values are a password, which is encrypted when it is stored in the federated database catalog, and `file:`*file_name*, where *file_name* is the fully qualified path to a stash file.
SSL_VERIFY_SERVER_CERTIFICATE	Specifies whether the server certificate is verified during SSL authentication. Valid values are Y and N. The default is N; the certificate is not verified.

Server options

Table 79. Server options for Web services

Name	Description
DB2_MAX_ASYNC_REQUESTS_PER_ QUERY	Specifies the maximum number of concurrent asynchronous requests from a query. Valid values are from -1 to 64000. The default is 1. -1 specifies that the federated query optimizer determines the number of requests. 0 specifies that the data source cannot accommodate additional asynchronous requests.
DB2_UM_PLUGIN	Specifies the implementation of the user mapping plug-in. For a plug-in written in Java, specifies a case-sensitive string for the class name that corresponds to the user mapping repository class. For example, `"UserMappingRepositoryLDAP"`. For a plug-in written in C, specifies any valid C library name.
DB2_UM_PLUGIN_LANG	Specifies the language of the user mapping plug-in. Valid values are Java and C. The default is Java.
PROXY_AUTHID	Specifies the user name for proxy server authentication.
PROXY_PASSWORD	Specifies the password for proxy server authentication.
PROXY_SERVER_NAME	Specifies the name or IP address of the proxy server. Valid IP addresses are in IPv4 (dot-separated) format or in IPv6 (colon-separated) format. Use IPv6 format only if IPv6 is configured.
PROXY_SERVER_PORT	Specifies the port or service name for the proxy service on the proxy server. Valid values are a decimal port number from 1 to 32760 or a service name.
PROXY_TYPE	Specifies the proxy type to use to access the Internet when the federated server is behind a firewall. Valid values are NONE, HTTP, and SOCKS. The default value is NONE.
SSL_CLIENT_CERTIFICATE_LABEL	Specifies the client certificate to send during SSL authentication. If you do not specify a value, the current authorization ID for the federated database is used to locate the certificate.
SSL_KEYSTORE_FILE	Specifies the certificate storage file for communications that use SSL or TSL. A valid value is a fully qualified path name that is accessible by the federated database agent or by a fenced-mode process. The default is *install path*`/cfg/WSWrapperKeystore.kdb`.

Table 79. Server options for Web services (continued)

Name	Description
SSL_KEYSTORE_PASSWORD	Specifies the password to use to access the file in the SSL_KEYSTORE_FILE option. Valid values are a password, which is encrypted when it is stored in the federated database catalog, and `file:`*`file_name`*, where *file_name* is the fully qualified path to a stash file.
SSL_VERIFY_SERVER_CERTIFICATE	Specifies whether to verify the server certificate during SSL authentication. Valid values are Y and N. The default is N; the certificate is not verified.

User mapping options

Table 80. User mapping options for Web services

Name	Description
PROXY_AUTHID	Specifies the user name for proxy server authentication.
PROXY_PASSWORD	Specifies the password for proxy server authentication. The password is encrypted when it is stored in the federated database catalog.
REMOTE_AUTHID	Specifies the remote user ID to which the local user ID is mapped. If you do not specify this option, the ID that is used to connect to the federated database is used.
REMOTE_PASSWORD	Specifies the remote password for the remote user ID. If you do not specify this option, the password that is used to connect to the federated database is used.
SSL_CLIENT_CERTIFICATE_LABEL	Specifies the client certificate to send during SSL authentication. If you do not specify a value, the current federated database authorization ID is used to locate the certificate.

Nickname options

Table 81. Nickname options for Web services

Name	Description

NAMESPACES	Specifies the namespaces that are associated with the namespace prefixes that are used in the XPATH and TEMPLATE options for each column. Use this syntax:
	`NAMESPACES'prefix1= "actual_namespace1", prefix2="actual_namespace2"'`
	
	Use a comma to separate multiple namespaces. For example:
	`NAMESPACES='http://www.myweb.com/cust", i='http://www.myweb.com/cust/id", n='http://www.myweb.com/cust/name"'`

SOAPACTION	Required for the root nickname. Specifies the URI SOAPACTION attribute from the Web Services Description Language (WSDL) format. The URL can contain a colon-separated IPv6 address if it is enclosed in square brackets. For example:`http://[1080:0:0:0:8:800:200C:417A]` **Note:** This option is not valid for nonroot nicknames.
STREAMING	Specifies whether the source document should be separated into logical fragments for processing. The fragments correspond to the node that matches the XPath expression of the nickname. The wrapper then parses and processes source data fragment-by-fragment. This type of parsing minimizes memory use. Valid values are Y and N. The default is N; the documents are not parsed. This option is valid only on the root nickname.
TEMPLATE	Specifies the nickname template fragment to use to construct a SOAP request. The fragment must conform to the specified template syntax. This option is valid only for the root nickname.
URL	Required for the root nickname. Specifies the URL for the Web service endpoint. Supported protocols are HTTP and HTTPS. The URL can contain a colon-separated IPv6 address if it is enclosed in square brackets. For example:`http://[1080:0:0:0:8:800:200C:417A]`
XML_CODESET	Specifies the encoding to use to send and receive XML data. This option overrides the internal encoding.

Table 81. Nickname options for Web services (continued)

Name	Description
XPATH	Required. Specifies the Xpath expression that identifies the SOAP response elements that represent individual tuples. The Xpath expression is used as a context for evaluating column values that the XPATH nickname column options identify.

Column options

Table 82. Column options for Web services

Name	Description
ESCAPE_INPUT	Specifies whether XML special characters are replaced in XML input values or not. Use this option to include XML fragments as input, for example, to include XML fragments that have repeating elements. Valid values are Y and N. N is the default; XML input values are retained. The column data type must be VARCHAR or CHAR. If ESCAPE_INPUT is set to Y, you must also specify the TEMPLATE column option.
FOREIGN_KEY	Indicates that this nickname is a child nickname and specifies the name of the corresponding parent nickname. A nickname can have at most one FOREIGN_KEY column option. The value for the option is case-sensitive. Do not specify the XPATH option for this column. The column can be used only to join a parent nicknames and a child nickname. A CREATE NICKNAME statement that includes a FOREIGN_KEY option fails if the parent nickname has a different schema name. Unless the nickname that is referred to in a FOREIGN_KEY clause was explicitly defined as lowercase or mixed case in the CREATE NICKNAME statement, you must specify the nickname in uppercase when you refer to this nickname in the FOREIGN_KEY clause. **Note:** • When this option is set on a column, no other option can be set on the column. • If you set this column option, you cannot later use the ALTER NICKNAME statement to drop the option. Instead, you must drop the nickname and then create the nickname again without this column option.

Table 82. Column options for Web services (continued)

Name	Description
PRIMARY_KEY	Required for a parent nickname that has one or more child nicknames. Specifies that this nickname is a parent nickname. The column data type must be VARCHAR(16). A nickname can have only one PRIMARY_KEY column option. Yes is the only valid value. Do not specify the XPATH option for this column. The column can be used only to join parent nicknames and child nicknames. **Note:** • When this option is set on a column, no other option can be set on the column. • If you set this column option, you cannot later use the ALTER NICKNAME statement to drop the option. Instead, you must drop the nickname and then create the nickname again without this column option.
SOAPACTIONCOLUMN	Specifies the column that dynamically specifies the SOAP action for the Web Services endpoint when you run a query. This option is valid only for the root nickname. If the host name is an IPv6 (colon-separated) address, enclose the host name in square brackets. For example: `'http://[1080:0:0:0:8:800:200C:417A]:99/soap'` When this option is set on a column, no other option can be set on the column.
TEMPLATE	Specifies the column template fragment to use to construct the XML input document. The fragment must conform to the specified template syntax. **Note:** If you set this column option, you cannot later use the ALTER NICKNAME statement to drop the option. Instead, you must drop the nickname and then create the nickname again without this column option.
URLCOLUMN	Specifies the column that dynamically specifies the SOAP action for the Web Services endpoint when you run a query. This option is valid only for the root nickname. If the host name is an IPv6 (colon-separated) address, enclose the host name in square brackets. For example: `'http://[1080:0:0:0:8:800:200C:417A]:99/soap'` When this option is set on a column, no other option can be set on the column.

Table 82. Column options for Web services (continued)

Name	Description
XPATH	Specifies the Xpath expression in the XML document that contains the data that corresponds to this column. The wrapper evaluates this XPath expression after the CREATE NICKNAME statement applies the XPath expression from the XPATH nickname option. **Note:** If you set this column option, you cannot later use the ALTER NICKNAME statement to drop the option. Instead, you must drop the nickname and then create the nickname again without this column option.

XML options reference

To configure how the federated server and its users interact with a data source, set and modify wrapper, server, user mapping, nickname, and column options.

Wrapper options

The following tables list the options that apply to this data source and identify the required options that you must specify.

Table 83. Wrapper options for XML

Name	Description
DB2_FENCED	Required. Specifies whether the wrapper runs in fenced mode or in trusted mode. Valid values are Y and N. N is the default; the wrapper runs in trusted mode.
PROXY_TYPE	Specifies the proxy type to use to access the Internet when the federated server is behind a firewall. Valid values are NONE, HTTP, and SOCKS. The default value is NONE.
PROXY_SERVER_NAME	Specifies the name or IP address of the proxy server. Valid IP addresses are in IPv4 (dot-separated) format or in IPv6 (colon-separated) format. Use IPv6 format only if IPv6 is configured.
PROXY_SERVER_PORT	Specifies the port or service name for the proxy service on the proxy server. Valid values are a decimal port number from 1 to 32760 or a service name.
SSL_KEYSTORE_FILE	Specifies the certificate storage file for communications that use SSL or TSL. A valid value is a fully qualified path name that is accessible by the federated database agent or by a fenced-mode process.

Table 83. Wrapper options for XML (continued)

Name	Description
SSL_KEYSTORE_PASSWORD	Specifies the password to use to access the file in the SSL_KEYSTORE_FILE option. Valid values are a password, which is encrypted when it is stored in the federated database catalog, and `file:`*`file_name`*, where *file_name* is the fully qualified path to a stash file.
SSL_VERIFY_SERVER_CERTIFICATE	Specifies whether the server certificate is verified during SSL authentication. Valid values are Y and N. The default is N; the certificate is not verified.

Server options

Table 84. Server options for XML

Name	Description
DB2_MAX_ASYNC_REQUESTS_PER_ QUERY	Specifies the maximum number of concurrent asynchronous requests from a query. Valid values are from -1 to 64000. The default is 1. -1 specifies that the federated query optimizer determines the number of requests. 0 specifies that the data source cannot accommodate additional asynchronous requests.
PROXY_AUTHID	Specifies the user name for proxy server authentication.
PROXY_PASSWORD	Specifies the password for proxy server authentication.
PROXY_SERVER_NAME	Specifies the name or IP address of the proxy server. Valid IP addresses are in IPv4 (dot-separated) format or in IPv6 (colon-separated) format. Use IPv6 format only if IPv6 is configured.
PROXY_SERVER_PORT	Specifies the port or service name for the proxy service on the proxy server. Valid values are a decimal port number from 1 to 32760 or a service name.
PROXY_TYPE	Specifies the proxy type to use to access the Internet when the federated server is behind a firewall. Valid values are NONE, HTTP, and SOCKS. The default value is NONE.
SOCKET_TIMEOUT	Specifies the maximum time, in minutes, that the federated server waits for results from the proxy server. A valid value is any number that is greater than or equal to 0. The default is 0; the server waits an unlimited amount of time.
SSL_CLIENT_CERTIFICATE_LABEL	Specifies the client certificate to send during SSL authentication. If you do not specify a value, the current authorization ID for the federated database is used to locate the certificate.

Table 84. Server options for XML (continued)

Name	Description
SSL_KEYSTORE_FILE	Specifies the certificate storage file for communications that use SSL or TSL. A valid value is a fully qualified path name that is accessible by the federated database agent or by a fenced-mode process.
SSL_KEYSTORE_PASSWORD	Specifies the password to use to access the file in the SSL_KEYSTORE_FILE option. Valid values are a password, which is encrypted when it is stored in the federated database catalog, and `file:`*`file_name`*, where *file_name* is the fully qualified path to a stash file.
SSL_VERIFY_SERVER_CERTIFICATE	Specifies whether to verify the server certificate during SSL authentication. The default is N; the certificate is not verified.

User mapping options

Table 85. User mapping options for XML

Name	Description
PROXY_AUTHID	Specifies the user name for proxy server authentication.
PROXY_PASSWORD	Specifies the password for proxy server authentication.
SSL_CLIENT_CERTIFICATE_LABEL	Specifies the client certificate to send during SSL authentication. If you do not specify a value, the current authorization ID for the federated database is used to locate the certificate.

Nickname options

Table 86. Nickname options for XML

Name	Description
DIRECTORY_PATH	Specifies the path name of a directory that contains one or more XML files. Use this option to create a single nickname over multiple XML source files. The XML wrapper uses only the files with an .xml extension that are located in the directory that you specify. The XML wrapper ignores all other files in this directory. If you specify this nickname option, do not specify a DOCUMENT column. This option is valid only for the root nickname.
FILE_PATH	Specified the file path of the XML document. If you specify FILE_PATH, do not specify a DOCUMENT column. This option is valid only for the root nickname.

