Find out what's new for federation in Big SQL V4.X

Mara Elisa de Paiva Fernandes Matias

September 11, 2017

Since my last tutorial on the subject appeared, some improvements have been made in terms of simplifying the setup process of the federation feature in Big SQL and adding support for new data sources or newer versions. This tutorial will take you through the incremental changes to the simplified configuration in the different versions of Big SQL V4 up to V4.2.4. It will also touch briefly on the more advanced topic of performance.

My last article on this subject was written when the federation feature was first made available, in V3 of Big SQL. Since then, the configuration has been simplified and improved. This tutorial shows the incremental changes introduced in Big SQL federation starting with V4.0 and up to V4.2.5. If you follow the examples, you should have a basic federation set up by the end of the tutorial.

This is a stand-alone tutorial, and you are not assumed to have read anything previous on the subject. To compare the current version with an older one, see the Related topics section for a link to my previous article.

Throughout the tutorial, some code samples are given. Command-line statements are in the format [<USER>@myserv]$ <COMMAND>. SQL statements have no prefix, and the output when shown was obtained using jsqsh.

What you need for this tutorial

- Knowledge of Big SQL in InfoSphere® BigInsights®, particularly about the characteristics of the Big SQL database. For more information about the product, check out the Hadoop Dev: IBM BigInsights for Hadoop Developer Community.

Overview of Big SQL federation

Federation can be described as an architecture for data warehousing in which different data sources connect and exchange information through a central federation database.

Big SQL can play the role of the central federation database when the federation feature is enabled.

Federation relies on three base objects to create a connection to a remote database: a wrapper, a server, and one (or more) user mappings, along with other important concepts, including function mappings and nicknames:
• **Wrapper** — A wrapper is an object that allows the federation server to support a specific type of data source. These objects hold information about the versions that are supported for their particular data type(s) and what the default function mappings are.

• **Server** — A server is an object that represents a connection to a specific remote data source. It holds information about the remote database — for example, whether the remote data source has a collation sequence matching Big SQL's binary collation.

• **User mapping** — A user mapping is an object that holds information about the user used as proxy when communicating with, and executing at, the remote data source. User mappings can be at a user level, where each local user is mapped to a remote user, or they can be public, where any user who authenticates to Big SQL will be mapped to a single remote user.

• **Function mappings** — Function mappings are objects that map a local function to a remote function, usually because the function either has a different return type, different arguments, or even a different name. When a wrapper is created, some default function mappings are put into place, but it is possible you to create a mapping for any function.

• **Nickname** — Nickname is an object that represents a remote table. In federation, it's possible to use three-part names to access a remote table. A nickname is more than a simple connection to the remote table. It adds an entry to the local catalog in order to collect statistics which will be used by the optimizer to calculate the cost of a query.

The figure below illustrates the architecture and how federation objects relate to real objects.

**Figure 1. Representation of federation architecture**
Incremental improvements

The changes to Big SQL federation have been gradual. Some simplifications are found only in the later versions. For those who have used federation before, here is a list of improvements and the version they were introduced in.

<table>
<thead>
<tr>
<th>Improvement</th>
<th>Big SQL version</th>
</tr>
</thead>
<tbody>
<tr>
<td>Support for data source MSSQL added.</td>
<td>V4.0</td>
</tr>
<tr>
<td>Paths to Oracle and ODBC libraries need to be added to db2dj.ini alone.</td>
<td>V4.1</td>
</tr>
<tr>
<td>Db2 data sources no longer need to be cataloged. A new syntax for create server allows for the definition of host and port.</td>
<td>V4.2</td>
</tr>
<tr>
<td>IBM-branded ODBC drivers come installed by default under /usr/ibmpacks/bigsql/&lt;version&gt;/db2/federation/odbc.</td>
<td>V4.2</td>
</tr>
<tr>
<td>Netezza ODBC drivers come installed by default under /usr/ibmpacks/bigsql/&lt;version&gt;/db2/federation/netezza.</td>
<td>V4.2</td>
</tr>
<tr>
<td>Sample db2dj.ini created by default under sqllib/cfg in instance owner's home directory.</td>
<td>V4.2</td>
</tr>
<tr>
<td>Wrappers do not need to be explicitly created.</td>
<td>V4.2</td>
</tr>
<tr>
<td>Support for ODBC connection to Oracle - Oracle client/tnsnames.ora are not required.</td>
<td>V4.2.5</td>
</tr>
<tr>
<td>New server types oracle_odbc and mssql_odbc.</td>
<td>V4.2.5</td>
</tr>
<tr>
<td>New syntax for create server for ODBC data sources means odbc.ini file is no longer required.</td>
<td>V4.2.5</td>
</tr>
</tbody>
</table>

Environment setup

Different data sources have different requirements, and newer Big SQL versions have simplified the setup steps required. You should take care to apply only the changes required, based on the data source(s) that will be used and the Big SQL version installed. The "Version changes" item on each of the following sections holds information about which Big SQL versions require each setup.

All of the examples that follow assume the instance owner is user bigsql and its home directory is /home/bigsql.

Db2 environment setup

**Version changes:** This setup is required for versions up to and including V4.1.

**Dependencies:** None

For V4.1 or earlier of Big SQL, the create server statement requires a TCP/IP node and remote database to be cataloged.

**Listing 1. Cataloging remote server SQL statement**

```
CATALOG TCPIP NODE MYNODE REMOTE 192.168.0.1 SERVER 32051
```
MYNODE – Name of the remote node on the local catalog

192.168.0.1 – IP address for remote server

32051 – Port the db2 service is running on

**Listing 2. Cataloging remote database SQL statement**

```
CATALOG DATABASE MYDB AS "RMTDB" AT NODE "MYNODE"
```

DATASD2 – Remote database instance name

RMTDB – Name of remote database on local catalog

MYNODE – Cataloged remote server

**Teradata environment setup**

*Version changes:* This setup is required for all Big SQL versions.

*Dependencies:* The Teradata wrapper depends on cliv2, which can be obtained from the official sources.

Once Teradata cliv2 is installed on your system (for the example, I assume the path is /opt/teradata/client/<VERSION>/), the wrapper libraries need to be linked to the client libraries. An executable `djxlinkTeradata` command is provided for this purpose under the sqllib/bin directory in the instance owner’s home directory.

