Using data federation technology in IBM WebSphere Information Integrator: Data federation design and configuration

Part 1 in a series introducing data federation

Anjali Betawadkar-Norwood
Eileen Lin
Ioana Ursu

Introduction

Perhaps the most significant challenges in corporate information management today are the multiplicity, heterogeneity, and geographic distribution of data sources. There can be any number of factors that lead to this data complexity within a company -- for instance, mergers and acquisitions that combine different organizations with varying IT infrastructures, business growth from a local to a transnational company, or simply piecemeal technologies used to build information systems over a period of time. No matter how difficult the problem, synchronizing and managing the diverse data sources is a key requirement to keep your business competitive.

By seamlessly tying together multiple data sources into a federated system, IBM WebSphere Information Integrator provides the ability to achieve:

- **Transparency**: the capability to code and use applications as though the data resides in a single database
- **Heterogeneity**: the ability to accommodate different data requirements and sources in the enterprise
- **Autonomy**: the absence of restrictions being enforced at the remote data source, allowing it to remain autonomous
• **High function**: the ability of applications to exploit not only the high degree of function provided by the federated system (see [Resources](#) for more information), but also the special functions unique to some of the data sources

• **Extensibility and openness**: the flexibility to seamlessly add a new data source to the enterprise information system

• **Optimized performance**: the power of applications developed for the federated system to achieve strong performance without the need to implement special strategies to evaluate the queries

Based on IBM DB2® Universal Database™ for Linux, UNIX®, and Windows®, WebSphere Information Integrator adds many new features to the data federation technology already available. DB2 Universal Database offers powerful facilities for accessing and combining information from diverse relational data sources (see [Resources](#)). The unique data federation capability in DB2 Universal Database builds on the best-of-breed technology from IBM DB2 DataJoiner® and has been enhanced with advanced features for extensibility and performance from IBM?s Garlic research project. (For more information about DataJoiner or Garlic, see [Resources](#).

This article is the first in a series that will show you how to use the data federation technology in WebSphere Information Integrator. For those who have a fundamental background in SQL and in using and administering relational databases such as DB2, this article provides a comprehensive guide to setting up and configuring a WebSphere Information Integrator federated database system. The second article will focus on usage examples and performance tuning.

### An e-commerce scenario

[Figure 1](#) illustrates an e-commerce scenario in which customers submit orders online. These orders, in the form of XML documents, are routed to the global warehouse, while customer information is maintained in a local database table called CUSTOMERS.

![Figure 1. The customer ordering scenario](#)

Using the federation technology in WebSphere Information Integrator, this global warehouse is connected to two regional warehouses in the U.S. and Canada. In each warehouse, information about items and suppliers is stored in tables ITEMS and SUPPLIERS. In addition, the item ID and the supplier ID for each item are stored in the table ITEM_SUPPLIED. The USA warehouse is based on a DB2 for z/OS and OS/390 system, while the Canada warehouse is in an Oracle system. Another Oracle instance, called the Credit Checking server, tracks customers with bad credit history and is accessible from the federated system.
Designing your federated system

The core of a WebSphere Information Integrator federated system consists of a DB2 instance that operates as a federated server. Other components include 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 use a single SQL statement to send distributed requests to multiple data sources. After you register the remote data sets on the federated system, you can reference them as easily as you would reference local tables. Applications communicate with the federated server through a supported programming interface. Because a federated system includes a DB2 database, you can also store local data in it, as well as combine information from both local and remote tables.

For the e-commerce example, Figure 2 below describes the column definitions of the existing tables that will be queried.

Figure 2 shows the entity-relationship diagram of the tables referenced in Figure 1. Both tables ITEMS and SUPPLIERS have check constraints defined on columns item_id and suppl_id respectively. The constraints specify the range in which the data lies.