Table 86. Nickname options for XML (continued)

Name	Description
INSTANCE_PARSE_TIME	Specifies the time, in milliseconds, that is required to parse one row of the XML source document. The valid value can be an integer or decimal value. The default is 7. This option is valid only for columns of the root nickname. To optimize queries of large or complex XML source structures, modify the INSTANCE_PARSE_TIME, XPATH_EVAL_TIME, and NEXT_TIME options.

NAMESPACES	Specifies the namespaces that are associated with the namespace prefixes that are used in the XPATH and TEMPLATE options for each column. Use this syntax:
	`NAMESPACES'prefix1= "actual_namespace1", prefix2="actual_namespace2"'`
	
	Use a comma to separate multiple namespaces. For example:
	`NAMESPACES='http://www.myweb.com/cust", i='http://www.myweb.com/cust/id", n='http://www.myweb.com/cust/name"'`

NEXT_TIME	Specified the time, in milliseconds, that is required to locate subsequent source elements from the Xpath expression. The default is 1. This option is valid for root nicknames and nonroot nicknames. To optimize queries of large or complex XML source structures, modify the INSTANCE_PARSE_TIME, XPATH_EVAL_TIME, and NEXT_TIME options.
STREAMING	Specifies whether the source document should be separated into logical fragments for processing. The fragments correspond to the node that matches the XPath expression of the nickname. The wrapper then parses and processes source data fragment-by-fragment. This type of parsing minimizes memory use. Valid values are Y and N. The default is N; the documents are not parsed. This option is valid only on the root nickname.
VALIDATE	Specifies whether the source document is validated to ensure that it conforms to an XML schema or document type definition (DTD) before data is extracted from it. The default is N; validation does not occur. Before you set the value to Y, the schema file or DTD file is in the location that the source document specifies. This option is valid only for the root nickname. Do not set both the STREAMING and VALIDATE options to Y.

Table 86. Nickname options for XML (continued)

Name	Description
XPATH	Required. Specifies the Xpath expression that identifies the elements that represent the individual tuples. The Xpath expression is used as a context for evaluating column values that are identified by the XPATH column option.
XPATH_EVAL_TIME	Specifies the time required, in milliseconds, to evaluate the Xpath expression of the nickname and to locate the first element. The value can be an integer or decimal value. The default is 1. This option is valid for root nicknames and nonroot nicknames. To optimize queries of large or complex XML source structures, modify the INSTANCE_PARSE_TIME, XPATH_EVAL_TIME, and NEXT_TIME options.

Column options

Table 87. Column options for XML

Name	Description
DOCUMENT	Specifies that this column is a DOCUMENT column. The value of the DOCUMENT column indicates the type of XML source data that is supplied to the nickname when the query runs. This option is valid only for columns of the root nickname (the nickname that identifies the elements at the top level of the XML document). Only one column for each nickname can be specified with the DOCUMENT option. If you use a DOCUMENT column option instead of a FILE_PATH or DIRECTORY_PATH nickname option, the document that corresponds to this nickname is supplied when the query runs. Valid values are FILE, DIRECTORY, URI, and COLUMN. The default is FILE. The column must be of VARCHAR data type.

Table 87. Column options for XML (continued)

Name	Description
FOREIGN_KEY	Indicates that this nickname is a child nickname and specifies the name of the corresponding parent nickname. A nickname can have at most one FOREIGN_KEY column option. The value for the option is case-sensitive. Do not specify the XPATH option for this column. The column can be used only to join a parent nicknames and a child nickname. A CREATE NICKNAME statement that includes a FOREIGN_KEY option fails if the parent nickname has a different schema name. Unless the nickname that is referred to in a FOREIGN_KEY clause was explicitly defined as lowercase or mixed case in the CREATE NICKNAME statement, you must specify the nickname in uppercase when you refer to this nickname in the FOREIGN_KEY clause. **Note:** • When this option is set on a column, no other option can be set on the column. • If you set this column option, you cannot later use the ALTER NICKNAME statement to drop the option. Instead, you must drop the nickname and then create the nickname again without this column option.
PRIMARY_KEY	Required for a parent nickname that has one or more child nicknames. Specifies that this nickname is a parent nickname. The column data type must be VARCHAR(16). A nickname can have only one PRIMARY_KEY column option. Yes is the only valid value. Do not specify the XPATH option for this column. The column can be used only to join parent nicknames and child nicknames. **Note:** • When this option is set on a column, no other option can be set on the column. • If you set this column option, you cannot later use the ALTER NICKNAME statement to drop the option. Instead, you must drop the nickname and then create the nickname again without this column option.

Table 87. Column options for XML (continued)

Name	Description
XPATH	Specifies the Xpath expression in the XML document that contains the data that corresponds to this column. The wrapper evaluates this XPath expression after the CREATE NICKNAME statement applies the XPath expression from the XPATH nickname option. **Note:** If you set this column option, you cannot later use the ALTER NICKNAME statement to drop the option. Instead, you must drop the nickname and then create the nickname again without this column option.

Chapter 42. Views in the global catalog table containing federated information

Most of the catalog views in a federated database are the same as the catalog views in any other DB2 for Linux, UNIX, and Windows database.

There are several unique views that contain information pertinent to a federated system, such as the SYSCAT.WRAPPERS view.

The SYSCAT views are read-only. You cannot issue an update or insert operation on a view in the SYSCAT schema. Using the SYSSTAT views is the recommended way to update the system catalog. Change applications that reference the SYSCAT view to reference the updatable SYSSTAT view instead.

The following table lists the SYSCAT views which contain federated information. These are read-only views.

Table 88. Catalog views typically used with a federated system

Catalog views	Description
SYSCAT.CHECKS	Contains check constraint information that you defined.
SYSCAT.COLCHECKS	Contains columns referenced by a check constraint.
SYSCAT.COLUMNS	Contains column information about the data source objects (tables and views) that you created nicknames for.
SYSCAT.COLOPTIONS	Contains information about column option values that you set for a nickname.
SYSCAT.CONSTDEP	Contains the dependency of an informational constraint that you defined.
SYSCAT.DATATYPES	Contains data type information about local built-in and user-defined DB2 data types.
SYSCAT.DBAUTH	Contains the database authorities held by individual users and groups.
SYSCAT.FUNCMAPOPTIONS	Contains information about option values that you have set for a function mapping.
SYSCAT.FUNCMAPPINGS	Contains the function mappings between the federated database and the data source objects.
SYSCAT.INDEXCOLUSE	Contains columns that participate in an index.
SYSCAT.INDEXES	Contains index specifications for data source objects.
SYSCAT.INDEXOPTIONS	Contains information about index options.
SYSCAT.KEYCOLUSE	Contains columns that participate in a key defined by a unique key, primary key, or foreign key constraint.
SYSCAT.NICKNAMES	Contains information about nicknames that you created.

Table 88. Catalog views typically used with a federated system (continued)

Catalog views	Description
SYSCAT.REFERENCES	Contains information about referential constraints that you defined.
SYSCAT.ROUTINES	Contains local DB2 user-defined functions, or function templates. Function templates are used to map to a data source function.
SYSCAT.REVTYPEMAPPINGS	This view is not used. All data type mappings are recorded in the SYSCAT.TYPEMAPPINGS view.
SYSCAT.ROUTINEOPTIONS	Contains information about federated routine option values.
SYSCAT.ROUTINEPARMOPTIONS	Contains information about federated routine parameter option values.
SYSCAT.ROUTINEPARMS	Contains a parameter or the result of a routine defined in SYSCAT.ROUTINES.
SYSCAT.ROUTINESFEDERATED	Contains information about federated routines that you defined.
SYSCAT.SERVERS	Contains server definitions that you create for data source servers.
SYSCAT.TABCONST	Each row represents a table and nickname constraints of type CHECK, UNIQUE, PRIMARY KEY, or FOREIGN KEY.
SYSCAT.TABLES	Contains information about each local DB2 table, federated view, and nickname that you create.
SYSCAT.TYPEMAPPINGS	Contains forward data type mappings and reverse data type mappings. The mapping is to local DB2 data types from data source data types. These mappings are used when you create a nickname on a data source object.
SYSCAT.USEROPTIONS	Contains user authorization information that you set when you create user mappings between the federated database and the data source servers.
SYSCAT.VIEWS	Contains information about local federated views that you create.
SYSCAT.WRAPOPTIONS	Contains information about option values that you have set for a wrapper.
SYSCAT.WRAPPERS	Contains the name of the wrapper and library file for each data source that you create a wrapper for.

The following table lists the SYSSTAT views which contain federated information. These are read-write views that contain statistics you can update.

Table 89. Federated updatable global catalog views

Catalog views	Description
SYSSTAT.COLUMNS	Contains statistical information about each column in the data source objects (tables and views) that you have created nicknames for. Statistics are not recorded for inherited columns of typed tables.
SYSSTAT.INDEXES	Contains statistical information about each index specification for data source objects.
SYSSTAT.ROUTINES	Contains statistical information about each user-defined function. Does not include built-in functions. Statistics are not recorded for inherited columns of typed tables.
SYSSTAT.TABLES	Contains information about each base table. View, synonym, and alias information is not included in this view. For typed tables, only the root table of a table hierarchy is included in the view. Statistics are not recorded for inherited columns of typed tables.

Chapter 43. Function mapping options for federated systems

The federated server provides default mappings between DB2 functions and data source functions. For most data sources, the default function mappings are in the wrappers. To use a data source function that the federated server does not recognize or to change the default mapping, you create a function mapping.

When you create a function mapping, you specify the name of the data source function and must enable the mapped function. Then when you use the mapped function, the query optimizer compares the cost of running the function at the data source with the cost of running the function at the federated server.

Table 90. Options for function mappings

Name	Description
DISABLE	Enable or disable a default function mapping. Valid values are Y and N. The default is N.
REMOTE_NAME	The name of the data source function. The default is the local name.

Chapter 44. Valid server types in SQL statements

Server types indicate the kind of data source that the server definition represents.

Server types vary by vendor, purpose, and operating system. Supported values depend on the data source.

For most data sources, you must specify a valid server type in the CREATE SERVER statement.

Table 91. Data sources and server types

Data source	Server type
BioRS	A server type is not required in the CREATE SERVER statement.
Excel	A server type is not required in the CREATE SERVER statement.
IBM DB2 Universal Database for Linux, UNIX, and Windows	DB2/UDB
IBM DB2 Universal Database for System i and AS/400®	DB2/ISERIES
IBM DB2 Universal Database for z/OS	DB2/ZOS
IBM DB2 for VM	DB2/VM
Informix	INFORMIX
JDBC	JDBC (Required for JDBC data sources that are supported by JDBC drivers 3.0 and later.)
Microsoft SQL Server	MSSQLSERVER (Required for data sources supported by the DataDirect Connect ODBC 4.2 (or later) driver or the Microsoft SQL Server ODBC 3.0 (or later) driver.)
ODBC	ODBC (Required for ODBC data sources that are supported by the ODBC 3.x driver.)
OLE DB	A server type is not required in the CREATE SERVER statement.
Oracle	ORACLE (Required for Oracle data sources supported by Oracle NET8 client software.)
Sybase (CTLIB)	SYBASE
Table-structured files	A server type is not required in the CREATE SERVER statement.
Teradata	TERADATA
Web services	A server type is not required in the CREATE SERVER statement.
XML	A server type is not required in the CREATE SERVER statement.

Chapter 45. Data type mappings

Data types mappings for relational data sources include forward type mappings, reverse type mappings, and type mappings that are specific to Unicode. Each of the nonrelational data sources support specific data types.

Default forward data type mappings

The two kinds of mappings between data source data types and federated database data types are forward type mappings and reverse type mappings. In a forward type mapping, the mapping is from a remote type to a comparable local type.

You can override a default type mapping, or create a new type mapping with the CREATE TYPE MAPPING statement.

These mappings are valid with all the supported versions, unless otherwise noted.

For all default forward data types mapping from a data source to the federated database, the federated schema is SYSIBM.

The following tables show the default forward mappings between federated database data types and data source data types.

Default forward data type mappings for DB2 Database for Linux, UNIX, and Windows data sources

The following table lists the default forward data type mappings for DB2 Database for Linux, UNIX, and Windows data sources.