As the root user, export the environment variable `TERADATA_LIB_DIR` to point to the cliv2 libraries, then execute the binary:

**Listing 3. Linking Teradata wrappers to Teradata cliv2 client**

```
[root@myserv]$ export TERADATA_LIB_DIR=/opt/teradata/client/15.10/lib
[root@myserv]$ /home/bigsql/sqllib/bin/djxlinkTeradata
```

**Oracle environment setup**

*Version changes:* This setup is required for Big SQL versions up to and including V4.2.

*Dependencies:* For Big SQL versions up to V4.2, the Net8 wrapper is available. This wrapper depends on Oracle’s instant client. For Big SQL versions before V4.2, you must create `db2dj.ini` under the sqllib/cfg directory in the instance owner’s home directory.

Once the Oracle instant client is installed (for the following example, I assume it is at /opt/oracle_instantclient_<VERSION>/), the client libraries must be added to the Big SQL path. This is achieved by adding variables `LIBPATH`, `DB2LIBPATH`, and `LD_LIBRARY_PATH` to the `db2dj.ini` file under the sqllib/cfg/ directory in the instance owner’s home directory.

Two other variables are required in `db2dj.ini`: `ORACLE_HOME`, with the value of the full path to the Oracle client and `TNS_ADMIN`, with the path to the folder where tnsnames.ora is located.
Netezza environment setup

Version changes: This setup is required for Big SQL versions up to and including V4.1.

Dependencies: For Big SQL up to V4.1, the Netezza data source requires generic ODBC drivers as well as the Netezza client ODBC libraries. The Netezza client can be downloaded from IBM Fix Central. IBM Fix Central provides a bundle with the Netezza client and the DataDirect ODBC drivers. Alternatively, if other ODBC data sources are going to be used, I recommend using the IBM-branded ODBC drivers instead of the DataDirect ones.

Once the ODBC drivers and the Netezza client are installed (for the following example, I assume under paths /opt/odbc_7.1/ and /opt/netezza, respectively), the library files for both the ODBC drivers and the Netezza client must be added to the Big SQL paths. This is done by adding their locations to variables LIBPATH, DB2LIBPATH, and LD_LIBRARY_PATH on file db2dj.ini under the sqllib/cfg/ directory in the instance owner's home directory.

Two other variables are required in db2dj.ini: NZ_ODBC_INI_PATH, with the value of the full path to the location of the odbc.ini file, and ODBCINST, the full path to the odbc.ini file.

Listing 4. /home/bigsql/sqllib/cfg/db2dj.ini for Oracle

LIBPATH=/opt/oracle_instantclient_12_1/lib:
DB2LIBPATH=/opt/oracle_instantclient_12_1/lib:
LD_LIBRARY_PATH=/opt/oracle_instantclient_12_1/lib:
ORACLE_HOME=/opt/oracle_instantclient_12_1
TNS_ADMIN=/home/bigsql/bin/

Listing 5. /home/bigsql/sqllib/cfg/db2dj.ini for Netezza

LIBPATH=/opt/odbc_7.1/lib:/opt/netezza/lib64:
DB2LIBPATH=/opt/odbc_7.1/lib:/opt/netezza/lib64:
LD_LIBRARY_PATH=/opt/odbc_7.1/lib:/opt/netezza/lib64:
NZ_ODBC_INI_PATH=/home/bigsql/sqllib/cfg
ODBCINST=/home/bigsql/sqllib/cfg/odbc.ini

Listing 6. /home/bigsql/sqllib/cfg/odbc.ini for Netezza

[ODBC Data Sources]
NZSQL = NetezzaSQL

[NZSQL]
Driver    = /opt/netezza/lib64/libnzodbc.so
Description= NetezzaSQL ODBC
Servername= 192.168.0.3
Port      = 5480
Database  = TESTDB
Username  = netuser
Password  = netpassword

[ODBC]
IANAAppCodePage=4
InstallDir=/opt/odbc_7.1.6
Trace=0
TraceDll=/opt/odbc_7.1.6/lib/F0trc27.so
TraceFile=odbctrace.out
UseCursorLib=0
Microsoft SQL Server environment setup

Version changes: This setup is required for Big SQL versions up to and including V4.2.

Dependencies: Access to Microsoft SQL Server is done through IBM-branded ODBC drivers.

Once the ODBC drivers are installed (for the following example, I assume they are at /opt/odbc_7.1), the library files for the ODBC drivers must be added to the Big SQL path. This can be accomplished by adding the path to variables LIBPATH, DB2LIBPATH, and LD_LIBRARY_PATH in file db2dj.ini under the sqllib/cfg/ directory in the instance owner's home directory.

One other variable is required in db2dj.ini: DJX_ODBC_LIBRARY_PATH, with the value of the full path to the lib folder of the ODBC driver.

A file or soft link named .odbc.ini must be created in the instance owner's home directory with the remote data sources definitions.

Listing 7. /home/bigsql/sqllib/cfg/db2dj.ini for Microsoft SQL Server

LIBPATH=/opt/odbc_7.1/lib:
DB2LIBPATH=/opt/odbc_7.1/lib:
LD_LIBRARY_PATH=/opt/odbc_7.1/lib:
DJX_ODBC_LIBRARY_PATH=/opt/odbc_7.1/lib:

Listing 8. /home/bigsql/.odbc.ini for Microsoft SQL Server

[mssql2012csdl3]
Driver=/opt/odbc_7.1/lib/FOsqls27.so
Description=SQL Server IBM-Branded Driver
Database=mssql2012db3
Address=192.168.0.4,1433

[mssql2016csdl1]
Driver=/opt/odbc_7.1/lib/FOsqls27.so
Description=DataDirect 7.1 SQL Server Wire Protocol
Database=mssql2016db1
Address=192.168.0.5,1433

[ODBC]
IANAAppCodePage=4
InstallDir=/opt/odbc_7.1/lib/
Trace=0
TraceDll=/opt/odbc_7.1/lib/F0trc27.so
TraceFile=odbctrace.out
UseCursorLib=0

Federation objects

There are three core objects underlying the architecture of Big SQL federation. These are a wrapper, a server, and a user mapping.

The user creating these federation objects must have DBADM privilege.

Federation objects have an inherent hierarchy. If an object is dropped, all dependent objects will be dropped as well.
Each data source type has an XML file under sqllib/cfg in the instance user's home directory. This file is named with the wrapper name, and can be consulted as a way to find out the options that are available for a data source, including wrapper options, server options, and data types supported. Db2 data sources, for example, use the DRDA wrapper so you can consult drda.xml to discover which options are available and what their default values are.