Figure 2. Entity-relationship diagram of the tables

You are now ready to set up the federated system so that you can register these remote tables. The DDL (Data Definition Language) statements throughout this article are provided as a sample. For more detailed usage descriptions, refer to the IBM DB2 Information Integrator Data Source Configuration Guide, and volumes 1 and 2 of the DB2 UDB SQL Reference (see Resources). An alternative registration method is to use the enhanced graphical interface of the DB2 Control Center to register remote objects. The Control Center offers a remote object discovery feature, which enables automatic discovery of data sources and objects that can be included in the federation.
Configuring the federated system
You create a federated system by first installing the federated engine, then configuring it to communicate with the data sources. There are four basic federated objects:

- The federated server communicates with the data sources by means of software modules called wrappers.
- Each data source must be identified to the system as a server.
- If the data sources require authentication, the remote authentication information can be registered with the federated system as user mappings.
- Identify remote data sets that you want to access as nicknames to the federated system. You can now reference the nickname in your application as if it were a local table.

The following sections examine these federated objects one by one.

Wrappers
A wrapper module provides the logic to facilitate:

- **Federated object registration**: A wrapper encapsulates the data source characteristics from the federated engine. It knows what information is needed to register each type of data source.
- **Communication with the data source**: Communication includes establishing and terminating connections with the data source, and maintaining the connection across statements within an application if possible.
- **Services and operations**: Each wrapper supports various operations, depending on the capabilities of the data sources that the wrapper is meant to access. These operations can include sending a query to retrieve results, updating remote data, supporting transactions, manipulating large objects, binding input values, and more.
- **Data modeling**: A wrapper is responsible for mapping the data representation of remote query results into the table format as required by the federated engine.

WebSphere Information Integrator provides a set of wrappers for various types of data sources. You can also use the wrapper development kit included in the product to write a custom wrapper library, which will allow you to access data sources not supported by the existing wrappers. For the list of data sources supported by WebSphere Information Integrator, refer to the *IBM DB2 Information Integrator Data Source Configuration Guide* (see Resources).

When using a wrapper to access data, you will normally get the best performance if the wrapper is implemented in the native API of the data source. For example, you will most likely choose the NET8 wrapper to access your Oracle data sources, even though the ODBC wrapper can also provide such access.

You need to register only one wrapper to access all the data sources of the type that the wrapper supports. For example, if you need to access XML files, you need to register one XML wrapper.

A wrapper module can execute within the DB2 engine, which is referred to as the trusted mode. If you run it in a separate process outside the DB2 engine, it is referred to as the fenced mode. In
general, the trusted mode provides better performance if the federated database is not partitioned. However, the robustness of the wrapper module can directly affect the DB2 engine. If your federated server is a partitioned database, the fenced mode execution takes better advantage of parallelism. You can specify if a wrapper is to be executed in the fenced mode by setting the DB2_FENCED wrapper option to ?Y? (default setting for this option is ?N? for trusted). The article "Parallelism in WebSphere Information Integrator V8.2" by S. Harris (see Resources) discusses this in more detail.

In the example online shop scenario, you would need three wrappers: a NET8 wrapper for the two Oracle systems, a DRDA wrapper for the DB2 for z/OS and OS/390 system, and an XML wrapper to access the online orders from XML documents. To register the wrappers with our federated system, use these SQL statements:

```
CREATE WRAPPER net8;
CREATE WRAPPER drda;
CREATE WRAPPER xml_wrapper LIBRARY ?libdb2lsxml.a?;
```

Figure 3 shows the federated system configuration that follows this step. Three wrappers provide access to four data sources (including the XML file that contains the Web order). Some client libraries are required for the wrappers used here. For example, the Oracle NET8 client software is required for the NET8 wrapper. These client libraries are not shown in Figure 3.

**Figure 3. The federated system**

![Figure 3. The federated system](image)

**Servers**

After you register the wrapper libraries with the federated system, register each data source as a server. For a relational database management system, a server usually represents a remote database. If the data source is a remote DB2 Universal Database instance, you can register each database on that instance as a server. Registering each database as a server allows data on each of these remote databases to be accessed from the federated system.