Table 92. DB2 Database for Linux, UNIX, and Windows default forward data type mappings (Not all columns shown)

Remote Typename	Remote Lower Len	Remote Upper Len	Remote Lower Scale	Remote Upper Scale	Remote Bit Data	Remote Data Operators	Federated Typename	Federated Length	Federated Scale	Federated Bit Data
BIGINT	-	-	-	-	-	-	BIGINT	-	0	-
BLOB	-	-	-	-	-	-	BLOB	-	-	-
CHAR	-	-	-	-	-	-	CHAR	-	0	N
CHAR	-	-	-	-	Y	-	CHAR	-	0	Y
CLOB	-	-	-	-	-	-	CLOB	-	-	-
DATE	-	-	-	-	-	-	DATE	-	0	-
DATE	-	-	-	-	-	-	TIMESTAMP[1]	-	0	-
DBCLOB	-	-	-	-	-	-	DBCLOB	-	-	-
DECIMAL	-	-	-	-	-	-	DECIMAL	-	-	-
DECFLOAT [2]	-	-	-	-	-	-	DECFLOAT	-	0	-
DOUBLE	-	-	-	-	-	-	DOUBLE	-	-	-
FLOAT	-	-	-	-	-	-	DOUBLE	-	-	-
GRAPHIC	-	-	-	-	-	-	GRAPHIC	-	0	N
INTEGER	-	-	-	-	-	-	INTEGER	-	0	-
LONGVAR	-	-	-	-	N	-	CLOB	-	-	-
LONGVAR	-	-	-	-	Y	-	BLOB	-	-	-

Table 92. DB2 Database for Linux, UNIX, and Windows default forward data type mappings (Not all columns shown) (continued)

Remote Typename	Remote Lower Len	Remote Upper Len	Remote Lower Scale	Remote Upper Scale	Remote Bit Data	Remote Data Operators	Federated Typename	Federated Length	Federated Scale	Federated Bit Data
LONGVARG	-	-	-	-	-	-	DBCLOB	-	-	-
REAL	-	-	-	-	-	-	REAL	-	-	-
SMALLINT	-	-	-	-	-	-	SMALLINT	-	0	-
TIME	-	-	-	-	-	-	TIME	-	0	-
TIMESTAMP(*p*)	-	-	*p*	*p*	-	-	TIMESTAMP(*p*)	-	*p*	-
VARCHAR	-	-	-	-	-	-	VARCHAR	-	0	N
VARCHAR	-	-	-	-	Y	-	VARCHAR	-	0	Y
VARGRAPH	-	-	-	-	-	-	VARGRAPHIC	-	0	N
VARGRAPHIC	-	-	-	-	-	-	VARGRAPHIC	-	0	N

Note:

1. The federated type is TIMESTAMP(0) if the date_compat configuration parameter is set to ON.
2. The SAME_DECFLT_ROUNDING server option is set to N by default and operations will not be pushed down to the remote data source unless SAME_DECFLT_ROUNDING is set to Y. For information on the SAME_DECFLT_ROUNDING server option, see DB2 database options reference.

Default forward data type mappings for DB2 for System i data sources

The following table lists the default forward data type mappings for DB2 for System i data sources.

Table 93. DB2 for System i default forward data type mappings (Not all columns shown)

Remote Typename	Remote Lower Len	Remote Upper Len	Remote Lower Scale	Remote Upper Scale	Remote Bit Data	Remote Data Operators	Federated Typename	Federated Length	Federated Scale	Federated Bit Data
BLOB	-	-	-	-	-	-	BLOB	-	-	-
CHAR	1	254	-	-	-	-	CHAR	-	0	N
CHAR	255	32672	-	-	-	-	VARCHAR	-	0	N
CHAR	1	254	-	-	Y	-	CHAR	-	0	Y
CHAR	255	32672	-	-	Y	-	VARCHAR	-	0	Y
CLOB	-	-	-	-	-	-	CLOB	-	-	-
DATE	-	-	-	-	-	-	DATE	-	0	-
DBCLOB	-	-	-	-	-	-	DBCLOB	-	-	-
DECIMAL	-	-	-	-	-	-	DECIMAL	-	-	-
FLOAT	4	-	-	-	-	-	REAL	-	-	-
FLOAT	8	-	-	-	-	-	DOUBLE	-	-	-
GRAPHIC	1	127	-	-	-	-	GRAPHIC	-	0	N
GRAPHIC	128	16336	-	-	-	-	VARGRAPHIC	-	0	N
INTEGER	-	-	-	-	-	-	INTEGER	-	0	-
NUMERIC	-	-	-	-	-	-	DECIMAL	-	-	-
SMALLINT	-	-	-	-	-	-	SMALLINT	-	0	-
TIME	-	-	-	-	-	-	TIME	-	0	-
TIMESTAMP	-	-	-	-	-	-	TIMESTAMP(6)	-	6	-
VARCHAR	1	32672	-	-	-	-	VARCHAR	-	0	N
VARCHAR	1	32672	-	-	Y	-	VARCHAR	-	0	Y

Table 93. DB2 for System i default forward data type mappings (Not all columns shown) (continued)

Remote Typename	Remote Lower Len	Remote Upper Len	Remote Lower Scale	Remote Upper Scale	Remote Bit Data	Remote Data Operators	Federated Typename	Federated Length	Federated Scale	Federated Bit Data
VARG	1	16336	-	-	-	-	VARGRAPHIC	-	0	N
VARGRAPHIC	1	16336	-	-	-	-	VARGRAPHIC	-	0	N

Default forward data type mappings for DB2 for VM and VSE data sources

The following table lists the default forward data type mappings for DB2 for VM and VSE data sources.

Table 94. DB2 Server for VM and VSE default forward data type mappings (Not all columns shown)

Remote Typename	Remote Lower Len	Remote Upper Len	Remote Lower Scale	Remote Upper Scale	Remote Bit Data	Remote Data Operators	Federated Typename	Federated Length	Federated Scale	Federated Bit Data
BLOB	-	-	-	-	-	-	BLOB	-	-	-
CHAR	1	254	-	-	-	-	CHAR	-	0	N
CHAR	1	254	-	-	Y	-	CHAR	-	0	Y
CLOB	-	-	-	-	-	-	CLOB	-	-	-
DATE	-	-	-	-	-	-	DATE	-	0	-
DBAHW	-	-	-	-	-	-	SMALLINT	-	0	-
DBAINT	-	-	-	-	-	-	INTEGER	-	0	-
DBCLOB	-	-	-	-	-	-	DBCLOB	-	-	-
DECIMAL	-	-	-	-	-	-	DECIMAL	-	-	-
FLOAT	4	-	-	-	-	-	REAL	-	-	-
FLOAT	8	-	-	-	-	-	DOUBLE	-	-	-
GRAPHIC	1	127	-	-	-	-	GRAPHIC	-	0	N
INTEGER	-	-	-	-	-	-	INTEGER	-	-	-
SMALLINT	-	-	-	-	-	-	SMALLINT	-	-	-
TIME	-	-	-	-	-	-	TIME	-	0	-
TIMESTAMP	-	-	-	-	-	-	TIMESTAMP(6)	-	6	-
VARCHAR	1	32672	-	-	-	-	VARCHAR	-	0	N
VARCHAR	1	32672	-	-	Y	-	VARCHAR	-	0	Y
VARGRAPHIC	1	16336	-	-	-	-	VARGRAPHIC	-	0	N
VARGRAPH	1	16336	-	-	-	-	VARGRAPHIC	-	0	N

Default forward data type mappings for DB2 for z/OS data sources

The following table lists the default forward data type mappings for DB2 for z/OS data sources.

Table 95. DB2 for z/OS default forward data type mappings (Not all columns shown)

Remote Typename	Remote Lower Len	Remote Upper Len	Remote Lower Scale	Remote Upper Scale	Remote Bit Data	Remote Data Operators	Federated Typename	Federated Length	Federated Scale	Federated Bit Data
BIGINT	0	8	0	0	" "	"/0"	BIGINT	0	0	N
BLOB	-	-	-	-	-	-	BLOB	-	-	-
CHAR	1	254	-	-	-	-	CHAR	-	0	N

Table 95. DB2 for z/OS default forward data type mappings (Not all columns shown) (continued)

Remote Typename	Remote Lower Len	Remote Upper Len	Remote Lower Scale	Remote Upper Scale	Remote Bit Data	Remote Data Operators	Federated Typename	Federated Length	Federated Scale	Federated Bit Data
CHAR	255	32672	-	-	-	-	VARCHAR	-	0	N
CHAR	1	254	-	-	Y	-	CHAR	-	0	Y
CHAR	255	32672	-	-	Y	-	VARCHAR	-	0	Y
CLOB	-	-	-	-	-	-	CLOB	-	-	-
DATE	-	-	-	-	-	-	DATE	-	0	-
DBCLOB	-	-	-	-	-	-	DBCLOB	-	-	-
DECIMAL	-	-	-	-	-	-	DECIMAL	-	-	-
FLOAT	4	-	-	-	-	-	REAL	-	-	-
FLOAT	8	-	-	-	-	-	DOUBLE	-	-	-
GRAPHIC	1	127	-	-	-	-	GRAPHIC	-	0	N
INTEGER	-	-	-	-	-	-	INTEGER	-	0	-
ROWID	-	-	-	-	Y	-	VARCHAR	40	-	Y
SMALLINT	-	-	-	-	-	-	SMALLINT	-	0	-
TIME	-	-	-	-	-	-	TIME	-	0	-
TIMESTAMP	-	-	-	-	-	-	TIMESTAMP(6)	-	6	-
VARCHAR	1	32672	-	-	-	-	VARCHAR	-	0	N
VARCHAR	1	32672	-	-	Y	-	VARCHAR	-	0	Y
VARG	1	16336	-	-	-	-	VARGRAPHIC	-	0	N
VARGRAPHIC	1	16336	-	-	-	-	VARGRAPHIC	-	0	N

Default forward data type mappings for Informix data sources

The following table lists the default forward data type mappings for Informix data sources.

Table 96. Informix default forward data type mappings (Not all columns shown)

Remote Typename	Remote Lower Len	Remote Upper Len	Remote Lower Scale	Remote Upper Scale	Remote Bit Data	Remote Data Operators	Federated Typename	Federated Length	Federated Scale	Federated Bit Data
BLOB	-	-	-	-	-	-	BLOB	2147483647	-	-
BOOLEAN	-	-	-	-	-	-	CHARACTER	1	-	-
BYTE	-	-	-	-	-	-	BLOB	2147483647	-	-
CHAR	1	254	-	-	-	-	CHARACTER	-	-	-
CHAR	255	32672	-	-	-	-	VARCHAR	-	-	-
CLOB	-	-	-	-	-	-	CLOB	2147483647	-	-
DATE	-	-	-	-	-	-	DATE	4	-	-
DATE	-	-	-	-	-	-	TIMESTAMP[1]	-	0	-
DATETIME[2]	0	4	0	4	-	-	DATE	4	-	-
DATETIME	6	10	6	10	-	-	TIME	3	-	-
DATETIME	0	4	6	15	-	-	TIMESTAMP(6)	10	6	-
DATETIME	6	10	11	15	-	-	TIMESTAMP(6)	10	6	-
DECIMAL	1	31	0	31	-	-	DECIMAL	-	-	-
DECIMAL	32	130	-	-	-	-	DOUBLE	8	-	-
DECIMAL	1	32	255	255	-	-	DOUBLE	-	-	-
FLOAT	-	-	-	-	-	-	DOUBLE	8	-	-

Table 96. Informix default forward data type mappings (Not all columns shown) (continued)

Remote Typename	Remote Lower Len	Remote Upper Len	Remote Lower Scale	Remote Upper Scale	Remote Bit Data	Remote Data Operators	Federated Typename	Federated Length	Federated Scale	Federated Bit Data
INTEGER	-	-	-	-	-	-	INTEGER	4	-	-
INTERVAL	-	-	-	-	-	-	VARCHAR	25	-	-
INT8	-	-	-	-	-	-	BIGINT	19	0	-
LVARCHAR	1	32672	-	-	-	-	VARCHAR	-	-	-
MONEY	1	31	0	31	-	-	DECIMAL	-	-	-
MONEY	32	32	-	-	-	-	DOUBLE	8	-	-
NCHAR	1	254	-	-	-	-	CHARACTER	-	-	-
NCHAR	255	32672	-	-	-	-	VARCHAR	-	-	-
NVARCHAR	1	32672	-	-	-	-	VARCHAR	-	-	-
REAL	-	-	-	-	-	-	REAL	4	-	-
SERIAL	-	-	-	-	-	-	INTEGER	4	-	-
SERIAL8	-	-	-	-	-	-	BIGINT	-	-	-
SMALLFLOAT	-	-	-	-	-	-	REAL	4	-	-
SMALLINT	-	-	-	-	-	-	SMALLINT	2	-	-
TEXT	-	-	-	-	-	-	CLOB	2147483647	-	-
VARCHAR	1	32672	-	-	-	-	VARCHAR	-	-	-

Notes:

1. The federated type is TIMESTAMP(0) if the date_compat configuration parameter is set to ON.
2. For the Informix DATETIME data type, the DB2 UNIX and Windows federated server uses the Informix high-level qualifier as the REMOTE_LENGTH and the Informix low-level qualifier as the REMOTE_SCALE.

The Informix qualifiers are the "TU_" constants defined in the Informix Client SDK `datatime.h` file. The constants are:

0 = YEAR	8 = MINUTE	13 = FRACTION(3)
2 = MONTH	10 = SECOND	14 = FRACTION(4)
4 = DAY	11 = FRACTION(1)	15 = FRACTION(5)
6 = HOUR	12 = FRACTION(2)	

Default forward data type mappings for JDBC data sources

The following table lists the default forward data type mappings for JDBC data sources.