**Wrapper**

A wrapper represents a generic type of data source.

**Fenced versus trusted**

One of the most important options for wrappers, DB2_FENCED, controls whether they are run as fenced or trusted processes. Trusted wrappers are run in the same process as the Big SQL engine, which may be faster; fenced wrappers are run in separate processes, which can have advantages due to independent allocation of resources.

When a wrapper is created implicitly by the `create server` statement, it is always created as fenced. There are several advantages to running fenced wrappers. The most obvious advantage is protecting the Big SQL engine. A fenced process's resources are allocated separately from the main engine, so even if the wrapper is doing some memory-intensive processing, it will not affect the normal functioning of the database. Another advantage is better performance. Fenced wrappers, unlike trusted ones, allow parallelized joins between nicknames and local tables, where each node will receive a part of the nickname data to do a local join.

Fenced wrappers, however, do not have support for federated procedures. Federated procedures are to remote procedures what nicknames are to remote tables, and are only supported by trusted wrappers.

**Default wrappers**

Since V4.2 of Big SQL, when a `create server` statement is run, a default fenced wrapper is created for that server type. Following is a list of mappings between server types and the default wrapper created.

**Table 2. Default wrappers by server type**

<table>
<thead>
<tr>
<th>Server type</th>
<th>V4.2.5 wrappers</th>
<th>V4.2 wrappers</th>
</tr>
</thead>
<tbody>
<tr>
<td>DB2</td>
<td>DRDA</td>
<td>DRDA</td>
</tr>
<tr>
<td>Teradata</td>
<td>TERADATA</td>
<td>TERADATA</td>
</tr>
<tr>
<td>Oracle</td>
<td>ODBC</td>
<td>Net8</td>
</tr>
<tr>
<td>Netezza</td>
<td>ODBC</td>
<td>ODBC</td>
</tr>
<tr>
<td>Microsoft SQL Server</td>
<td>ODBC</td>
<td>MSSQL:ODBC3</td>
</tr>
</tbody>
</table>

**How to create or drop a wrapper**

Since V4.2, a default wrapper is implicitly created by the `create server` statement. It is always possible to create a personalized wrapper.
You can choose to use the data type's default wrapper name from Table 2, or a different name. If a different name is used, the LIBRARY clause must be defined with the file name of the corresponding wrapper library. Using the default wrapper name makes that unnecessary. Wrapper libraries are located under sqllib/lib64 in the instance owner’s home directory.

The ODBC wrapper has an extra required option: MODULE. In the following examples, the ODBC library path used is the one that is installed by default since Big SQL V4.2. For older versions, the MODULE option should point to where the ODBC library was installed.

Wrappers are at the top of the hierarchy, so if a wrapper is dropped, all servers, user mappings, and nicknames that use it will be dropped as well.

Wrapper options can be consulted in the `<wrapper_options>` section of the XML file under sqllib/cfg for the respective wrapper.

Following are a series of examples for each of the data source types supported. The examples use the following format:

1. Simplest `create wrapper` statement
2. `drop wrapper` statement
3. `create wrapper` with options
4. `create wrapper` with a different name

**Listing 9. DRDA wrapper**

```
CREATE WRAPPER DRDA;
DROP WRAPPER DRDA;
CREATE WRAPPER DRDA OPTIONS (DB2_FENCED 'Y');
CREATE WRAPPER MYDRDA LIBRARY 'libdb2drda.so';
```

DRDA – Default name of Db2 wrappers

MYDRDA – Custom name for a wrapper, using a custom name requires the definition of clause LIBRARY

libdb2drda.so – Library for Db2 wrapper

DB2_FENCED – Option to set wrapper as a fenced process

**Listing 10. Teradata wrapper**

```
CREATE WRAPPER TERADATA;
DROP WRAPPER TERADATA;
CREATE WRAPPER TERADATA OPTIONS (DB2_FENCED 'Y');
CREATE WRAPPER MYTERADATA LIBRARY 'libdb2teradata.so';
```

TERADATA – Default Teradata wrapper name

libdb2teradata.so – Library for Teradata wrapper
Listing 11. ODBC wrapper

CREATE WRAPPER ODBC OPTIONS (MODULE '/usr/ibmpacks/bigsql/4.2.5.0/db2/federation/odbc/lib/libodbc.so');
DROP WRAPPER MYODBC;
CREATE WRAPPER MYODBC LIBRARY 'libdb2rcodbc.so' OPTIONS (MODULE '/usr/ibmpacks/bigsql/4.2.5.0/db2/federation/odbc/lib/libodbc.so');

ODBC – Default ODBC wrapper name

libdb2rcodbc.so – Library for ODBC wrappers

MODULE – Option is required for ODBC wrappers and should be the absolute path to ODBC library file

How to alter a wrapper

Once a wrapper is created, you can alter it by using the ALTER WRAPPER statement.

Listing 12. ALTER WRAPPER

ALTER WRAPPER DRDA OPTIONS (SET DB2_FENCED 'N');
ALTER WRAPPER ODBC OPTIONS(SET DB2_FENCED 'Y', SET MODULE '/opt/odbc7.1/lib/libodbc.so');

Catalogs for wrappers and wrapper options

There are two views with information about wrappers under the SYSCAT schema: SYSCAT.WRAPPERS and SYSCAT.WRAPPOPTIONS can be queried to consult existing wrappers and their options, respectively.

Listing 13. Wrapper catalog queries

select * from syscat.wrappers;
+----------+----------+-------------+-----------------+---------+
| WRAPNAME | WRAPTYPE | WRAPVERSION | LIBRARY         | REMARKS |
+----------+----------+-------------+-----------------+---------+
| DRDA     | R        | 0           | libdb2drda.so   | [NULL]  |
| ODBC     | R        | 0           | libdb2rcodbc.so | [NULL]  |
+----------+----------+-------------+-----------------+---------+
2 row in results(first row: 0.928s; total: 0.930s)

select * from syscat.wrapoptions where wrapname='ODBC';
+----------+------------+-----------------------------------------------------------------+
| WRAPNAME | OPTION     | SETTING                                                         |
+----------+------------+-----------------------------------------------------------------+
| ODBC     | DB2_FENCED | Y                                                               |
| ODBC     | MODULE     | /usr/ibmpacks/bigsql/4.2.5.0/db2/federation/odbc/lib/libodbc.so |
+----------+------------+-----------------------------------------------------------------+
2 rows in results(first row: 0.086s; total: 0.090s)

Server

Federation server objects represent remote database instances.