Some relational database management systems do not allow the registration of multiple databases per instance. Instead, each instance represents a server. For example, on Oracle systems, each instance server ID (SID) represents a server.
In the case of our online store, we need to define four servers. DB2 for z/OS and OS/390 sources might require additional node and database entries to be cataloged. Use the following statements to accomplish this:

```
CATALOG TCPIP NODE mvsnode REMOTE hostname SERVER servicename;
CATALOG DB mvsdb2 AS mvsdb2 AT NODE mvsnode AUTHENTICATION SERVER;
CREATE SERVER usa_server TYPE db2/390 VERSION 7.1 WRAPPER drda
      AUTHORIZATION ?MVSUSER1" PASSWORD ?password"
      OPTIONS (DBNAME ?MVSDB2?);
```

Similarly, to register an Oracle server, you refer to it using the node name you configured in the Oracle client. This value is stored in the file `tnsnames.ora`. The SQL statements to complete the Oracle and XML server registration for the example federated system are:

```
CREATE SERVER canada_server TYPE oracle VERSION 9 WRAPPER net8
      OPTIONS (NODE ?nodename?);
CREATE SERVER credit_server TYPE oracle VERSION 8.1.7 WRAPPER net8
      OPTIONS (NODE ?nodename?);
CREATE SERVER xml_server WRAPPER xml_wrapper;
```

### Server options

The WebSphere Information Integrator federated system relies on server attributes to ensure that the capabilities of each data source are properly exploited. Server attributes on the federated server store the characteristics of each data source. The query compiler uses these characteristics and restrictions when planning the query. Using server options to set external server attributes, you can specify the data source location (machine node), connection security information (ID and password), and some server characteristics that affect performance. Each wrapper module maintains a set of server attributes pertaining to the type and version of the data sources it supports.

"Pushing down" an operation -- allowing it to take place on the remote data source -- can be very beneficial. For example, if certain SQL operations will reduce the amount of data returned to the federated server, pushing these down to the remote data source can reduce the amount of data that is brought to the federated system over the network, thus helping the performance of the query. The major challenge of allowing an operation to be pushed down is to ensure that the same query result can be achieved regardless of where the operation is performed.

Some attributes, such as `COLLATING_SEQUENCE`, can affect which operations can be pushed down to a data source for evaluation. This server attribute tells the federated server whether the collating sequence is the same on both the remote data source and the local federated server. Besides allowing string data to be sorted on a remote source, this server option also allows range comparison (for example, `string_col > string_constant`) and LIKE predicates to execute remotely. Depending on the operation, allowing it to be evaluated on a remote source might help the overall performance of the query.

For example, data sources such as DB2 for z/OS and OS/390 use a collating sequence based on the EBCDIC encoding scheme. The default setting for the server option `COLLATING_SEQUENCE` is 'N' for such sources, because the EBCDIC encoding scheme is specific for DB2 for z/OS and
OS/390. By default, DB2 Universal Database uses ASCII encoding, and the default sorting order is dictionary sort.

Although some other data sources like Oracle also use ASCII encoding, their sorting order differs from that of DB2 Universal Database. It is possible to configure the federated system to match the sorting order of sources like Oracle, which uses IDENTITY sort. To do this, use the additional clause COLLATE USING IDENTITY in the CREATE DATABASE statement. In this case, you can set the server option COLLATING_SEQUENCE to ?Y? for sources with ASCII encoding and identity sorting order.

The federated system provides the SET SERVER OPTION statement. Use this statement when you want a server option setting to remain in effect only while your application is connected to the federated server. When the connection ends, the previous server option setting is reinstated. For more detailed descriptions, refer to the DB2 UDB SQL Reference (see Resources). For the complete list of general and data source specific server options that you can tune, refer to the IBM DB2 Information Integrator Data Source Configuration Guide (see Resources).

User mappings

To provide an additional layer of security, WebSphere Information Integrator supports user mappings. You can create a mapping for each WebSphere Information Integrator user to an ID and password on a remote data source. Such user mappings can be defined for relational data sources as well as some nonrelational data sources. XML files do not require the registration of user mappings, and thus no additional layer of authentication is applied. For the example federated system, the SQL statements to register user mappings on behalf of a user are:

```
CREATE USER MAPPING FOR user SERVER usa_server
    OPTIONS (REMOTE_AUTHID ?mvsuser1?, REMOTE_PASSWORD ?password?);
CREATE USER MAPPING FOR user SERVER canada_server
    OPTIONS (REMOTE_AUTHID ?orauser1?, REMOTE_PASSWORD ?password?);
CREATE USER MAPPING FOR user SERVER credit_server
    OPTIONS (REMOTE_AUTHID ?crduser1?, REMOTE_PASSWORD ?password?);
```