Table 97. JDBC default forward data type mappings

Remote Typename	Remote Lower Len	Remote Upper Len	Remote Lower Scale	Remote Upper Scale	Remote Bit Data	Remote Data Operators	Federated Typename	Federated Length	Federated Scale	Federated Bit Data
BIGINT	-	-	-	-	-	-	BIGINT	8	-	-
BINARY	-	254	-	-	-	-	CHAR	-	-	Y
BINARY	255	32672	-	-	-	-	VARCHAR	-	-	Y
BINARY	32673	2147483647	-	-	-	-	BLOB	2147483647	-	-
BIT	-	-	-	-	-	-	SMALLINT	2	-	-
BLOB	-	-	-	-	-	-	BLOB	2147483647	-	-
BOOLEAN	-	-	-	-	-	-	SMALLINT	2	-	-
CHAR	-	254	-	-	-	-	CHAR	-	-	-
CHAR	255	32672	-	-	-	-	VARCHAR	-	-	-
CHAR	32673	2147483647	-	-	-	-	CLOB	2147483647	-	-

Table 97. JDBC default forward data type mappings (continued)

Remote Typename	Remote Lower Len	Remote Upper Len	Remote Lower Scale	Remote Upper Scale	Remote Bit Data	Remote Data Operators	Federated Typename	Federated Length	Federated Scale	Federated Bit Data
CLOB	-	-	-	-	-	-	CLOB	2147483647	-	-
DATE	-	-	-	-	-	-	DATE	-	-	-
DATE	-	-	-	-	-	-	TIMESTAMP[1]	-	-	-
DECIMAL	1	31	0	31	-	-	DECIMAL	-	-	-
DECIMAL	32	38	0	38	-	-	DOUBLE	8	-	-
DOUBLE	-	-	-	-	-	-	DOUBLE	8	-	-
FLOAT	-	-	-	-	-	-	FLOAT	4	-	-
INTEGER	-	-	-	-	-	-	INTEGER	4	-	-
LONGVARCHAR	-	32672	-	-	-	-	VARCHAR	-	-	-
LONGVARCHAR	32673	2147483647	-	-	-	-	CLOB	2147483647	-	-
LONGVARBINARY	-	32672	-	-	-	-	VARCHAR	-	-	Y
LONGVARBINARY	32673	2147483647	-	-	-	-	BLOB	2147483647	-	-
LONGNVARCHAR	-	16336	-	-	-	-	VARGRAPHIC	-	-	-
LONGNVARCHAR[2]	16337	1073741823	-	-	-	-	DBCLOB	-	-	-
NCHAR[2]	-	127	-	-	-	-	GRAPHIC	-	-	-
NCHAR[2]	128	16336	-	-	-	-	VARGRAPHIC	-	-	-
NCHAR[2]	16337	1073741823	-	-	-	-	DBCLOB	-	-	-
NCLOB[2]	-	-	-	-	-	-	DBCLOB	-	-	-
NUMERIC	1	31	0	31	-	-	DECIMAL	-	-	-
NUMERIC	32	38	0	38	-	-	DOUBLE	8	-	-
NVARCHAR[2]	-	16336	-	-	-	-	VARGRAPHIC	-	-	-
NVARCHAR[2]	16337	1073741823	-	-	-	-	DBCLOB	-	-	-
REAL							REAL	4		
SMALLINT	-	-	-	-	-	-	SMALLINT	2	-	-
TIME	-	-	-	-	-	-	TIME	3	-	-
TIMESTAMP	-	-	-	-	-	-	TIMESTAMP(6)	10	6	-
TIMESTAMP(*p*)	-	-	-	-	-	-	TIMESTAMP(6)	10	6	-
TINYINT	-	-	-	-	-	-	SMALLINT	2	-	-
VARBINARY	-	32672	-	-	-	-	VARCHAR	-	-	Y
VARBINARY	32673	2147483647	-	-	-	-	BLOB	2147483647	-	-
VARCHAR	-	32672	-	-	-	-	VARCHAR	-	-	-
VARCHAR	32673	2147483647	-	-	-	-	CLOB	2147483647	-	-

Note:

1. The federated type is TIMESTAMP(0) if the date_compat configuration parameter is set to ON.
2. Data types that are only supported by the JDBC 4.0 driver: NCHAR, NVARCHAR, LONGVARCHAR, and NCLOB.

The following data types are unsupported by the JDBC wrapper: DATALINK, OTHER, JAVA_OBJECT, DISTINCT, STRUCT, ARRAY, and REF.

Default forward data type mappings for Microsoft SQL Server data sources

The following table lists the default forward data type mappings for Microsoft SQL Server data sources.

Table 98. Microsoft SQL Server default forward data type mappings

Remote Typename	Remote Lower Len	Remote Upper Len	Remote Lower Scale	Remote Upper Scale	Remote Bit Data	Remote Data Operators	Federated Typename	Federated Length	Federated Scale	Federated Bit Data
bigint	-	-	-	-	-	-	BIGINT	-	-	-
binary	1	254	-	-	-	-	CHARACTER	-	-	Y
binary	255	8000	-	-	-	-	VARCHAR	-	-	Y
bit	-	-	-	-	-	-	SMALLINT	2	-	-
char	1	254	-	-	-	-	CHAR	-	-	N
char	255	8000	-	-	-	-	VARCHAR	-	-	N
datetime	-	-	-	-	-	-	TIMESTAMP(6)	10	6	-
decimal	1	31	0	31	-	-	DECIMAL	-	-	-
decimal	32	38	0	38	-	-	DOUBLE	-	-	-
float	-	8	-	-	-	-	DOUBLE	8	-	-
float	-	4	-	-	-	-	REAL	4	-	-
image	-	-	-	-	-	-	BLOB	2147483647	-	Y
int	-	-	-	-	-	-	INTEGER	4	-	-
money	-	-	-	-	-	-	DECIMAL	19	4	-
nchar	1	127	-	-	-	-	CHAR	-	-	N
nchar	128	4000	-	-	-	-	VARCHAR	-	-	N
numeric	1	31	0	31	-	-	DECIMAL	-	-	-
numeric	32	38	0	38	-	-	DOUBLE	8	-	-
ntext	-	-	-	-	-	-	CLOB	2147483647	-	Y
nvarchar	1	4000	-	-	-	-	VARCHAR	-	-	N
real	-	-	-	-	-	-	REAL	4	-	-
smallint	-	-	-	-	-	-	SMALLINT	2	-	-
smalldatetime	-	-	-	-	-	-	TIMESTAMP(6)	10	6	-
smallmoney	-	-	-	-	-	-	DECIMAL	10	4	-
SQL_BIGINT	-	-	-	-	-	-	BIGINT	-	-	-
SQL_BINARY	1	254	-	-	-	-	CHARACTER	-	-	Y
SQL_BINARY	255	8000	-	-	-	-	VARCHAR	-	-	Y
SQL_BIT	-	-	-	-	-	-	SMALLINT	2	-	-
SQL_CHAR	1	254	-	-	-	-	CHAR	-	-	N
SQL_CHAR	255	8000	-	-	-	-	VARCHAR	-	-	N
SQL_DATE	-	-	-	-	-	-	DATE	4	-	-
SQL_DECIMAL	1	31	0	31	-	-	DECIMAL	-	-	-
SQL_DECIMAL	32	38	0	38	-	-	DOUBLE	8	-	-
SQL_DOUBLE	-	-	-	-	-	-	DOUBLE	8	-	-
SQL_FLOAT	-	-	-	-	-	-	DOUBLE	8	-	-
SQL_GUID	-	-	-	-	-	-	VARCHAR	-	-	Y
SQL_INTEGER	-	-	-	-	-	-	INTEGER	4	-	-
SQL_ LONGVARCHAR	-	-	-	-	-	-	CLOB	2147483647	-	N
SQL_ LONGVARBINARY	-	-	-	-	-	-	BLOB	-	-	Y
SQL_NUMERIC	1	31	0	31	-	-	DECIMAL	-	-	-
SQL_NUMERIC	32	38	0	38	-	-	DOUBLE	8	-	-
SQL_REAL	-	-	-	-	-	-	REAL	8	-	-
SQL_SMALLINT	-	-	-	-	-	-	SMALLINT	2	-	-

Table 98. Microsoft SQL Server default forward data type mappings (continued)

Remote Typename	Remote Lower Len	Remote Upper Len	Remote Lower Scale	Remote Upper Scale	Remote Bit Data	Remote Data Operators	Federated Typename	Federated Length	Federated Scale	Federated Bit Data
SQL_TIME	-	-	-	-	-	-	TIME	3	-	-
SQL_TIMESTAMP	-	-	-	-	-	-	TIMESTAMP	10	6	-
SQL_TINYINT	-	-	-	-	-	-	SMALLINT	2	-	-
SQL_VARBINARY	1	8000	-	-	-	-	VARCHAR	-	-	Y
SQL_VARCHAR	1	8000	-	-	-	-	VARCHAR	-	-	N
SQL_WCHAR	1	254	-	-	-	-	CHARACTER	-	-	N
SQL_WCHAR	255	8800	-	-	-	-	VARCHAR	-	-	N
SQL_WLONGVARCHAR	-	1073741823	-	-	-	-	CLOB	2147483647	-	N
SQL_WVARCHAR	1	16336	-	-	-	-	VARCHAR	-	-	N
text	-	-	-	-	-	-	CLOB	-	-	N
timestamp	-	-	-	-	-	-	VARCHAR	8		Y
tinyint	-	-	-	-	-	-	SMALLINT	2	-	-
uniqueidentifier	1	4000	-	-	Y	-	VARCHAR	16	-	Y
varbinary	1	8000	-	-	-	-	VARCHAR	-	-	Y
varchar	1	8000	-	-	-	-	VARCHAR	-	-	N

Default forward data type mappings for ODBC data sources

The following table lists the default forward data type mappings for ODBC data sources.

Table 99. ODBC default forward data type mappings (Not all columns shown)

Remote Typename	Remote Lower Len	Remote Upper Len	Remote Lower Scale	Remote Upper Scale	Remote Bit Data	Remote Data Operators	Federated Typename	Federated Length	Federated Scale	Federated Bit Data
SQL_BIGINT	-	-	-	-	-	-	BIGINT	8	-	-
SQL_BINARY	1	254	-	-	-	-	CHARACTER	-	-	Y
SQL_BINARY	255	32672	-	-	-	-	VARCHAR	-	-	Y
SQL_BIT	-	-	-	-	-	-	SMALLINT	2	-	-
SQL_CHAR	1	254	-	-	-	-	CHAR	-	-	N
SQL_CHAR	255	32672	-	-	-	-	VARCHAR	-	-	N
SQL_DATE	-	-	-	-	-	-	DATE	-	-	-
SQL_DATE	-	-	-	-	-	-	TIMESTAMP[1]	-	-	-
SQL_DECIMAL	1	31	0	31	-	-	DECIMAL	-	-	-
SQL_DECIMAL	32	38	0	38	-	-	DOUBLE	8	-	-
SQL_DOUBLE	-	-	-	-	-	-	DOUBLE	8	-	-
SQL_FLOAT	-	8	-	-	-	-	FLOAT	8	-	-
SQL_FLOAT	-	4	-	-	-	-	FLOAT	4	-	-
SQL_INTEGER	-	-	-	-	-	-	INTEGER	4	-	-
SQL_ LONGVARCHAR	-	-	-	-	-	-	CLOB	2147483647	-	N
SQL_ LONGVARBINARY	-	-	-	-	-	-	BLOB	2147483647	-	Y
SQL_NUMERIC	1	31	0	31	-	-	DECIMAL	-	-	-
SQL_NUMERIC	32	32	0	31	-	-	DOUBLE	8	-	-

Table 99. ODBC default forward data type mappings (Not all columns shown) (continued)

Remote Typename	Remote Lower Len	Remote Upper Len	Remote Lower Scale	Remote Upper Scale	Remote Bit Data	Remote Data Operators	Federated Typename	Federated Length	Federated Scale	Federated Bit Data
SQL_REAL	-	-	-	-	-	-	REAL	4	-	-
SQL_SMALLINT	-	-	-	-	-	-	SMALLINT	2	-	-
SQL_TIMESTAMP	-	-	-	-	-	-	TIMESTAMP(6)	10	6	-
SQL_TIMESTAMP(*p*)	-	-	-	-	-	-	TIMESTAMP(6)	10	6	-
SQL_TYPE_DATE	-	-	-	-	-	-	DATE	4	-	-
SQL_TYPE_TIME	-	-	-	-	-	-	TIME	3	-	-
SQL_TYPE_ TIMESTAMP	-	-	-	-	-	-	TIMESTAMP	10	-	-
SQL_TINYINT	-	-	-	-	-	-	SMALLINT	2	-	-
SQL_VARBINARY	1	32672	-	-	-	-	VARCHAR	-	-	Y
SQL_VARCHAR	1	32672	-	-	-	-	VARCHAR	-	-	N
SQL_WCHAR	1	127	-	-	-	-	CHAR	-	-	N
SQL_WCHAR	128	16336	-	-	-	-	VARCHAR	-	-	N
SQL_WVARCHAR	1	16336	-	-	-	-	VARCHAR	-	-	N
SQL_ WLONGVARCHAR	-	1073741823	-	-	-	-	CLOB	2147483647	-	N

Note:

1. The federated type is TIMESTAMP(0) if the date_compat configuration parameter is set to ON.

Default forward data type mappings for Oracle NET8 data sources

The following table lists the default forward data type mappings for Oracle NET8 data sources.

Table 100. Oracle NET8 default forward data type mappings

Remote Typename	Remote Lower Len	Remote Upper Len	Remote Lower Scale	Remote Upper Scale	Remote Bit Data	Remote Data Operators	Federated Typename	Federated Length	Federated Scale	Federated Bit Data
BLOB	0	0	0	0	-	\0	BLOB	2147483647	0	Y
CHAR	1	254	0	0	-	\0	CHAR	0	0	N
CHAR	255	2000	0	0	-	\0	VARCHAR	0	0	N
CLOB	0	0	0	0	-	\0	CLOB	2147483647	0	N
DATE	0	0	0	0	-	\0	TIMESTAMP(6)	0	0	N
FLOAT	1	126	0	0	-	\0	DOUBLE	0	0	N
LONG	0	0	0	0	-	\0	CLOB	2147483647	0	N
LONG RAW	0	0	0	0	-	\0	BLOB	2147483647	0	Y
NUMBER	10	18	0	0	-	\0	BIGINT	0	0	N
NUMBER	1	38	-84	127	-	\0	DOUBLE	0	0	N
NUMBER	1	31	0	31	-	>=	DECIMAL	0	0	N
NUMBER	1	4	0	0	-	\0	SMALLINT	0	0	N
NUMBER	5	9	0	0	-	\0	INTEGER	0	0	N
NUMBER	-	10	0	0	-	\0	DECIMAL	0	0	N
RAW	1	2000	0	0	-	\0	VARCHAR	0	0	Y
ROWID	0	0	0	NULL	-	\0	CHAR	18	0	N
TIMESTAMP(*p*) [1]	-	-	-	-	-	\0	TIMESTAMP(6)	10	6	N

Table 100. Oracle NET8 default forward data type mappings (continued)

Remote Typename	Remote Lower Len	Remote Upper Len	Remote Lower Scale	Remote Upper Scale	Remote Bit Data	Remote Data Operators	Federated Typename	Federated Length	Federated Scale	Federated Bit Data
VARCHAR2	1	4000	0	0	-	\0	VARCHAR	0	0	N

Note:

1.