Data source to server type/wrapper mapping

The following table shows the relation between the data source, the server type to be used in the create server type, and the wrapper used.
Table 3. Data source mapping to server type for `create server` statement

<table>
<thead>
<tr>
<th>Data source</th>
<th>Server type</th>
<th>Wrapper</th>
<th>Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>Db2 LUW</td>
<td>db2/udb</td>
<td>drda</td>
<td></td>
</tr>
<tr>
<td>Db2 z/OS®</td>
<td>db2/zos</td>
<td>drda</td>
<td></td>
</tr>
<tr>
<td>Oracle</td>
<td>oracle</td>
<td>Net8</td>
<td></td>
</tr>
<tr>
<td>Oracle</td>
<td>oracle_odbc</td>
<td>odbc</td>
<td>Only available in V4.2.5</td>
</tr>
<tr>
<td>Microsoft SQL Server</td>
<td>mssqlserver</td>
<td>mssqlodbc3</td>
<td></td>
</tr>
<tr>
<td>Microsoft SQL Server</td>
<td>mssql_odbc</td>
<td>odbc</td>
<td>Only available in V4.2.5</td>
</tr>
<tr>
<td>Netezza</td>
<td>netezza</td>
<td>odbc</td>
<td></td>
</tr>
<tr>
<td>Teradata</td>
<td>teradata</td>
<td>teradata</td>
<td></td>
</tr>
</tbody>
</table>

How to create or drop servers

Servers are the second highest object in the hierarchy of federation objects. When a server is dropped, all user mappings and nicknames associated with it are dropped as well. Server options can be found in the `<server_options>` subsection for the server type used, from the wrapper XML file under sqllib/cfg in the instance owner's home directory. The `create server` statement is one of the statements that has been simplified in newer versions of Big SQL. The syntax changed slightly, and some sections such as WRAPPER became optional. Following are examples for each data source with the Big SQL version they are valid on:

- Create server statement
- Drop server statement

Listing 14. Db2 data sources (before V4.2)

```sql
CREATE SERVER MYDB2 TYPE DB2/UDB VERSION 11 WRAPPER DRDA AUTHORIZATION "newton" PASSWORD "Password" OPTIONS (NODE 'MYNODE', DBNAME 'RMTDB', PUSHDOWN 'Y');
DROP SERVER MYDB2;
```

- **Password** – Password is quoted to preserve case
- **NODE** – Node cataloged during setup
- **DBNAME** – Remote database cataloged during setup

Listing 15. Db2 data sources (from V4.2)

```sql
CREATE SERVER MYDB2 TYPE DB2/UDB VERSION 11 AUTHORIZATION "newton" PASSWORD "Password" OPTIONS (HOST '192.168.0.1', PORT '5000', DBNAME 'MYDB', PUSHDOWN 'N');
DROP SERVER MYDB2;
```

- **HOST** – Can use the IP, fully qualified DN, or recognized alias from local hosts file
- **PORT** – Port the remote Db2 server is listening on
- **DBNAME** – Remote database instance name
PUSHDOWN – One of the possible options for this data source. It blocks all functions or orderings from being executed at the remote server.

### Listing 16. Teradata data sources

```sql
CREATE SERVER TERA TYPE TERADATA VERSION 15 (NODE '192.168.0.6')
DROP SERVER TERA
```

**NODE** – Can be IP, fully qualified DN, or a recognized alias from local hosts file

### Listing 17. Oracle data sources (before V4.2.5)

```sql
CREATE SERVER ORA TYPE ORACLE WRAPPER Net8 VERSION 11 OPTIONS (NODE 'oracle11r2');
DROP SERVER ORA;
```

**NODE** – entry in tnsname.ora

### Listing 18. Oracle data sources (4.2.5)

```sql
CREATE SERVER ORA TYPE ORACLE_ODBC OPTIONS (HOST '192.168.0.2', PORT '1521', SERVICE_NAME 'orcl.ibm.com');
DROP SERVER ORA;
```

**HOST** – IP, fully qualified DN, or recognized alias from hosts files

**PORT** – Port the remote server is running on

**SERVICE_NAME** – Service name for database instance

### Listing 19. Netezza data sources (before V4.2.5)

```sql
CREATE SERVER NTZ TYPE NETEZZA WRAPPER ODBC VERSION 7 OPTIONS (NODE 'netezza');
DROP SERVER NTZ;
```

**VERSION** – The ODBC driver version not Netezza's

**NODE** – entry in odbc.ini

### Listing 20. Netezza data sources (V4.2.5)

```sql
CREATE SERVER NTZ TYPE NETEZZA OPTIONS (HOST '192.168.0.3', DBNAME 'TESTDB');
DROP SERVER NTZ;
```

**HOST** – IP, fully qualified DN, or recognized alias from hosts files

**DBNAME** – Remote database name

### Listing 21. Microsoft SQL server data sources (before V4.2.5)

```sql
CREATE SERVER MSSQL TYPE MSSQLSERVER WRAPPER MSSQLODBC3 VERSION 2012 OPTIONS (NODE 'mssql2016csdl1');
DROP SERVER MSSQL;
```

**NODE** – entry in odbc.ini
Listing 22. Microsoft SQL server data sources (V4.2.5)

```sql
CREATE SERVER MSSQL TYPE MSSQL_ODBC VERSION 2016 OPTIONS (HOST '192.168.0.5', DBNAME 'mssql2016db1');
DROP SERVER MSSQL;
```

**HOST** – IP, fully qualified DN, or recognized alias from hosts files

**DBNAME** – Remote database instance name

**How to alter a server**

Once a server is created, its options can be altered by using the `ALTER SERVER` statement.

The example shows:

1. Drop a server option
2. Update the value of an existing server options
3. Add a server option

Listing 23. Alter server

```sql
ALTER SERVER MYDB2 OPTIONS (DROP PUSHDOWN);
ALTER SERVER MYDB2 OPTIONS (SET COLLATING_SEQUENCE 'Y')
ALTER SERVER MSSQL OPTIONS (ADD DB2_MAXIMAL_PUSHDOWN 'Y')
```

**DB2_MAXIMAL_PUSHDOWN** – Server option that causes the optimizer to calculate query cost based solely on network cost. This will cause most queries to be sent to the remote server as long as they are accepted.