If the user ID and password you use to connect to the federated database are the same as those you use to access the remote data source, you do not need to create a user mapping. However, if you do not use a user mapping, ensure that you explicitly specify the user ID and password when you connect to the federated database. For example:

```
CONNECT TO my_federated_db USER etlin USING etlin_pwd
```

Pass-through sessions and privileges

A pass-through session allows you to submit SQL statements and some subset of commands directly to a remote server. After remote authentication registration, if pass-through is supported for the data source, you can use it to test the connection to that data source. Testing the connection at this stage helps isolate configuration problems before you move on to nickname registration.
The SQL statements and commands submitted in a pass-through session do not go through the federated query compiler. DB2 Universal Database assumes that the pass-through session includes some 'write' operations against the remote server. As a result, if you have a transaction that contains a pass-through session to server A, the same transaction cannot perform any 'write' operation to either local data on the federated server, or to remote data on a different server that is not server A.

The administrator of the federated system controls which DB2 users can open a pass-through session to a remote server. Access is granted using the GRANT PASSTHRU statement. For example:

```
GRANT PASSTHRU ON SERVER credit_server TO GROUP managers;
```

After pass-through privileges are granted, the affected users (such as those belonging to the group managers in the above example) can use the pass-through facility. The following example initiates the RUNSTATS equivalent operation on the Oracle server:

```
SET PASSTHRU credit_server;
ANALYZE TABLE bad_credit COMPUTE STATISTICS;
SET PASSTHRU RESET;
```

Having pass-through privilege is not sufficient to access the data source through WebSphere Information Integrator. Users are required to have user mappings or an equivalent mechanism (refer to the previous section on User Mappings) in order to successfully connect to the specific data source.

**Nicknames**

Remote objects such as tables and views are registered on the federated server as nicknames. For relational nicknames, the wrapper validates the existence of the data source objects and retrieves the column definitions and index information, if any, when the nickname is defined. If the statistics maintained on the data source are similar to those on the federated system, the wrapper nickname registration function will look up and retrieve the statistics from the remote system catalog.

Accurate index information and statistics are fundamental to cost-based decisions in the query optimizer. On a federated system, unique index information might be required to perform an UPDATE/DELETE operation for some data sources. If data needs to be stored locally before it is updated (for example, when the table to be updated is also referenced on the same UPDATE statement in a predicate), the federated system uses a unique index to simulate positioned cursors for data sources that do not support rowid (for example, DB2 family). Information about the column definitions, index, and statistics for a nickname is stored in the DB2 federated database system catalogs.

The registration of nicknames provides location transparency, since a nickname looks just like a local DB2 table to the users on the federated server. Because relational data sources usually
provide system catalog information about the column definition of an object, the DDL syntax only needs to identify the remote object for which you are creating a nickname.

The following statements define the relational nicknames for the example scenario:

```sql
CREATE NICKNAME usa.items FOR usa_server.mvsuser1.items;
CREATE NICKNAME usa.suppliers FOR usa_server.mvsuser1.suppliers;
CREATE NICKNAME usa.item_supplied FOR usa_server.mvsuser1.item_supplied;
CREATE NICKNAME canada.items FOR canada_server.orauser1.items;
CREATE NICKNAME canada.suppliers FOR canada_server.orauser1.suppliers;
CREATE NICKNAME canada.item_supplied FOR canada_server.orauser1.item_supplied;
CREATE NICKNAME bad_credit FOR credit_server.crduser1.bad_credit;
```

Because the syntax for defining nicknames for most nonrelational sources contains more details, you might need to provide the column definitions when you register a nickname for some sources (such as table-structured flat files). The Control Center in WebSphere Information Integrator provides an automatic nickname DDL generation facility for XML nicknames, so the DDL statements provided here are for reference only. Because a customer order usually contains a list of items, it is represented using two nicknames.

```sql
CREATE NICKNAME xml.orders
(order_id char (10) OPTIONS (XPATH ?@id?),
 order_date date OPTIONS (XPATH ?date/text()?),
 customer_id char (10) OPTIONS (XPATH ?cid?),
 order_amount decimal (31, 2) OPTIONS (XPATH ?./amount/text()?),
 oid varchar (16) OPTIONS (PRIMARY_KEY ?YES?))
FOR SERVER xml_server
OPTIONS (FILE_PATH ?/home/administrator/orders/orders.xml?,
 XPATH?//order?);

CREATE NICKNAME xml.order_items
(oid varchar (16) OPTIONS (FOREIGN_KEY ?ORDERS?),
 item_id char (10) OPTIONS (XPATH ?./item_id/text()?),
 item_quantity integer OPTIONS (XPATH ?./quantity/text()?))
FOR SERVER xml_server
OPTIONS (XPATH?//item?);
```

**Nickname options**

Changing certain nickname attributes can improve performance by influencing which operations can be considered for pushdown.