- TIMESTAMP*(p)* represents a timestamp with a variable scale from 0-9. The scale of the Oracle timestamp is mapped to TIMESTAMP(6) by default. You can change this default type mapping and map the Oracle TIMESTAMP to a federated TIMESTAMP of the same scale by using a user-defined type mapping.
- This type mapping is valid only for Oracle 9i (or later) client and server configurations.

Default forward data type mappings for Sybase data sources

The following table lists the default forward data type mappings for Sybase data sources.

Table 101. Sybase CTLIB default forward data type mappings

Remote Typename	Remote Lower Len	Remote Upper Len	Remote Lower Scale	Remote Upper Scale	Remote Bit Data	Remote Data Operators	Federated Typename	Federated Length	Federated Scale	Federated Bit Data
binary	1	254	-	-	-	-	CHAR	-	-	Y
binary	255	32672	-	-	-	-	VARCHAR	-	-	Y
bit	-	-	-	-	-	-	SMALLINT	-	-	-
char	1	254	-	-	-	-	CHAR	-	-	N
char	255	32672	-	-	-	-	VARCHAR	-	-	N
char null (see varchar)										
date	-	-	-	-	-	-	DATE	-	-	-
date	-	-	-	-	-	-	TIMESTAMP[1]	-	-	-
datetime	-	-	-	-	-	-	TIMESTAMP(6)	-	-	-
datetimn	-	-	-	-	-	-	TIMESTAMP	-	-	-
decimal	1	31	0	31	-	-	DECIMAL	-	-	-
decimal	32	38	0	38	-	-	DOUBLE	-	-	-
decimaln	1	31	0	31	-	-	DECIMAL	-	-	-
decimaln	32	38	0	38	-	-	DOUBLE	-	-	-
float	-	4	-	-	-	-	REAL	-	-	-
float	-	8	-	-	-	-	DOUBLE	-	-	-
floatn	-	4	-	-	-	-	REAL	-	-	-
floatn	-	8	-	-	-	-	DOUBLE	-	-	-
image	-	-	-	-	-	-	BLOB	-	-	-
int	-	-	-	-	-	-	INTEGER	-	-	-
intn	-	-	-	-	-	-	INTEGER	-	-	-
money	-	-	-	-	-	-	DECIMAL	19	4	-
moneyn	-	-	-	-	-	-	DECIMAL	19	4	-
nchar	1	254	-	-	-	-	CHAR	-	-	N
nchar	255	32672	-	-	-	-	VARCHAR	-	-	N
nchar null (see nvarchar)										
numeric	1	31	0	31	-	-	DECIMAL	-	-	-
numeric	32	38	0	38	-	-	DOUBLE	-	-	-

Table 101. Sybase CTLIB default forward data type mappings (continued)

Remote Typename	Remote Lower Len	Remote Upper Len	Remote Lower Scale	Remote Upper Scale	Remote Bit Data	Remote Data Operators	Federated Typename	Federated Length	Federated Scale	Federated Bit Data
numericn	1	31	0	31	-	-	DECIMAL	-	-	-
numericn	32	38	0	38	-	-	DOUBLE	-	-	-
nvarchar	1	32672	-	-	-	-	VARCHAR	-	-	N
real	-	-	-	-	-	-	REAL	-	-	-
smalldatetime	-	-	-	-	-	-	TIMESTAMP(6)	-	-	-
smallint	-	-	-	-	-	-	SMALLINT	-	-	-
smallmoney	-	-	-	-	-	-	DECIMAL	10	4	-
sysname	-	-	-	-	-	-	VARCHAR	30	-	N
text	-	-	-	-	-	-	CLOB	-	-	-
time	-	-	-	-	-	-	TIME	-	-	-
timestamp	-	-	-	-	-	-	VARCHAR	8	-	Y
tinyint	-	-	-	-	-	-	SMALLINT	-	-	-
unichar[2]	1	254	-	-	-	-	CHAR	-	-	N
unichar[2]	255	32672	-	-	-	-	VARCHAR	-	-	N
unichar null (see univarchar)										
univarchar[2]	1	32672	-	-	-	-	VARCHAR	-	-	N
varbinary	1	32672	-	-	-	-	VARCHAR	-	-	Y
varchar	1	32672	-	-	-	-	VARCHAR	-	-	N

Note:

1. The federated type is TIMESTAMP(0) if the date_compat configuration parameter is set to ON.
2. Valid for non-Unicode federated databases.

Default forward data type mappings for Teradata data sources

The following table lists the default forward data type mappings for Teradata data sources.

Table 102. Teradata default forward data type mappings (Not all columns shown)

Remote Typename	Remote Lower Len	Remote Upper Len	Remote Lower Scale	Remote Upper Scale	Remote Bit Data	Remote Data Operators	Federated Typename	Federated Length	Federated Scale	Federated Bit Data
BIGINT[1]	-	-	-	-	-	-	BIGINT[1]	-	-	-
BLOB	1	2097088000	-	-	-	-	BLOB	-	-	-
BYTE	1	254	-	-	-	-	CHAR	-	-	Y
BYTE	255	32672	-	-	-	-	VARCHAR	-	-	Y
BYTE	32673	64000	-	-	-	-	BLOB	-	-	-
BYTEINT	-	-	-	-	-	-	SMALLINT	-	-	-
CHAR	1	254	-	-	-	-	CHARACTER	-	-	-
CHAR	255	32672	-	-	-	-	VARCHAR	-	-	-
CHAR	32673	64000	-	-	-	-	CLOB	-	-	-
CLOB	1	2097088000 (Latin)	-	-	-	-	CLOB	-	-	-
CLOB	1	1048544000 (Unicode)	-	-	-	-	CLOB	-	-	-
DATE	-	-	-	-	-	-	DATE	-	-	-

Table 102. Teradata default forward data type mappings (Not all columns shown) (continued)

Remote Typename	Remote Lower Len	Remote Upper Len	Remote Lower Scale	Remote Upper Scale	Remote Bit Data	Remote Data Operators	Federated Typename	Federated Length	Federated Scale	Federated Bit Data
DATE	-	-	-	-	-	-	TIMESTAMP[2]	-	-	-
DECIMAL	1	31[3]	0	31[3]	-	-	DECIMAL	-	-	-
DOUBLE PRECISION	-	-	-	-	-	-	DOUBLE	-	-	-
FLOAT	-	-	-	-	-	-	DOUBLE	-	-	-
GRAPHIC	1	127	-	-	-	-	GRAPHIC	-	-	-
GRAPHIC	128	16336	-	-	-	-	VARGRAPHIC	-	-	-
GRAPHIC	16337	32000	-	-	-	-	DBCLOB	-	-	-
INTEGER	-	-	-	-	-	-	INTEGER	-	-	-
INTERVAL	-	-	-	-	-	-	CHAR	-	-	-
NUMERIC	1	18	0	18	-	-	DECIMAL	-	-	-
REAL	-	-	-	-	-	-	DOUBLE	-	-	-
SMALLINT	-	-	-	-	-	-	SMALLINT	-	-	-
TIME	0	21	0	21	-	-	TIME	-	-	-
TIMESTAMP(*p*)	-	-	*p*	*p*	-	-	TIMESTAMP(6)	10	6	-
VARBYTE	1	32762	-	-	-	-	VARCHAR	-	-	Y
VARBYTE	32763	64000	-	-	-	-	BLOB	-	-	-
VARCHAR	1	32672	-	-	-	-	VARCHAR	-	-	-
VARCHAR	32673	64000	-	-	-	-	CLOB	-	-	-
VARGRAPHIC	1	16336	-	-	-	-	VARGRAPHIC	-	-	-
VARGRAPHIC	16337	32000	-	-	-	-	DBCLOB	-	-	-

Notes:

1. The BIGINT data type applies to Teradata version 2 release 6 or later.
2. The federated type is TIMESTAMP(0) if the date_compat configuration parameter is set to ON.
3. The remote upper length and remote upper scale value 31 applies to the Teradata version 2 release 6 or later. For earlier versions, this value is 18.

Sample forward data type mappings

You can use the sample forward type mappings to take advantage of support for the TIMESTAMP data type with a precision.

For Informix data sources, these type mappings are used for nickname column types, federated procedures parameters, passthru, and federated procedure result sets.

For data sources other than Informix, these type mappings affect only the mappings for nickname column types and federated procedure parameters. The mappings do not affect passthru and federated procedure result sets.

Forward data type mappings - Informix example

When creating federated objects, you can use the sample forward type mapping provided for Informix.

You need to create these mappings before you create a federated object.

```
CREATE TYPE MAPPING FROM LOCAL TYPE timestamp(0)
   TO SERVER TYPE informix REMOTE TYPE datetime(0,10);
```

```
CREATE TYPE MAPPING FROM LOCAL TYPE timestamp(1)
   TO SERVER TYPE informix REMOTE TYPE datetime(0,11);

CREATE TYPE MAPPING FROM LOCAL TYPE timestamp(2)
   TO SERVER TYPE informix REMOTE TYPE datetime(0,12);

CREATE TYPE MAPPING FROM LOCAL TYPE timestamp(3)
   TO SERVER TYPE informix REMOTE TYPE datetime(0,13);

CREATE TYPE MAPPING FROM LOCAL TYPE timestamp(4)
   TO SERVER TYPE informix REMOTE TYPE datetime(0,14);

CREATE TYPE MAPPING FROM LOCAL TYPE timestamp(5)
   TO SERVER TYPE informix REMOTE TYPE datetime(0,15);
```

Forward data type mappings - Microsoft SQL Server example

When creating federated objects, you can use the sample forward type mapping provided for Microsoft SQL Server.

You need to create these mappings before you create a nickname or federated procedure.

```
CREATE TYPE MAPPING FROM LOCAL TYPE timestamp(3)
   TO SERVER TYPE mssqlserver REMOTE TYPE "datetime";

CREATE TYPE MAPPING FROM LOCAL TYPE timestamp(0)
   TO SERVER TYPE mssqlserver REMOTE TYPE "smalldatetime";
```

Forward data type mappings - Oracle example

When creating federated objects, you can use the sample forward type mapping provided for Oracle.

You need to create these mappings before you create a nickname or federated procedure.

```
CREATE TYPE MAPPING FROM LOCAL TYPE timestamp(0)
   TO SERVER TYPE oracle REMOTE TYPE timestamp(0);

CREATE TYPE MAPPING FROM LOCAL TYPE timestamp(1)
   TO SERVER TYPE oracle REMOTE TYPE timestamp(1);

CREATE TYPE MAPPING FROM LOCAL TYPE timestamp(2)
   TO SERVER TYPE oracle REMOTE TYPE timestamp(2);

CREATE TYPE MAPPING FROM LOCAL TYPE timestamp(3)
   TO SERVER TYPE oracle REMOTE TYPE timestamp(3);

CREATE TYPE MAPPING FROM LOCAL TYPE timestamp(4)
   TO SERVER TYPE oracle REMOTE TYPE timestamp(4);

CREATE TYPE MAPPING FROM LOCAL TYPE timestamp(5)
   TO SERVER TYPE oracle REMOTE TYPE timestamp(5);

CREATE TYPE MAPPING FROM LOCAL TYPE timestamp(7)
   TO SERVER TYPE oracle REMOTE TYPE timestamp(7);

CREATE TYPE MAPPING FROM LOCAL TYPE timestamp(8)
   TO SERVER TYPE oracle REMOTE TYPE timestamp(8);

CREATE TYPE MAPPING FROM LOCAL TYPE timestamp(9)
   TO SERVER TYPE oracle REMOTE TYPE timestamp(9);

CREATE TYPE MAPPING FROM LOCAL TYPE timestamp(0)
   TO SERVER TYPE oracle REMOTE TYPE date;
```

Forward data type mappings - Sybase example

When creating federated objects, you can use the sample forward type mapping provided for Sybase.

You need to create these mappings before you create a nickname or federated procedure.

```
CREATE TYPE MAPPING FROM LOCAL TYPE timestamp(3)
   TO SERVER TYPE sybase REMOTE TYPE datetime);

CREATE TYPE MAPPING FROM LOCAL TYPE timestamp(0)
   TO SERVER TYPE sybase REMOTE TYPE smalldatetime);
```

Forward data type mappings - Teradata example

When creating federated objects, you can use the sample forward type mapping provided for Teradata.