**Catalog for servers and server options**

There are two views that provide information on servers in the SYSCAT schema:

`SYSCAT.SERVERS` and `SYSCAT.SERVEROPTIONS` will allow the querying of existing servers and their options, respectively.

Listing 24. Server catalog queries

```sql
select * from syscat.servers;
```

```
+----------+------------+------------+---------------+---------+
| WRAPNAME | SERVERNAME | SERVERTYPE | SERVERVERSION | REMARKS |
+----------+------------+------------+---------------+---------+
| DRDA     | BIGSQL_1   | BIGSQL     | 4.2           | [NULL]  |
| DRDA     | ZOS_1      | Db2/ZOS    | 11.0          | [NULL]  |
| DRDA     | UDB_1      | Db2/UDB    | 11.1          | [NULL]  |
| ODBC     | MSSQL      | MSSQL_ODBC | 2016          | [NULL]  |
+----------+------------+------------+---------------+---------+
4 rows in results (first row: 0.028s; total: 0.030s)
```

```sql
select OPTION, SETTING from syscat.serveroptions where servername='MSSQL';
```

```
+------------------------+---------------+
| OPTION                 | SETTING       |
+------------------------+---------------+
| DBNAME                 | mssql2016db1  |
| HOST                   | 9.181.139.172 |
| DB2_MAXIMAL_PUSHDOWN   | Y             |
+------------------------+---------------+
3 row in results(first row: 0.520s; total: 0.522s)
```
User mapping

User mappings are mappings between local users and remote users.

User mappings or public mappings?

User mapping can be created between a single local user and a remote user. It is also possible to create a public mapping, where every user with access to Big SQL is mapped to the same remote user.

Public mappings and single user mappings cannot coexist.

The main consideration that might help you decide which of these methods to use is: Who needs what access to the remote system?

If the required access to the remote data is uniform, and restricted for all users with access to Big SQL, then a public mapping to a user with, for example, only select authorization on a restricted list of tables might be the best and simplest choice.

If different users will require different levels of access, or access to different remote schemas, then user mapping is the simplest choice.

How to create/drop user mappings

There are two special keywords for the FOR clause of the create user mappings statement. PUBLIC creates a public mapping to the remote user described in the OPTIONS clause of the statement. USER creates a mapping for the user currently attached or connected to Big SQL.

For Db2 data sources, an authorization is required on the create server statement, and a user mapping is also required. This is not required for other data sources.

User mappings are at the bottom of the hierarchy; dropping them will not affect any other objects. Removing the mapping may remove or change a user’s access to the other objects, but it won’t change the objects themselves.

Once user mappings have been created, a basic setup is complete. The setup can now be tested by connecting to a remote data source via a nickname, a three-part name, or by using a passthru session.

The following example shows:

1. Create a public user mapping.
2. Drop public mapping.
3. Create a user mapping for the connected user.
4. Drop connected user mapping.
5. Create a user mapping for a named user.
6. Drop named user mapping.
Listing 25. Create user mapping

CREATE USER MAPPING FOR PUBLIC SERVER UDB_1 OPTIONS (REMOTE_AUTHID 'newton', REMOTE_PASSWORD 'Password');
DROP USER MAPPING FOR PUBLIC SERVER UDB_1;
CREATE MAPPING FOR USER SERVER MSSQL OPTIONS (REMOTE_AUTHID 'user1', REMOTE_PASSWORD 'VeryHardPassword');
DROP USER MAPPING FOR USER SERVER MSSQL;
CREATE MAPPING FOR dataadmin SERVER UDB_1 OPTIONS (REMOTE_AUTHID 'newton', REMOTE_PASSWORD 'Password');
DROP USER MAPPING FOR dataadmin SERVER UDB_1;

REMOTE_AUTHID/REMOTE_PASSWORD – These are string literals and so must be enclosed in single quotes.

How to alter user mappings

If a remote user changes (for example, because its password expired), it is possible to alter the user mapping by running the ALTER USER MAPPING statement.

Listing 26. Alter user mapping

ALTER USER MAPPING FOR dataadmin SERVER UDB_1 OPTIONS (SET REMOTE PASSWORD 'newPassword');

Catalog to query existing user mappings

There is one view of SYSCAT.USEROPTIONS where information on existing user mappings can be consulted.

Listing 27. User mappings catalog query

list * from syascat.options;

+---------------+--------------+-----------------+-----------------+----------+
| AUTHID | AUTHIDTYPE | SERVERNAME | OPTION          | SETTING  |
+---------------+--------------+-----------------+-----------------+----------+
| PUBLIC | G           | UDB_1         | REMOTE_AUTHID   | newton   |
| PUBLIC | G           | UDB_1         | REMOTE_PASSWORD | ********  |
| BIGSQL | U           | BIGSQL_1      | REMOTE_AUTHID   | bigsql   |
| BIGSQL | U           | BIGSQL_1      | REMOTE_PASSWORD | ********  |
+---------------+--------------+-----------------+-----------------+----------+
4 rows in results(first row: 0.097s; total: 0.100s)

Nickname

Nicknames represent remote tables or views.

Difference between nicknames and three-part names

Three-part names are a way to access remote table or view names. A three-part name comprises three dot-separated tokens: the server name, the remote schema, and the remote table name.

The advantage of using three-part names is they do not exist locally and so do not take up space in the local directory. They do require a server object to be in place and a user mapping to exist just as nicknames do.

Nicknames, on the other hand, are entries on the local directory. Their main advantage is that they can have statistics calculated for them, even for data sources that do not have statistics. These
statistics are used in query optimization to decide which, if any, part of an SQL statement should be sent to the remote data source. They also enable inter-partition parallelism on joins between nicknames and local tables where a nickname's data is partitioned and the join is processed in parallel by all the nodes.

Nicknames cannot be created over tables with distinct, reference, or user-defined column types.

How to create or drop nicknames

A nickname can be created over any table or view that the remote user can access, unless the table or view includes columns of the types mentioned in the previous section, that is: distinct, reference, or user-defined column types.

Nicknames are at the bottom of the federation hierarchy; dropping them will affect none of the other objects. Dropping a nickname does not affect the remote table.

Listing 28. Create nickname

```sql
CREATE NICKNAME "UDBPEOPLE" FOR UDB_1.NEWTON.PEOPLE;
DROP NICKNAME "UDBPEOPLE";
```

**UDBPEOPLE** – Local name for nickname  
**UDB_1** – Server name  
**NEWTON** – Remote schema  
**PEOPLE** – Remote table name

How to alter nicknames

Once a nickname is created, the remote column types are mapped to local types using default mappings. A user might want to alter the nickname to represent the remote data more accurately. For example, Oracle supports tables with **INT** columns but the underlying table structure will contain **NUMBER**, and **NUMBER** will always map locally to **DECIMAL**.