During the registration of a data source object on a relational data source, default type mappings determine how a data source data type should map to a DB2 type for each column. These mappings are built into the wrapper modules. For the list of default type mappings for each data source, refer to the [IBM DB2 Information Integrator Federated Systems Guide](#) (see Resources).

You might want to change a column data type to a different type to match the data more closely. For example, the Oracle DATE type is mapped to the DB2 TIMESTAMP type by default. But if a column of type DATE is only used to store the date of birth of employees, you can map it to the DB2 DATE type. Defining the local column as DATE is convenient because it allows predicates that involve DATE literals, but does not require casting functions (which tend to impact performance).
With the ALTER NICKNAME statement, you have the option to change the column data type on the federated server for a remote object. You can also use this statement to customize the name of a nickname column. By default, the column name is set to the remote name of the same nickname column when the nickname is registered.

There are at least two column options that you can set to improve the federated server’s understanding of the remote data and encourage more operations to be pushed down. These are VARCHAR_NO_TRAILING_BLANKS and NUMERIC_STRING.

Use the VARCHAR_NO_TRAILING_BLANKS column option to identify a nickname column that contains no trailing blanks. The query compiler uses this information while checking any character comparison operations, to decide how to evaluate the operations. DB2 Universal Database uses blank padded comparison semantics -- that is, while comparing character strings of unequal length, a copy of the shorter string is padded on the right with blanks so that its length is equal to that of the longer string. This means that the string ‘A’ is considered equivalent to ‘A ’ in DB2 Universal Database.

However, this behavior does not apply to all character data types across all data sources. For example, the VARCHAR2 data type in Oracle does not have this behavior. In general, comparisons on string columns without blank padding comparison semantics need to be evaluated locally, unless the query compiler is able to find functions to enforce similar logic remotely. The fundamental premise is that query results should be identical regardless of where any portion of the query is evaluated.

To compensate for the difference in blank padding comparison semantics, for certain operations such as predicates, the federated system rewrites the predicates to ensure the same semantics when these predicates are sent to an Oracle server. Performance of operations such as DISTINCT, ORDER BY, GROUP BY, UNION, column functions (MIN()MAX()) evaluation, relational comparison and IN predicates might be affected. Setting the VARCHAR_NO_TRAILING_BLANKS column option generally improves performance, because it allows more operations to be pushed down and more optimization of the remote SQL statements to encourage index access.

VARCHAR_NO_TRAILING_BLANKS can also be set as a server option, if you are sure that all VARCHAR2 columns from this Oracle server have no trailing blanks. In general, such a restriction is hard to impose. Therefore, using VARCHAR_NO_TRAILING_BLANKS as a server option is not recommended.

NUMERIC_STRING is another column option that affects performance. It applies to character data types and those data sources for which the COLLATING_SEQUENCE server option is not set to ‘Y.’

If there are differences in collating sequences between the federated system and the data source, the federated system will consider pushing down an operation only if an identical query result can be achieved regardless of where the operation is evaluated. However, if the column is a character data type and contains only numeric characters, you can indicate this by setting the NUMERIC_STRING option to ‘Y.’ This setting gives the federated compiler the option of allowing
the data source to perform operations that are affected by different collating sequences -- because numbers are always sorted the same.

For our example, item_id is a CHAR column that stores only numeric characters, as we indicate in the following statement:

```sql
ALTER NICKNAME usa.items ALTER COLUMN item_id OPTIONS (ADD NUMERIC_STRING ?Y?);
ALTER NICKNAME usa.item_supplied ALTER COLUMN item_id OPTIONS (ADD NUMERIC_STRING ?Y?);
ALTER NICKNAME canada.items ALTER COLUMN item_id OPTIONS (ADD NUMERIC_STRING ?Y?);
ALTER NICKNAME canada.item_supplied ALTER COLUMN item_id OPTIONS (ADD NUMERIC_STRING ?Y?);
```

### Informational constraints on nicknames

If the remote table includes attributes such as primary key constraints or other forms of constraints, they can be captured in the federated system through the specification of informational constraints on nicknames. Unique, primary key, referential, check, and functional dependency constraints are supported. The SQL compiler uses informational constraints to improve query performance. They provide more information about the data, and enable more optimization. The informational constraints are not enforced by the federated server during operations such as insert, delete and update.