You need to create these mappings before you create a nickname or federated procedure.

```
CREATE TYPE MAPPING FROM LOCAL TYPE timestamp(0)
   TO SERVER TYPE teradata REMOTE TYPE timestamp(0);

CREATE TYPE MAPPING FROM LOCAL TYPE timestamp(1)
   TO SERVER TYPE teradata REMOTE TYPE timestamp(1);

CREATE TYPE MAPPING FROM LOCAL TYPE timestamp(2)
   TO SERVER TYPE teradata REMOTE TYPE timestamp(2);

CREATE TYPE MAPPING FROM LOCAL TYPE timestamp(3)
   TO SERVER TYPE teradata REMOTE TYPE timestamp(3);

CREATE TYPE MAPPING FROM LOCAL TYPE timestamp(4)
   TO SERVER TYPE teradata REMOTE TYPE timestamp(4);

CREATE TYPE MAPPING FROM LOCAL TYPE timestamp(5)
   TO SERVER TYPE teradata REMOTE TYPE timestamp(5);
```

Default reverse data type mappings

For most data sources, the default type mappings are in the wrappers.

The two kinds of mappings between data source data types and federated database data types are forward type mappings and reverse type mappings. In a forward type mapping, the mapping is from a remote type to a comparable local type. The other type of mapping is a reverse type mapping, which is used with transparent DDL to create or modify remote tables.

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

When you define a remote table or view to the federated database, the definition includes a reverse type mapping. The mapping is from a local federated database data type for each column, and the corresponding remote data type. For example, there is a default reverse type mapping in which the local type REAL points to the Informix type SMALLFLOAT.

Federated databases do not support mappings for LONG VARCHAR, LONG VARGRAPHIC, and user-defined types.

When you use the CREATE TABLE statement to create a remote table, you specify the local data types you want to include in the remote table. These default reverse type mappings will assign corresponding remote types to these columns. For example, suppose that you use the CREATE TABLE statement to define an Informix table with a column C2. You specify BIGINT as the data type for C2 in the statement. The default reverse type mapping of BIGINT depends on which version of Informix you are creating the table on. The mapping for C2 in the Informix table will be to DECIMAL in Informix Version 8 and to INT8 in Informix Version 9.

You can override a default reverse type mapping, or create a new reverse type mapping with the CREATE TYPE MAPPING statement.

The following tables show the default reverse mappings between federated database local data types and remote data source data types.

These mappings are valid with all the supported versions, unless otherwise noted.

Default reverse data type mappings for DB2 Database for Linux, UNIX, and Windows data sources

The following table lists the default reverse data type mappings for DB2 Database for Linux, UNIX, and Windows data sources.

Table 103. DB2 Database for Linux, UNIX, and Windows default reverse data type mappings (Not all columns shown)

Federated Typename	Federated Lower Len	Federated Upper Len	Federated Lower Scale	Federated Upper Scale	Federated Bit Data	Federated Data Operators	Remote Typename	Remote Length	Remote Scale	Federated Bit Data
BIGINT	-	8	-	-	-	-	BIGINT	-	-	-
BLOB	-	-	-	-	-	-	BLOB	-	-	-
CHARACTER	-	-	-	-	-	-	CHAR	-	-	N
CHARACTER	-	-	-	-	Y	-	CHAR	-	-	Y
CLOB	-	-	-	-	-	-	CLOB	-	-	-
DATE[1]	-	4	-	-	-	-	DATE	-	-	-
DBCLOB	-	-	-	-	-	-	DBCLOB	-	-	-
DECIMAL	-	-	-	-	-	-	DECIMAL	-	-	-
DECFLOAT [2]	-	8	-	-	-	-	DECFLOAT	-	0	-
DECFLOAT [2]	-	16	-	-	-	-	DECFLOAT	-	0	-
DOUBLE	-	8	-	-	-	-	DOUBLE	-	-	-
FLOAT	-	8	-	-	-	-	DOUBLE	-	-	-
GRAPHIC	-	-	-	-	-	-	GRAPHIC	-	-	N
INTEGER	-	4	-	-	-	-	INTEGER	-	-	-
REAL	-	-	-	-	-	-	REAL	-	-	-
SMALLINT	-	2	-	-	-	-	SMALLINT	-	-	-
TIME	-	3	-	-	-	-	TIME	-	-	-
TIMESTAMP(*p*)	-	-	*p*	*p*	-	-	TIMESTAMP(*p*)	-	*p*[3]	-
VARCHAR	-	-	-	-	-	-	VARCHAR	-	-	N
VARCHAR	-	-	-	-	Y	-	VARCHAR	-	-	Y
VARGRAPH	-	-	-	-	-	-	VARGRAPHIC	-	-	N
VARGRAPHIC	-	-	-	-	-	-	VARGRAPHIC	-	-	-

Table 103. DB2 Database for Linux, UNIX, and Windows default reverse data type mappings (Not all columns shown) (continued)

Federated Typename	Federated Lower Len	Federated Upper Len	Federated Lower Scale	Federated Upper Scale	Federated Bit Data	Federated Data Operators	Remote Typename	Remote Length	Remote Scale	Federated Bit Data

Note:

1. When the date_compat parameter is set to OFF, the federated DATE maps to TIMESTAMP(0).
2. The SAME_DECFLT_ROUNDING server option is set to N by default and operations will not be pushed down to the remote data source unless SAME_DECFLT_ROUNDING is set to Y. For information on the SAME_DECFLT_ROUNDING server option, see DB2 database options reference.
3. For Version 9.5 or earlier, the remote scale for TIMESTAMP is 6.

Default reverse data type mappings for DB2 for System i data sources

The following table lists the default reverse data type mappings for DB2 for System i data sources.

Table 104. DB2 for System i default reverse data type mappings (Not all columns shown)

Federated Typename	Federated Lower Len	Federated Upper Len	Federated Lower Scale	Federated Upper Scale	Federated Bit Data	Federated Data Operators	Remote Typename	Remote Length	Remote Scale	Remote Bit Data
BLOB	-	-	-	-	-	-	BLOB	-	-	-
CHARACTER	-	-	-	-	-	-	CHARACTER	-	-	N
CHARACTER	-	-	-	-	Y	-	CHARACTER	-	-	Y
CLOB	-	-	-	-	-	-	CLOB	-	-	-
DATE	-	4	-	-	-	-	DATE	-	-	-
DBCLOB	-	-	-	-	-	-	DBCLOB	-	-	-
DECIMAL	-	-	-	-	-	-	NUMERIC	-	-	-
DECIMAL	-	-	-	-	-	-	DECIMAL	-	-	-
DOUBLE	-	8	-	-	-	-	FLOAT	-	-	-
GRAPHIC	-	-	-	-	-	-	GRAPHIC	-	-	N
INTEGER	-	4	-	-	-	-	INTEGER	-	-	-
REAL	-	4	-	-	-	-	FLOAT	-	-	-
SMALLINT	-	2	-	-	-	-	SMALLINT	-	-	-
TIME	-	3	-	-	-	-	TIME	-	-	-
TIMESTAMP*(p)*	-	-	*p*	*p*	-	-	TIMESTAMP	-	-	-
VARCHAR	-	-	-	-	-	-	VARCHAR	-	-	N
VARCHAR	-	-	-	-	Y	-	VARCHAR	-	-	Y
VARGRAPHIC	-	-	-	-	-	-	VARG	-	-	N

Default reverse data type mappings for DB2 for VM and VSE data sources

The following table lists the default reverse data type mappings for DB2 for VM and VSE data sources.

Table 105. DB2 for VM and VSE default reverse data type mappings (Not all columns shown)

Federated Typename	Federated Lower Len	Federated Upper Len	Federated Lower Scale	Federated Upper Scale	Federated Bit Data	Federated Data Operators	Remote Typename	Remote Length	Remote Scale	Remote Bit Data
BLOB	-	-	-	-	-	-	BLOB	-	-	-

Table 105. DB2 for VM and VSE default reverse data type mappings (Not all columns shown) (continued)

Federated Typename	Federated Lower Len	Federated Upper Len	Federated Lower Scale	Federated Upper Scale	Federated Bit Data	Federated Data Operators	Remote Typename	Remote Length	Remote Scale	Remote Bit Data
CHARACTER	-	-	-	-	-	-	CHAR	-	-	-
CHARACTER	-	-	-	-	Y	-	CHAR	-	-	Y
CLOB	-	-	-	-	-	-	CLOB	-	-	-
DATE	-	4	-	-	-	-	DATE	-	-	-
DBCLOB	-	-	-	-	-	-	DBCLOB	-	-	-
DECIMAL	-	-	-	-	-	-	DECIMAL	-	-	-
DOUBLE	-	8	-	-	-	-	FLOAT	-	-	-
GRAPHIC	-	-	-	-	-	-	GRAPHIC	-	-	N
INTEGER	-	4	-	-	-	-	INTEGER	-	-	-
REAL	-	4	-	-	-	-	REAL	-	-	-
SMALLINT	-	2	-	-	-	-	SMALLINT	-	-	-
TIME	-	3	-	-	-	-	TIME	-	-	-
TIMESTAMP*(p)*	-	-	*p*	*p*	-	-	TIMESTAMP	-	-	-
VARCHAR	-	-	-	-	-	-	VARCHAR	-	-	-
VARCHAR	-	-	-	-	Y	-	VARCHAR	-	-	Y
VARGRAPH	-	-	-	-	-	-	VARGRAPH	-	-	N

Default reverse data type mappings for DB2 for z/OS data sources

The following table lists the default reverse data type mappings for DB2 for z/OS data sources.

Table 106. DB2 for z/OS default reverse data type mappings (Not all columns shown)

Federated Typename	Federated Lower Len	Federated Upper Len	Federated Lower Scale	Federated Upper Scale	Federated Bit Data	Federated Data Operators	Remote Typename	Remote Length	Remote Scale	Remote Bit Data
BIGINT	0	8	0	0	" "	"/0"	BIGINT	0	0	N
BLOB	-	-	-	-	-	-	BLOB	-	-	-
CHARACTER	-	-	-	-	-	-	CHAR	-	-	N
CHARACTER	-	-	-	-	Y	-	CHAR	-	-	Y
CLOB	-	-	-	-	-	-	CLOB	-	-	-
DATE	-	4	-	-	-	-	DATE	-	-	-
DBCLOB	-	-	-	-	-	-	DBCLOB	-	-	-
DECIMAL	-	-	-	-	-	-	DECIMAL	-	-	-
DOUBLE	-	8	-	-	-	-	DOUBLE	-	-	–
FLOAT	-	8	-	-	-	-	DOUBLE	-	-	-
GRAPHIC	-	-	-	-	-	-	GRAPHIC	-	-	N
INTEGER	-	4	-	-	-	-	INTEGER	-	-	-
REAL	-	4	-	-	-	-	REAL	-	-	-
SMALLINT	-	2	-	-	-	-	SMALLINT	-	-	-
TIME	-	3	-	-	-	-	TIME	-	-	-
TIMESTAMP*(p)*	-	-	*p*	*p*	-	-	TIMESTAMP	-	-	-
VARCHAR	-	-	-	-	-	-	VARCHAR	-	-	N
VARCHAR	-	-	-	-	Y	-	VARCHAR	-	-	Y

Table 106. DB2 for z/OS default reverse data type mappings (Not all columns shown) (continued)

Federated Typename	Federated Lower Len	Federated Upper Len	Federated Lower Scale	Federated Upper Scale	Federated Bit Data	Federated Data Operators	Remote Typename	Remote Length	Remote Scale	Remote Bit Data
VARGRAPHIC	-	-	-	-	-	-	VARGRAPHIC	-	-	N

Default reverse data type mappings for Informix data sources

The following table lists the default reverse data type mappings for Informix data sources.

Table 107. Informix default reverse data type mappings

Federated Typename	Federated Lower Len	Federated Upper Len	Federated Lower Scale	Federated Upper Scale	Federated Bit Data	Federated Data Operators	Remote Typename	Remote Length	Remote Scale	Remote Bit Data
BIGINT [1]	-	-	-	-	-	-	DECIMAL	19	-	-
BIGINT [2]	-	-	-	-	-	-	INT8	-	-	-
BLOB	1	2147483647	-	-	-	-	BYTE	-	-	-
CHARACTER	-	-	-	-	N	-	CHAR	-	-	-
CHARACTER	-	-	-	-	Y	-	BYTE	-	-	-
CLOB	1	2147483647	-	-	-	-	TEXT	-	-	-
DATE	-	4	-	-	-	-	DATE	-	-	-
DECIMAL	-	-	-	-	-	-	DECIMAL	-	-	-
DOUBLE	-	8	-	-	-	-	FLOAT	-	-	-
INTEGER	-	4	-	-	-	-	INTEGER	-	-	-
REAL	-	4	-	-	-	-	SMALLFLOAT	-	-	-
SMALLINT	-	2	-	-	-	-	SMALLINT	-	-	-
TIME	-	3	-	-	-	-	DATETIME	6	10	-
TIMESTAMP	-	10	-	-	-	-	DATETIME	0	15	-
VARCHAR	1	254	-	-	N	-	VARCHAR	-	-	-
VARCHAR [1]	255	32672	-	-	N	-	TEXT	-	-	-
VARCHAR	-	-	-	-	Y	-	BYTE	-	-	-
VARCHAR [2]	255	2048	-	-	N	-	LVARCHAR	-	-	-
VARCHAR [2]	2049	32672	-	-	N	-	TEXT	-	-	-

Note:

1. This type mapping is valid only with Informix server Version 8 (or lower).
2. This type mapping is valid only with Informix server Version 9 (or higher).

For the Informix DATETIME data type, the federated server uses the Informix high-level qualifier as the REMOTE_LENGTH and the Informix low-level qualifier as the REMOTE_SCALE.