Mapping remote data accurately to the correct local data type will help with query optimization and will improve performance.

Note that an incorrect mapping may result in poor performance or incorrect results.

Listing 29. Alter nickname

```sql
ALTER NICKNAME "ORAPEOPLE" ALTER COLUMN ID LOCAL TYPE INT;
```

Catalog for nicknames

Nicknames do not have a specific catalog. They can be queried in the SYSCAT.TABLES view where they have the **TYPE 'N'**.
Type mappings
In the nickname section above, I introduced the topic of data type mapping. As well as changing the data type mapping at a nickname level, it is possible to create generic data type mapping that will apply to all nicknames created for a given server.

How to create or drop data type mappings
Generic data type mapping can be created at three different degrees of granularity: server type, server type and version, or server object. In other words, it can be created for all servers of a specific type, for all servers of a specific type and a specific version, or for a specific server (by name).

Let's suppose our system has two servers. Both have type Db2/UDB, but one is called UDB_1 (and is V10.5), while the other is called UDB_2 (and runs V11).

This type of data type mapping affects nicknames only if they are created after the mapping is created or dropped.

The following examples use this format:

1. Create type mapping
2. Drop type mapping

Listing 31. Server type data type mapping
```
CREATE DATA TYPE MAPPING MY_INT_MAP FROM BIGINT TO SERVER TYPE DB2/UDB TYPE INT;
DROP DATA TYPE MAPPING MY_INT_MAP;
```

In the example above, I created a server type level data type mapping. This would apply to both our servers and would cause any nicknames created after this type mapping to automatically map any remote INT columns into local BIGINT columns.

Listing 32. Server type and version data type mapping
```
CREATE DATA TYPE MAPPING MY_INT_MAP2 FROM BIGINT TO SERVER DB2/UDB VERSION 10.5 TYPE INT;
DROP DATA TYPE MAPPING MY_INT_MAP2;
```

In the example above, a server type and version mapping is created and dropped. This mapping would apply only to UDB_1 because of the version specification. Nicknames created under UDB_1 while this mapping was active would see their remote INT columns mapped locally to BIGINT, while nicknames under UDB_2 would still have the default mapping of remote INT columns into INT.
Listing 33. Server object data type mapping
CREATE DATA TYPE MAPPING MY_INT_MAP3 FROM BIGINT TO SERVER UDB_1 TYPE INT;
DROP DATA TYPE MAPPING MY_INT;

This last example shows a mapping created for a specific server object. This mapping would be applied only to nicknames under the specified server object, in this case, UDB_1.

Function mappings
It is also possible to create mappings between local and remote functions as a way to improve performance.

How to create or disable function mappings
There are a great number of functions that are mapped by default. It is not possible to create a function mapping on a function that is already mapped by default, but it is possible to disable default mappings.

A function might not be mapped by default if there is a difference between the return type of the local and remote functions. If this difference is not relevant, then you can create a function mapping that will lead to performance gains.

Function mappings can be created, for system or user functions, at three different degrees of granularity: server type, server type and version, or server object. That is, for all servers of a specific type; for all servers of a specific type and a specific version; or for a specific server (by name).

The following examples use the format:

1. Create function mapping
2. Drop function mapping

Listing 34. Server type function mapping
CREATE FUNCTION MAPPING MYSTDDEV FOR SYSIBM.STDDEV (INT) SERVER TYPE DB2/UDB OPTIONS (REMOTE_NAME 'SYSIBM.STDDEV');
DROP FUNCTION MAPPING MYSTDDEV;

Listing 35. Server type and version function mapping
CREATE FUNCTION MAPPING MYSTDDEV FOR SYSIBM.STDDEV (INT) SERVER TYPE DB2/UDB VERSION 10.5 OPTIONS (REMOTE_NAME 'SYSIBM.STDDEV');
DROP FUNCTION MAPPING MYSTDDEV;

Listing 36. Server object function mapping
CREATE FUNCTION MAPPING MYSTDDEV2 FOR SYSIBM.STDDEV (SMALLINT) SERVER UDB_1 OPTIONS (REMOTE_NAME 'SYSIBM.STDDEV');
DROP FUNCTION MAPPING MYSTDDEV2;

Passthru sessions
Passthru is a mode in which statements are passed directly to the remote data source. Big SQL becomes a client of the remote data source, connected using the user defined in user mapping.
Passthru requires a wrapper, server, and valid user mapping to have been created.

This mode is particularly useful when testing if the remote server and user mapping have been correctly configured. It can also be used to access the remote data source directly if required.

While in passthru mode, the user does not have access to local tables.

Passthru mode is entered by running the SQL statement `SET PASSTHRU` with the server object name you want to access, and is terminated by running `SET PASSTHRU RESET`.

**Listing 37. Passthru example**

```
SET PASSTHRU ORASERV;
0 rows affected (total: 0.011s)
select * from product_component_version where product like 'Oracle Database%';
| PRODUCT                                | VERSION    | STATUS           |
|----------------------------------------|------------+------------------|
| Oracle Database 12c Enterprise Edition | 12.1.0.1.0 | 64bit Production |
+----------------------------------------+------------+------------------+
1 row in results(first row: 3.828s; total: 4.011s)
SET PASSTHRU RESET;
0 rows affected (total: 0.002s)
```

**Performance**

Query optimization, and predicate pushdown, depend on having correct statistics for all objects involved in the query. Both nicknames and local Hadoop tables should have up-to-date statistics when you do joins between local and remote tables for example.

**Automatic statistics**

By default, Big SQL has automatic collection of statistics. There are differences between automatic collection of statistics for Hadoop tables and nicknames. Nicknames behave more closely to cache tables in terms of automatic statistics collection. By default, Hadoop tables have their statistics collected by auto-analyze which (by default) runs every 10 minutes; nicknames, like cache tables, are evaluated by `auto_runstats` which runs approximately every two hours.

Analyze or runstats have the same common procedure, when they run they use criteria (has the table been in use, has it had statistics run before, and so on) to decide if statistics should be recalculated.

The database configuration can be consulted to check if `auto_runstats` is enabled on a Big SQL instance.