In the example data model, columns item_id and suppl_id are primary keys in the remote tables, items and supplier. For data sources such as DB2/390 and Oracle, the federated system automatically retrieves remote primary key information during the CREATE NICKNAME command. A simple way to validate that the federated system captures the primary key information is to query the catalog view SYSCAT.TABCONST with the following statement:

```sql
SELECT CHAR(constname, 30), type
FROM syscat.tabconst
WHERE tabname = 'ITEMS' and tabschema = 'USA';
```

When a primary key constraint is defined, the federated system automatically creates a unique index specification.

In addition, columns item_id and suppl_id are defined as foreign keys for the remote table item_supplied. You can convey this information to the federated system by defining informational referential constraints, using the following commands:

```sql
ALTER NICKNAME canada.item_supplied ADD CONSTRAINT fk1
  FOREIGN KEY (item_id) REFERENCES canada.items(item_id)
  NOT ENFORCED;
ALTER NICKNAME canada.item_supplied ADD CONSTRAINT fk2
  FOREIGN KEY (suppl_id) REFERENCES canada.supplier(suppl_id)
  NOT ENFORCED;
```

You can use similar definitions to define foreign key constraints on usa.item_supplied nickname. The NOT ENFORCED clause simply specifies that the constraint is not enforced at the federated
server. It will only be used by the SQL compiler to optimize the federated statement. If you want to disable the usage of the constraints by the SQL compiler, you can use the DISABLE QUERY OPTIMIZATION clause. For example:

```
ALTER NICKNAME canada.item_supplied ALTER FOREIGN KEY fk1 DISABLE QUERY OPTIMIZATION;
```

If the remote tables have restrictions on the data such as check constraints, you can convey these restrictions to the federated server using informational check constraints. Knowing such constraints can help the federated server to further simplify user queries.

Given the remote table definitions for items and suppliers, you can declare check constraints on the corresponding nicknames with the following commands:

```
ALTER NICKNAME canada.items ADD CONSTRAINT ck
CHECK (item_id BETWEEN '0000000001' AND '5000000000');
ALTER NICKNAME usa.items ADD CONSTRAINT ck
CHECK (item_id BETWEEN '5000000001' AND '9999999999');
```

Define similar constraints for the supplier nicknames:

```
ALTER NICKNAME canada.supplier ADD CONSTRAINT ck
CHECK (suppl_id) BETWEEN '0000000001' AND '2000000000');
ALTER NICKNAME usa.supplier ADD CONSTRAINT ck
CHECK (item_id) BETWEEN '2000000001' AND '3000000000');
```

If informational check constraints exist on the primary key column, you should explicitly specify informational check constraints on their corresponding foreign key columns. For example:

```
ALTER NICKNAME canada.item_supplied ADD CONSTRAINT ck1
CHECK (item_id) BETWEEN '0000000001' AND '5000000000');
ALTER NICKNAME canada.item_supplied ADD CONSTRAINT ck2
CHECK (suppl_id) BETWEEN '0000000001' AND '2000000000');
```

Add similar information to the usa.item_supplied nickname.

### Index specifications

During the registration step for relational nicknames, the wrapper modules try to retrieve remote catalog information for the underlying object. Sometimes, it is not possible to locate index information --- a remote data source might not have a system catalog for indexes, or the remote object might not appear to have any indexes associated with it in the remote system catalog. It is also possible that a remote index is added to the remote object on the data source after the creation of the nickname. You can define an index specification for this remote object, to make the federated server aware of the index and facilitate the access of this remote object. The syntax of this statement is an extension to the CREATE INDEX statement (see the DB2 UDB SQL Reference for more information). No physical index is built at the federated server on behalf of this nickname. An entry is added to the system catalog to indicate to the query optimizer that such a remote index exists. Awareness of all applicable remote indexes can help the federated query optimizer generate better execution plans.
An index specification that defines a unique index also conveys information about the uniqueness of the index columns to the federated system. Just like a regular unique index definition registered during relational nickname registration, such information can help the query optimizer to generate a more optimal plan with strategies such as eliminating unnecessary DISTINCT operations.