The Informix qualifiers are the "TU_" constants defined in the Informix Client SDK `datatime.h` file. The constants are:

0 = YEAR	8 = MINUTE	13 = FRACTION(3)
2 = MONTH	10 = SECOND	14 = FRACTION(4)
4 = DAY	11 = FRACTION(1)	15 = FRACTION(5)
6 = HOUR	12 = FRACTION(2)	

Default reverse data type mappings for JDBC data sources

The following table lists the default reverse data type mappings for JDBC data sources that conform to the type mappings of the DB2 JDBC driver.

Table 108. JDBC default reverse data type mappings

Federated Typename	Federated Lower Len	Federated Upper Len	Federated Lower Scale	Federated Upper Scale	Federated Bit Data	Federated Data Operators	Remote Typename	Remote Length	Remote Scale	Remote Bit Data
BIGINT	-	-	-	-	-	-	BIGINT	-	-	-
BLOB	-	-	-	-	-	-	BLOB	-	-	-
CHAR	-	-	-	-	Y	-	BINARY	-	-	-
CHAR	-	-	-	-	-	-	CHAR	-	-	-
CLOB	-	-	-	-	-	-	CLOB	-	-	-
DATE	-	4	-	-	-	-	DATE	-	-	-
DBCLOB	-	-	-	-	-	-	NCLOB	-	-	-
DECIMAL	-	-	-	-	-	-	DECIMAL	-	-	-
DOUBLE	-	8	-	-	-	-	DOUBLE	-	-	-
GRAPHIC	-	-	-	-	-	-	NCHAR	-	-	-
INTEGER	-	-	-	-	-	-	INTEGER	-	-	-
REAL	-	4	-	-	-	-	REAL	-	-	-
SMALLINT	-	-	-	-	-	-	SMALLINT	-	-	-
TIME	-	3	-	-	-	-	TIME	-	-	-
TIMESTAMP	-	-	-	-	-	-	TIMESTAMP	-	-	-
TIMESTAMP(*p*)	-	-	*p*	*p*	-	-	TIMESTAMP(*p*)	-	min(9,*p*)	-
VARCHAR	-	-	-	-	Y	-	VARBINARY	-	-	-
VARCHAR	-	-	-	-	N	-	VARCHAR	-	-	-
VARGRAPHIC	-	-	-	-	-	-	NVARCHAR	-	-	-

Default reverse data type mappings for Microsoft SQL Server data sources

The following table lists the default reverse data type mappings for Microsoft SQL Server data sources.

Table 109. Microsoft SQL Server default reverse data type mappings (Not all columns shown)

Federated Typename	Federated Lower Len	Federated Upper Len	Federated Lower Scale	Federated Upper Scale	Federated Bit Data	Federated Data Operators	Remote Typename	Remote Length	Remote Scale	Remote Bit Data
BIGINT	-	-	-	-	-	-	bigint	-	-	-
BLOB	-	-	-	-	-	-	image	-	-	-
CHARACTER	-	-	-	-	Y	-	binary	-	-	-
CHARACTER	-	-	-	-	N	-	char	-	-	-
CLOB	-	-	-	-	-	-	text	-	-	-
DATE	-	4	-	-	-	-	datetime	-	-	-
DECIMAL	-	-	-	-	-	-	decimal	-	-	-
DOUBLE	-	8	-	-	-	-	float	-	-	-
INTEGER	-	-	-	-	-	-	int	-	-	-
SMALLINT	-	-	-	-	-	-	smallint	-	-	-
REAL	-	4	-	-	-	-	real	-	-	-
TIME	-	3	-	-	-	-	datetime	-	-	-
TIMESTAMP	-	10	-	-	-	-	datetime	-	-	-
VARCHAR	1	8000	-	-	N	-	varchar	-	-	-
VARCHAR	8001	32672	-	-	N	-	text	-	-	-

Table 109. Microsoft SQL Server default reverse data type mappings (Not all columns shown) (continued)

Federated Typename	Federated Lower Len	Federated Upper Len	Federated Lower Scale	Federated Upper Scale	Federated Bit Data	Federated Data Operators	Remote Typename	Remote Length	Remote Scale	Remote Bit Data
VARCHAR	1	8000	-	-	Y	-	varbinary	-	-	-
VARCHAR	8001	32672	-	-	Y	-	image	-	-	-

Default reverse data type mappings for ODBC data sources

The following table lists the default reverse data type mappings for ODBC data sources.

Table 110. ODBC default reverse data type mappings (Not all columns shown)

Federated Typename	Federated Lower Len	Federated Upper Len	Federated Lower Scale	Federated Upper Scale	Federated Bit Data	Federated Data Operators	Remote Typename	Remote Length	Remote Scale	Remote Bit Data
BIGINT	-	-	-	-	-	-	SQL_BIGINT	-	-	-
BLOB	-	-	-	-	-	-	SQL_LONGVARBINARY	-	-	-
CHAR	-	-	-	-	Y	-	SQL_BINARY	-	-	-
CHAR	-	-	-	-	N	-	SQL_CHAR	-	-	-
CLOB	-	-	-	-	-	-	SQL_LONGVARCHAR	-	-	-
DATE	-	4	-	-	-	-	SQL_TYPE_DATE	-	-	-
DBCLOB	-	-	-	-	-	-	SQL_WLONGVARCHAR	-	-	-
DECIMAL	-	-	-	-	-	-	SQL_DECIMAL	-	-	-
DOUBLE	-	8	-	-	-	-	SQL_DOUBLE	-	-	-
FLOAT	-	-	-	-	-	-	SQL_FLOAT	-	-	-
GRAPHIC	-	-	-	-	-	-	SQL_WCHAR	-	-	-
INTEGER	-	-	-	-	-	-	SQL_INTEGER	-	-	-
REAL	-	4	-	-	-	-	SQL_REAL	-	-	-
SMALLINT	-	-	-	-	-	-	SQL_SMALLINT	-	-	-
TIME	-	3	-	-	-	-	SQL_TYPE_TIME	-	-	-
TIMESTAMP	-	10	-	-	-	-	SQL_TYPE_TIMESTAMP(*p*)	-	-	-
VARCHAR	-	-	-	-	Y	-	SQL_VARBINARY	-	-	-
VARCHAR	-	-	-	-	N	-	SQL_VARCHAR	-	-	-
VARGRAPHIC	-	-	-	-	Y	-	SQL_WVARCHAR	-	-	-

Default reverse data type mappings for Oracle NET8 data sources

The following table lists the default reverse data type mappings for Oracle NET8 data sources.

Table 111. Oracle NET8 default reverse data type mappings

Federated Typename	Federated Lower Len	Federated Upper Len	Federated Lower Scale	Federated Upper Scale	Federated Bit Data	Federated Data Operators	Remote Typename	Remote Length	Remote Scale	Remote Bit Data
BIGINT	0	8	0	0	N	\0	NUMBER	19	0	N
BLOB	0	2147483647	0	0	Y	\0	BLOB	0	0	Y
CHARACTER	1	254	0	0	N	\0	CHAR	0	0	N
CHARACTER	1	254	0	0	Y	\0	RAW	0	0	Y
CLOB	0	2147483647	0	0	N	\0	CLOB	0	0	N

Table 111. Oracle NET8 default reverse data type mappings (continued)

Federated Typename	Federated Lower Len	Federated Upper Len	Federated Lower Scale	Federated Upper Scale	Federated Bit Data	Federated Data Operators	Remote Typename	Remote Length	Remote Scale	Remote Bit Data
DATE[1]	0	4	0	0	N	\0	DATE	0	0	N
DECIMAL	0	0	0	0	N	\0	NUMBER	0	0	N
DECFLOAT	0	8	0	0	N	\0	NUMBER	0	0	N
DECFLOAT	0	16	0	0	N	\0	NUMBER	0	0	N
DOUBLE	0	8	0	0	N	\0	FLOAT	126	0	N
FLOAT	0	8	0	0	N	\0	FLOAT	126	0	N
INTEGER	0	4	0	0	N	\0	NUMBER	10	0	N
REAL	0	4	0	0	N	\0	FLOAT	63	0	N
SMALLINT	0	2	0	0	N	\0	NUMBER	5	0	N
TIME	0	3	0	0	N	\0	DATE	0	0	N
TIMESTAMP [2]	0	10	0	0	N	\0	DATE	0	0	N
TIMESTAMP*(p)* [3]	-	-	-	-	N	\0	TIMESTAMP*(p)*	-	-	N
VARCHAR	1	4000	0	0	N	\0	VARCHAR2	0	0	N
VARCHAR	1	2000	0	0	Y	\0	RAW	0	0	Y

Note:

1. When the date_compat parameter is set to OFF, the federated DATE maps to the Oracle date. When the date_compat parameter is set to ON, the federated DATE (equivalent to TIMESTAMP(0)), maps to Oracle TIMESTAMP(0).
2. This type mapping is valid only with Oracle Version 8.
3.
   - TIMESTAMP*(p)* represents a timestamp with a variable scale from 0-9 for Oracle and 0-12 for federation. When the scale is 0-9, the remote Oracle TIMESTAMP has the same scale as the federated TIMESTAMP. If the scale of the federated TIMESTAMP is greater than 9, the corresponding scale of the Oracle TIMESTAMP is 9 which is the largest Oracle scale.
   - This type mapping is valid only with Oracle Version 9, 10, and 11.

Default reverse data type mappings for Sybase data sources

The following table lists the default reverse data type mappings for Sybase data sources.

Table 112. Sybase CTLIB default reverse data type mappings

Federated Typename	Federated Lower Len	Federated Upper Len	Federated Lower Scale	Federated Upper Scale	Federated Bit Data	Federated Data Operators	Remote Typename	Remote Length	Remote Scale	Remote Bit Data
BIGINT	-	-	-	-	-	-	decimal	19	0	-
BLOB	-	-	-	-	-	-	image	-	-	-
CHARACTER	-	-	-	-	N	-	char	-	-	-
CHARACTER	-	-	-	-	Y	-	binary	-	-	-
CLOB	-	-	-	-	-	-	text	-	-	-
DATE	-	-	-	-	-	-	datetime	-	-	-
DECIMAL	-	-	-	-	-	-	decimal	-	-	-
DOUBLE	-	-	-	-	-	-	float	-	-	-
INTEGER	-	-	-	-	-	-	integer	-	-	-
REAL	-	-	-	-	-	-	real	-	-	-
SMALLINT	-	-	-	-	-	-	smallint	-	-	-
TIME	-	-	-	-	-	-	datetime	-	-	-
TIMESTAMP	-	-	-	-	-	-	datetime	-	-	-
VARCHAR[1]	1	255	-	-	N	-	varchar	-	-	-

Table 112. Sybase CTLIB default reverse data type mappings (continued)

Federated Typename	Federated Lower Len	Federated Upper Len	Federated Lower Scale	Federated Upper Scale	Federated Bit Data	Federated Data Operators	Remote Typename	Remote Length	Remote Scale	Remote Bit Data
VARCHAR[1]	256	32672	-	-	N	-	text	-	-	-
VARCHAR [2]	1	16384	-	-	N	-	varchar	-	-	-
VARCHAR [2]	16385	32672	-	-	N	-	text	-	-	-
VARCHAR[1]	1	255	-	-	Y	-	varbinary	-	-	-
VARCHAR[1]	256	32672	-	-	Y	-	image	-	-	-
VARCHAR [2]	1	16384	-	-	Y	-	varbinary	-	-	-
VARCHAR [2]	16385	32672	-	-	Y	-	image	-	-	-

Note:

1. This type mapping is valid only for CTLIB with Sybase server version 12.0 (or earlier).
2. This type mapping is valid only for CTLIB with Sybase server version 12.5 (or later).

Default reverse data type mappings for Teradata data sources

The following table lists the default reverse data type mappings for Teradata data sources.

Table 113. Teradata default reverse data type mappings (Not all columns shown)

Federated Typename	Federated Lower Len	Federated Upper Len	Federated Lower Scale	Federated Upper Scale	Federated Bit Data	Federated Data Operators	Remote Typename	Remote Length	Remote Scale	Remote Bit Data
BIGINT[1]	-	-	-	-	-	-	BIGINT	-	-	-
BLOB	1	2097088000	-	-	-	-	BLOB	-	-	-
CHARACTER	-	-	-	-	-	-	CHARACTER	-	-	-
CHARACTER	-	-	-	-	Y	-	BYTE	-	-	-
CLOB	1	2097088000	-	-		-	CLOB	-	-	-
DATE	-	-	-	-	-	-	DATE	-	-	-
DBCLOB [2]	1	64000	-	-	-	-	VARGRAPHIC	-	-	-
DECIMAL	1	18/31[3]	0	18/31[3]	-	-	DECIMAL	-	-	-
DECIMAL[4]	19	31	0	31	-	-	FLOAT	8	-	-
DOUBLE	-	-	-	-	-	-	FLOAT	-	-	-
GRAPHIC	-	-	-	-	-	-	GRAPHIC	-	-	-
INTEGER	-	-	-	-	-	-	INTEGER	-	-	-
REAL	-	-	-	-	-	-	FLOAT	8	-	-
SMALLINT	-	-	-	-	-	-	SMALLINT	-	-	-
TIME	-	-	-	-	-	-	TIME	15	-	-
TIMESTAMP	10	10	6	6	-	-	TIMESTAMP	26	6	-
VARCHAR	-	-	-	-	-	-	VARCHAR	-	-	-
VARCHAR	-	-	-	-	Y	-	VARBYTE	-	-	-
VARGRAPHIC	-	-	-	-	-	-	VARGRAPHIC	-	-	-

Notes:

1. The BIGINT data type applies to Teradata version 2 release 6 or later.
2. The Teradata VARGRAPHIC data type can contain only the specified length (1 to 32000) of a DBCLOB data type.
3. The value 31 applies to Teradata version 2 release 6 or later. The value 18 applies to earlier versions.
4. This DECIMAL row applies Teradata version 2 release 5 or earlier.