**Listing 38. Runstats on db cfg**

```
[bigsql@myserv ~]$ db2 get db cfg
  ...
  Automatic runstats (AUTO_RUNSTATS) = ON
  Real-time statistics (AUTO_STMT_STATS) = ON
  Statistical views (AUTO_STATS_VIEWS) = OFF
  Automatic sampling (AUTO_SAMPLING) = ON
  ...
```
One way to check if a table has had statistics run is to query the TABLES and COLUMNS views under the SYSSTAT schema. Tables that have not had statistics run will have a cardinality (card) column of -1. In the following example, neither the nickname nor the Hadoop table have statistics.

**Listing 39. SYSSTAT.TABLES view**

```sql
select tabschema, tabname, card from sysstat.tables where tabschema='BIGSQL';
```

+-----------+-----------+------+
| TABSCHEMA | TABNAME   | CARD |
+-----------+-----------+------+
| BIGSQL    | HADOOP_T1 |   -1 |
| BIGSQL    | NICK1     |   -1 |
| BIGSQL    | TEST      |    0 |
+-----------+-----------+------+
3 rows in results(first row: 0.056s; total: 0.059s)

**Manually collecting statistics**

If a nickname is going to be in use soon after creation, and performance from the start is important, nicknames statistics can be updated by using the SYSPROC.NNSTAT procedure.

In the examples provided, collection method 0 was used. This collection method will try to retrieve statistics from the remote source. If it cannot, it will do a series of queries to fill the statistics views. This might not always be the best option in cases where the remote statistics are not up-to-date.

**Listing 40. Updating a nickname’s statistics**

```sql
CALL SYSPROC.NNSTAT('UDB_1','BIGSQL','NICK1','ID, COL_VCH, DEC1','IX1',0,'/home/bigsql/stats.log', ?, 1)
```

*a1* – Server object.

*BIGSQL* – Schema where nickname is.

*NICK1* – Nickname to run statistics for. If this parameter and schema parameter are NULL, statistics are run for every nickname created for server UDB_1.

'**ID..**' – Columns to collect statistics for.

'**IX1**' – Index to collect statistics for.

0 – Collection method, can be 1 (grab statistics from remote data source), 2 (query data and calculate statistics locally), or 0/NULL (try method 1, if that fails try method 2).

'*/home/bigsql/stats.log*' – Log file for operation.

? – Output parameter

1 – Update method, which is relevant when statistics are being run for more than one nickname. Can be 0 (all statistics will be updated at the same time), or 1 (statistics are updated one nickname at a time).
**Listing 41. Updating statistics for all nicknames created under a schema**

```
CALL SYSPROC.NNSTAT(NULL,'BIGSQL',NULL,NULL,NULL,0,NULL,?,1);
ok. (total: 14.230s)
```

<table>
<thead>
<tr>
<th>Return Code</th>
<th>Param #2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Successful</td>
</tr>
</tbody>
</table>

**How to get a plan for a query and read it**

To get a query plan, you must install explain tables on the schema for the user requesting the plan.

One method of installing them is by using the EXPLAIN.DDL file located under sqllib/misc on the instance owner's home directory.

**Listing 42. Installing the explain tables**

```
[bigsq1@myserv ~]$ db2 -tvf ~/sqllib/misc/EXPLAIN.DDL
```

One way to explain a query is by prefixing it with "explain all for" (which populates the explain tables), and then using db2exfmt to output this information into a file in a format that is easy to read.

**Listing 43. Explain all and db2exfmt**

```
explain all for select count (*) from nick1;
0 rows affected (total: 0.094s)
```

```
[bigsq1@myserv ~]$ db2exfmt -d bigsql -1 -o count.plan
```

**Pushdown opportunities**

In this section, I look at predicate pushdown opportunities, specifically function ABS, which returns the absolute value of the argument passed. For this example, I will be using a Db2 data source which has the same exact function (same name and same return types), so a default mapping exists for this function.
Listing 44. Request plan for ABS

```sql
explain all for select abs(id) from nick1 where abs(id)>6;
0 rows affected (total: 0.094s)
```

```bash
[bigs@cerv ]$ db2exfmt -d bigsql -1 -o abs.plan
DB2 Universal Database Version 11.1, 5622-044 (c) Copyright IBM Corp. 1991, 2016
Licensed Material - Program Property of IBM
IBM DATABASE 2 Explain Table Format Tool

Connecting to the Database.
Connect to Database Successful.
Binding package - Bind was Successful
Output is in abs.plan.
Executing Connect Reset -- Connect Reset was Successful.
```

In the `abs.plan` file, locate the `SHIP` section. This section corresponds to the operation sent to the remote data source. Within this section, the `RMTQTXT` argument is the query sent to the remote server. In principle, for the query, the whole query can be pushed down.