Nicknames over remote views

A nickname can be defined over a remote view just as it can be defined over a remote table. Defining a nickname over a remote view is useful in cases where you want to push down the query defining the remote view to the data source. This is useful in performance testing. However, it is not a recommended long-term solution for most general situations.

There is usually no index or statistics information provided at the remote data source on a view. If the view can use an index internally, you can create an index specification that is stored in the DB2 database system catalog on the federated server, and used by the query optimizer.

Accurately representing the statistics characteristics for a remote view is tricky. If the view represents a single table on the remote data source, you can use a simple program to update the nickname statistics in the DB2 federated database system catalog with the statistics of the underlying base table.

If the remote view contains a complex query, using table-level statistics such as the cardinality for the nickname over the view can be misleading. For example, a view based on a very complex query that returns only five rows (for example, a GROUP BY query) is costly to evaluate at the remote source. However, if the cost of a view representing this complex query is to be modeled with table-level statistics at the federated server (say, table cardinality), accessing five rows makes the cost of evaluating this remote view appear very cheap. This will have an undesirable effect on query optimization.

A better solution is to define a nickname over each remote table, and to define views over these nicknames on the federated server if necessary. This allows the query optimizer to do a better job of analyzing the complex query, as the statistics can now closely reflect the actual cost.

Nickname statistics

WebSphere Information Integrator uses a cost-based optimizer, so for the optimizer to arrive at an optimal plan, the availability of accurate statistics is crucial. Currently, you can collect statistics for nicknames using any one of the following methods:

- **Nickname creation:**
  When a nickname is created, the wrapper collects certain statistics for the nickname to be created by looking at the statistics available on the remote data source catalogs and mapping those statistics to the DB2 statistics. Currently, the relational wrappers attempt to collect and map the following statistics (when appropriate):

<table>
<thead>
<tr>
<th>Table 1. Table statistics</th>
</tr>
</thead>
<tbody>
<tr>
<td>Statistic</td>
</tr>
<tr>
<td>-----------</td>
</tr>
</tbody>
</table>

Using data federation technology in IBM WebSphere Information Integrator: Data federation design and configuration
card (table cardinality)  total number of rows in a table
npages  total number of pages on which the rows of the table exist
fpages  total number of pages
overflow  total number of overflow records in the table

Table 2. Column statistics

<table>
<thead>
<tr>
<th>Statistic</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>colcard (column cardinality)</td>
<td>number of distinct values in the column</td>
</tr>
<tr>
<td>high2key</td>
<td>second highest value of the column</td>
</tr>
<tr>
<td>low2key</td>
<td>second lowest value of the column</td>
</tr>
</tbody>
</table>

Table 3. Index statistics

<table>
<thead>
<tr>
<th>Statistic</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>firstkeycard</td>
<td>number of distinct first key values</td>
</tr>
<tr>
<td>fullkeycard</td>
<td>number of distinct full key values</td>
</tr>
<tr>
<td>nleaf</td>
<td>number of leaf pages</td>
</tr>
<tr>
<td>nlevels</td>
<td>number of index levels</td>
</tr>
<tr>
<td>clusterratio</td>
<td>degree of data clustering with the index</td>
</tr>
</tbody>
</table>

It is important that you run the equivalent of the 'runstats' utility on the data source, so that nickname creation will pick up the current statistics. Not every wrapper may be able to collect all the statistics listed above, because a correct mapping between the data source statistic and its counterpart on the federated system may not exist. Once the statistics are gathered for a nickname, they are not automatically modified when the remote statistics are updated on the data source for the corresponding remote table.

If you have a nickname whose statistics need to be updated to reflect the new remote statistics on its data source, you might be tempted to drop and recreate the nickname. However, doing so would also drop other database objects, such as indexes that depend on this nickname. Two ways to update statistics -- without dropping of objects dependent on the nickname -- are outlined below.