Unicode default data type mappings

Specific data sources support forward data type mappings and reverse data type mappings for Unicode databases.

Unicode default forward data type mappings for JDBC data sources

The following table lists the default forward data type mapping for JDBC data sources when the federated database is a Unicode database.

Table 114. Unicode default forward data type mappings for JDBC data sources

UTF-8	**JDBC**	
Data type	**Data type**	**Length**
CHAR	CHAR	1 to 254 bytes
VARCHAR	VARCHAR	1 to 32672 bytes
CLOB	CLOB	-
GRAPHIC	NCHAR	1 to 127 characters
VARGRAPHIC	NVARCHAR	1 to 16336 characters
DBCLOB	NCLOB	-

Unicode default reverse data type mappings for JDBC data sources

The following table lists the default reverse data type mapping for JDBC data sources when the federated database is a Unicode database.

Table 115. Unicode default reverse data type mappings for JDBC data sources

UTF-8		**JDBC**
Data type	**Length**	**Data type**
CHAR	1 to 254 bytes	CHAR
VARCHAR	1 to 32672 bytes	VARCHAR
CLOB	1 to 2147483647 bytes	CLOB
GRAPHIC	1 to 127 characters	NCHAR
VARGRAPHIC	1 to 16336 characters	NVARCHAR
DBCLOB	1 to 1073741823 characters	NCLOB

Unicode default forward data type mappings for Microsoft SQL Server data sources

The following table lists the default forward data type mapping for Microsoft SQL Server data sources when the federated database is a Unicode database.

Table 116. Unicode default forward data type mappings for Microsoft SQL Server data sources

UTF-8	**Microsoft SQL Server**	
Data type	**Data type**	**Length**
CHAR	CHAR	1 to 254 bytes

Table 116. Unicode default forward data type mappings for Microsoft SQL Server data sources (continued)

UTF-8	Microsoft SQL Server	
Data type	**Data type**	**Length**
VARCHAR	CHAR	255 to 8000 bytes
	VARCHAR	1 to 8000 bytes
CLOB	TEXT	-
GRAPHIC	NCHAR	1 to 127 characters
VARGRAPHIC	NCHAR	128 to 16336 characters
	NVARCHAR	1 to 16336 characters
DBCLOB	NTEXT	-

Unicode default reverse data type mappings for Microsoft SQL Server data sources

The following table lists the default reverse data type mapping for Microsoft SQL Server data sources when the federated database is a Unicode database.

Table 117. Unicode default reverse data type mappings for Microsoft SQL Server data sources

UTF-8		Microsoft SQL Server
Data type	**Length**	**Data type**
CHAR	1 to 254 bytes	CHAR
VARCHAR	1 to 32672 bytes	VARCHAR
CLOB	1 to 2 147 483 647 bytes	TEXT
GRAPHIC	1 to 127 characters	NCHAR
VARGRAPHIC	1 to 16336 characters	NVARCHAR
DBCLOB	1 to 1 073 741 823 characters	NTEXT

Unicode default forward data type mappings for NET8 data sources

The following table lists the default forward data type mapping for NET8 data sources when the federated database is a Unicode database.

Table 118. Unicode default forward data type mappings for NET8 data sources

UTF-8	Oracle	
Data type	**Data type**	**Length**
CHAR	CHAR	1 to 254 bytes
VARCHAR	CHAR	255 to 2000 bytes
	VARCHAR2	1 to 4000 bytes
DBCLOB	NCLOB	
GRAPHIC	NCHAR	1 to 127 characters
VARGRAPHIC	NCHAR	128 to 1000 characters
	NVARCHAR2	1 to 2000 characters

Unicode default reverse data type mappings for NET8 data sources

The following table lists the default reverse data type mapping for NET8 data sources when the federated database is a Unicode database.

Table 119. Unicode default reverse data type mappings for NET8 data sources

UTF-8		Oracle
Data type	**Length**	**Data type**
CHAR	1 to 254 bytes	CHAR
VARCHAR	1 to 4000 bytes	VARCHAR2
CLOB	1 to 2 147 483 647 bytes	CLOB
GRAPHIC	1 to 127 characters	NCHAR
VARGRAPHIC	1 to 2000 characters	NVARCHAR2
DBCLOB	1 to 1 073 741 823 characters	NCLOB

Unicode default forward data type mappings for ODBC data sources

The following table lists the default forward data type mapping for ODBC data sources when the federated database is a Unicode database.

Table 120. Unicode default forward data type mappings for ODBC data sources

UTF-8	ODBC	
Data type	**Data type**	**Length**
CHAR	SQL_CHAR	1 to 254 bytes
VARCHAR	SQL_CHAR	255 to 32672 bytes
	SQL_VARCHAR	1 to 32672 bytes
CLOB	SQL_LONGVARCHAR	-
GRAPHIC	SQL_WCHAR	1 to 127 characters
VARGRAPHIC	SQL_WCHAR	128 to 16336 characters
	SQL_WVARCHAR	1 to 16336 characters
DBCLOB	SQL_WLONGVARCHAR	-

Unicode default reverse data type mappings for ODBC data sources

The following table lists the default reverse data type mapping for ODBC data sources when the federated database is a Unicode database.

Table 121. Unicode default reverse data type mappings for ODBC data sources

UTF-8		ODBC
Data type	**Length**	**Data type**
CHAR	1 to 254 bytes	SQL_CHAR
VARCHAR	1 to 32672 bytes	SQL_VARCHAR
CLOB	1 to 2 147 483 647 bytes	SQL_LONGVARCHAR
GRAPHIC	1 to 127 characters	SQL_WCHAR

Table 121. Unicode default reverse data type mappings for ODBC data sources (continued)

UTF-8		ODBC
Data type	**Length**	**Data type**
VARGRAPHIC	1 to 16336 characters	SQL_WVARCHAR
DBCLOB	1 to 1 073 741 823 characters	SQL_WLONGVARCHAR

Unicode default forward data type mappings for Sybase data sources

The following table lists the default forward data type mapping for Sybase CTLIB data sources when the federated database is a Unicode database.

Table 122. Unicode default forward data type mappings for Sybase CTLIB data sources

UTF-8	Sybase	
Data type	**Data type**	**Length**
CHAR	char	1 to 254 bytes
	nchar	1 to 127 characters
VARCHAR	char	255 to 32672 bytes
	varchar	1 to 32672 bytes
	nchar	128 to 16336 characters
	nvarchar	1 to 16336 characters
CLOB	text	
GRAPHIC	unichar	1 to 127 characters
VARGRAPHIC	unichar	128 to 16336 characters
	univarchar	1 to 16336 characters

Unicode default reverse data type mappings for Sybase data sources

The following table lists the default reverse data type mapping for Sybase CTLIB data sources when the federated database is a Unicode database.

Table 123. Unicode default reverse data type mappings for Sybase CTLIB data sources

UTF-8		Sybase
Data type	**Length**	**Data type**
CHAR	1 to 254 bytes	char
VARCHAR	1 to 32672 bytes	varchar
CLOB	1 to 2 147 483 647 bytes	text
GRAPHIC	1 to 127 characters	unichar
VARGRAPHIC	1 to 16336 characters	univarchar

Data types supported for nonrelational data sources

For most of the nonrelational data sources, you must specify the column information, including data type, when you create the nicknames to access the data source.

Some of the nonrelational wrappers create all of the columns required to access a data source. These are called *fixed columns*. Other wrappers let you specify some or all of the data types for the columns in the CREATE NICKNAME statement.

The following sections list the wrappers that you can specify the data types for, and the data types that the wrapper supports.

Data types supported by the BioRS wrapper

The following table lists the DB2 data types that the BioRS wrapper supports.

Table 124. BioRS data types that map to DB2 data types

BioRS data types	DB2 data type
AUTHOR	CHARACTER, CLOB, VARCHAR
DATE	CHARACTER, CLOB, VARCHAR
NUMBER	CHARACTER, CLOB, VARCHAR
REFERENCE	CHARACTER, CLOB, VARCHAR
TEXT	CHARACTER, CLOB, VARCHAR

The maximum length allowed for the CLOB data type is 5 megabytes.

Data types supported by the Excel wrapper

The following table lists the DB2 data types that the Excel wrapper supports.

Table 125. Excel data types that map to DB2 data types

Excel data types	DB2 data type
character	CHARACTER
date	DATE
number	DOUBLE
number	FLOAT
integer	INTEGER
character	VARCHAR

Data types supported by the Script wrapper

The following table lists the DB2 data types that the Script wrapper supports.

Table 126. Script data types that map to DB2 data types

XML data types	DB2 data type
character	BLOB
character	CHARACTER
character	CHARACTER FOR BIT DATA
character	CLOB (maximum length is 5 megabytes)
date	DATE
number	DECIMAL
number	DOUBLE
number	FLOAT

Table 126. Script data types that map to DB2 data types (continued)

XML data types	DB2 data type
integer	INTEGER
number	REAL
integer	SMALLINT
character	VARCHAR
character	VARCHAR FOR BIT DATA

Data types supported by the table-structured file wrapper

The following table lists the DB2 data types that the table-structured file wrapper supports.

Table 127. Table-structured file data types that map to DB2 data types

Table-structured file data types	DB2 data type
character	CHARACTER
character	CLOB (maximum length is 5 megabytes)
number	DECIMAL
number	DOUBLE
number	FLOAT
integer	INTEGER
number	REAL
integer	SMALLINT
character	VARCHAR

Data types supported by the Web services wrapper

The following table lists the DB2 data types that the Web services wrapper supports. The Web services wrapper uses XML data types.

Table 128. XML data types that map to DB2 data types for the Web services wrapper

XML data types	DB2 data type
character	BLOB
character	CHARACTER
character	CHARACTER FOR BIT DATA
character	CLOB (maximum length is 5 megabytes)
date	DATE
number	DECIMAL
number	DOUBLE
number	FLOAT
integer	INTEGER
number	REAL
integer	SMALLINT
character	VARCHAR
character	VARCHAR FOR BIT DATA

Data types supported by the XML wrapper

The following table lists the DB2 data types that the XML wrapper supports.

Table 129. XML data types that map to DB2 data types for the XML wrapper

XML data types	DB2 data type
character	BLOB
character	CHARACTER
character	CHARACTER FOR BIT DATA
character	CLOB (maximum length is 5 megabytes)
date	DATE
number	DECIMAL
number	DOUBLE
number	FLOAT
integer	INTEGER
number	REAL
integer	SMALLINT
character	VARCHAR
character	VARCHAR FOR BIT DATA
XML document	XML

Accessible documentation

Documentation is provided in XHTML format, which is viewable in most Web browsers.

XHTML allows you to view documentation according to the display preferences that you set in your browser. It also allows you to use screen readers and other assistive technologies.

Syntax diagrams are provided in dotted decimal format. This format is available only if you access the online documentation by using a screen reader.

Notices and trademarks

This information was developed for products and services offered in the U.S.A.

Notices

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 character set (DBCS) information, contact the IBM Intellectual Property Department in your country or send inquiries, in writing, to:

Intellectual Property Licensing
Legal and Intellectual Property Law
IBM Japan Ltd.
1623-14, Shimotsuruma, Yamato-shi
Kanagawa 242-8502 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
J46A/G4
555 Bailey Avenue
San Jose, CA 95141-1003 U.S.A.

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

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

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

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

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

This information is for planning purposes only. The information herein is subject to change before the products described become available.

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

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. The sample programs are provided "AS IS", without warranty of any kind. IBM shall not be liable for any damages arising out of your use of the sample programs.

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

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

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

Trademarks

IBM, the IBM logo, and ibm.com are trademarks of International Business Machines Corp., registered in many jurisdictions worldwide. Other product and service names might be trademarks of IBM or other companies. A current list of IBM trademarks is available on the Web at www.ibm.com/legal/copytrade.shtml.

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

Adobe is a registered trademark of Adobe Systems Incorporated in the United States, and/or other countries.

IT Infrastructure Library is a registered trademark of the Central Computer and Telecommunications Agency which is now part of the Office of Government Commerce.

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.

Linux is a registered trademark of Linus Torvalds 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.

ITIL is a registered trademark, and a registered community trademark of the Office of Government Commerce, and is registered in the U.S. Patent and Trademark Office

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

Cell Broadband Engine is a trademark of Sony Computer Entertainment, Inc. in the United States, other countries, or both and is used under license therefrom.

Java and all Java-based trademarks and logos are trademarks or registered trademarks of Oracle and/or its affiliates.

The United States Postal Service owns the following trademarks: CASS, CASS Certified, DPV, LACS^{Link}, ZIP, ZIP + 4, ZIP Code, Post Office, Postal Service, USPS and United States Postal Service. IBM Corporation is a non-exclusive DPV and LACS^{Link} licensee of the United States Postal Service.

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

Index

E

F

G

H

I

J

L

M

N

O

P

Q

R

S

T

U

V

W

X

Printed in USA

SC19-3694-00

Spine information:

IBM InfoSphere Federation Server **Version 10.1** **Administration Guide for Federated Systems** IBM