### Listing 45. ABS plan SHIP section

```plaintext
2) SHIP  : (Ship)
    
    Arguments:
    ---------
    CSERQY  : (Remote common subexpression)
            FALSE
    DSTSEVER: (Destination (ship to) server)
            - (NULL).
    RMTQTXT : (Remote statement)
            SELECT A0."ID" C0 FROM "NEWTON"."TABLE1" A0 WHERE (6 < SYSIBM.ABS(A0."ID")) FOR READ ONLY
    SRCSEVER: (Source (ship from) server)
            UDB_1

In the example, only the filter part was pushed down; the returned values are the raw INT values rather than the ABS that was expected. Since the predicate was pushed down as a filter, we know there is an existing mapping and the predicate can be pushed down.

The optimizer uses complex heuristics to calculate the cost of an operation. In cases like this, it is likely that calculating ABS remotely would have worse performance than doing it locally. You can check this by forcing the optimizer to push down everything it can, taking into consideration only network cost. This can be done by adding or setting the server object option `DB2_MAXIMAL_PUSHDOWN` to 'Y'.

Listing 46. ABS plan with DB2_MAXIMAL_PUSHDOWN

alter server udb_1 options (add DB2_MAXIMAL_PUSHDOWN 'Y');
0 rows affected (total: 0.181s)
explain all for select abs(id) from nick1 where abs(id)>6;
0 rows affected (total: 0.035s)

[bigsq1@myserv ~]$ db2exfmt -d bigsql -l -o abs2.plan
DB2 Universal Database Version 11.1, 5622-044 (C) Copyright IBM Corp. 1991, 2016
Licensed Material - Program Property of IBM
IBM DATABASE 2 Explain Table Format Tool

Connecting to the Database.
Connect to Database Successful.
Output is in abs2.plan.
Executing Connect Reset -- Connect Reset was Successful.

Listing 47. ABS with DB2_MAXIMAL_PUSHDOWN SHIP section

2) SHIP  : (Ship)
  ...
  Arguments:
  -------
  CSERQY  : (Remote common subexpression)
  FALSE
  DSTSEVER: (Destination (ship to) server)
  - (NULL).
  RMTQTX : (Remote statement)
  SELECT SYSIBM.ABS(A0."ID") C0 FROM "NEWTON"."TABLE1" A0 WHERE (6 <
  SYIBM.ABS(A0."ID") FOR READ ONLY
  SRCSEVER: (Source (ship from) server)
  UDB_1

This time the full query was pushed down. While analyzing the pushdown opportunities for a
function, if the function is not pushed down, even after DB2_MAXIMAL_PUSHDOWN is set to 'Y', it is
likely that no default mapping exists for that function. In those cases, you would have the option to
evaluate if an equivalent function exists and create a function mapping.

Inter-partition parallelism

When doing joins between local Hadoop and remote tables, it is possible to use inter-partition
parallelism. What this means is this: Although only the head node ever communicates with the
remote data source if both the local and remote table have statistics run, when performing a join,
remote data will be distributed so that each node will receive a portion of the total data and do a
local join.

Joining a Hadoop table to a nickname

The only requirement for inter-partition parallelism when joining a local Hadoop table and a
nickname is that the wrapper used for the nickname is a fenced wrapper.

A parallel approach will only happen if the amount of nickname data in the join is smaller than the
amount of local partitioned data; otherwise the cost of partitioning and sending data to the nodes
would make the operation slower rather than faster.
If statistics are missing or incorrect, it is less likely that the optimizer will calculate a parallelized query to be the most cost effective.

The query plan for an inter-parallel query will have a BTQ operator above the SHIP operator in the Access Plan section.

**Listing 48. No parallelism on join**

Access Plan:
```
---------------
| Total Cost: 1510.36 |
| Query Degree: 8 |
```

<table>
<thead>
<tr>
<th>Rows</th>
<th>RETURN</th>
</tr>
</thead>
<tbody>
<tr>
<td>( 1)</td>
<td>Cost I/O</td>
</tr>
<tr>
<td>694676</td>
<td>HSJOIN</td>
</tr>
<tr>
<td>( 2)</td>
<td>1510.36</td>
</tr>
<tr>
<td>13</td>
<td>BTQ SHIP</td>
</tr>
<tr>
<td>25451.6</td>
<td>13365.6</td>
</tr>
<tr>
<td>1311.88</td>
<td>193.226</td>
</tr>
<tr>
<td>10</td>
<td>3</td>
</tr>
<tr>
<td>64787.7</td>
<td>355360</td>
</tr>
<tr>
<td>LTQ</td>
<td>NICKNM: BIGSQL</td>
</tr>
<tr>
<td>( 4)</td>
<td>ALLTYPES_NETZ</td>
</tr>
<tr>
<td>1299.55</td>
<td>Q2</td>
</tr>
<tr>
<td>10</td>
<td></td>
</tr>
<tr>
<td>64787.7</td>
<td>TBSCAN</td>
</tr>
<tr>
<td>( 5)</td>
<td>1292.19</td>
</tr>
<tr>
<td>10</td>
<td></td>
</tr>
<tr>
<td>3.42692e+06</td>
<td>HTABLE: BIGSQL</td>
</tr>
<tr>
<td>BIGAFED_ALLTYPES_SDS</td>
<td>BIGAFED_ALLTYPES_SDS</td>
</tr>
<tr>
<td>Q1</td>
<td></td>
</tr>
</tbody>
</table>
```

**Listing 49. Parallelism on join**

Access Plan:
```
---------------
| Total Cost: 672368 |
| Query Degree: 8 |
```

<table>
<thead>
<tr>
<th>Rows</th>
<th>RETURN</th>
</tr>
</thead>
<tbody>
<tr>
<td>( 1)</td>
<td>Cost I/O</td>
</tr>
<tr>
<td>694676</td>
<td>HSJOIN</td>
</tr>
<tr>
<td>( 2)</td>
<td>1510.36</td>
</tr>
<tr>
<td>13</td>
<td>BTQ SHIP</td>
</tr>
<tr>
<td>25451.6</td>
<td>13365.6</td>
</tr>
<tr>
<td>1311.88</td>
<td>193.226</td>
</tr>
<tr>
<td>10</td>
<td>3</td>
</tr>
<tr>
<td>64787.7</td>
<td>355360</td>
</tr>
<tr>
<td>LTQ</td>
<td>NICKNM: BIGSQL</td>
</tr>
<tr>
<td>( 4)</td>
<td>ALLTYPES_NETZ</td>
</tr>
<tr>
<td>1299.55</td>
<td>Q2</td>
</tr>
<tr>
<td>10</td>
<td></td>
</tr>
<tr>
<td>64787.7</td>
<td>TBSCAN</td>
</tr>
<tr>
<td>( 5)</td>
<td>1292.19</td>
</tr>
<tr>
<td>10</td>
<td></td>
</tr>
<tr>
<td>3.42692e+06</td>
<td>HTABLE: BIGSQL</td>
</tr>
<tr>
<td>BIGAFED_ALLTYPES_SDS</td>
<td>BIGAFED_ALLTYPES_SDS</td>
</tr>
<tr>
<td>Q1</td>
<td></td>
</tr>
</tbody>
</table>
Conclusion

If you have followed along with this tutorial, you should now know how to do a basic setup of the federation. You will know how to set up your environment, create the basic federation objects, and even do some basic investigation of performance of queries involving remote tables.

BigSQL's federation is an enriching feature for a big data product. It allows a user to leverage data located in other silos before consolidating it. It can even be used as a way to decide if consolidation of these data sources is useful. I have used the federation feature for some time now and am very exited about its possibilities. I hope this tutorial will help any who would like to try this feature and that you enjoy using it as much as I have.
Related topics

- "Set up and use federation in InfoSphere BigInsights Big SQL V4.0" (developerWorks, January 2013) explains the use of federation in V4 of Big SQL.
- InfoSphere BigInsights describes Big SQL. Pay particular attention to the characteristics of the BIG SQL database.
- Hadoop Dev: IBM BigInsights for Hadoop Developer Community contains many posts from BigInsights users.