- **The nickname statistics update facility in Control Center or the SYSPROC.NNSTAT stored procedure on command line:**
  The nickname statistics update facility in Control Center, and the SYSPROC.NNSTAT stored procedure on the command line, both retrieve the remote statistics stored for a remote object corresponding to an existing nickname. Internally, a shadow nickname is created on the remote table. This nickname receives the latest statistics at nickname creation time. These new statistics are then copied from the shadow nickname to the existing nickname. The shadow nickname is then dropped.
  For detailed information on how to use the nickname statistics update facility in Control Center or the stored procedure, please refer to the IBM DB2 Information Integrator Federated Systems Guide (see Resources).

- **The getstats utility:**
The `getstats` utility sends queries to the data source in order to estimate the statistics. For example, table cardinality is estimated as `select count(*) from nick`.

This method of collecting statistics is particularly helpful for nicknames over remote views, because the data sources typically do not collect statistics on views. It is useful for ODBC nicknames as well, because the ODBC interface can connect to any data source, and the query optimizer is not aware of the mapping of statistics between the remote catalog and the DB2 system catalogs.

Since the `getstats` utility can only collect statistics that can be measured with SQL queries, it cannot collect physical characteristics of the data (such as the `npages` and `fpages` table statistics, or the `nleaf` and `nlevels` index statistics). Also, because the `getstats` utility issues queries against the data in the nickname, the performance of this utility depends on the amount of data referenced by the nickname. For more information on how to invoke the `getstats` utility, refer to the WebSphere Information Integrator support site (see Resources).

**Nickname privileges**

Just like tables, nicknames have an equivalent set of privileges that can control user access to the nickname objects on the federated system.

```sql
GRANT SELECT ON eileen.nick1 TO PUBLIC;
GRANT SELECT ON usa.items TO GROUP managers;
```

Even though the user eileen has the proper privilege to access a nickname, the underlying remote object privilege will be rechecked by the remote data source against the remote user id that the federated system uses to establish the connection on behalf of eileen. Figure 4 illustrates the difference.

**Figure 4. Comparing nickname privileges with table privileges**

![Diagram comparing nickname privileges with table privileges](image)

**Nickname privileges in views and packages**

Views and packages provide an additional layer of security by shielding you from the internal details. When you are granted privileges to use a view or a package, you automatically get access to the tables referenced in the view or package. However, for nicknames, another layer of privilege checking is performed on the data source: the remote authentication required to connect to the remote data source. Figure 5 illustrates the difference in the behaviors.
Functions

In addition to default data type mappings, the wrapper libraries included in WebSphere Information Integrator also contain a set of default function mappings. These instruct the query compiler to map a DB2 built-in function (such as SUM) to a relational data source equivalent, if possible. If you want the federated server to push down the evaluation of a function to a relational data source, you must use a function mapping for that data source. This can improve query performance in general.

If there is some special function specific to a data source and not available on WebSphere Information Integrator, you need a function template definition for that function. A function template is like an extension to a user-defined function. It is required to allow queries to reference those functions on the data source that do not have corresponding functions on the federated server. For the CREATE FUNCTION statement extension for function templates, refer to the DB2 UDB SQL Reference (see Resources).

Figure 6 shows the issues to consider in determining when a function template needs to be registered.

When a query references a function template, the query optimizer tries to generate a feasible plan that allows this function reference to be evaluated on the remote data source. If this is not possible, an error SQL0142N is returned to indicate that this SQL statement cannot be supported.
The federated system catalog views

Just as in local tables, all the data registration steps described above store the information in the DB2 federated database system catalogs. You can use those system catalogs to verify the information about federated objects.

Table 4 lists the respective federated objects and the DB2 system catalog views that can be used to locate the information. For more information about the definition of each catalog view, refer to the DB2 UDB SQL Reference (see Resources).

Table 4. Federated objects information in the catalog views

<table>
<thead>
<tr>
<th>Federated objects</th>
<th>SYSCAT catalog views</th>
<th>Descriptions</th>
</tr>
</thead>
<tbody>
<tr>
<td>Wrappers</td>
<td>SYSCAT.WRAPPERS</td>
<td>These two views display registered wrappers and their specific options.</td>
</tr>
<tr>
<td></td>
<td>SYSCAT.WRAPOPTIONS</td>
<td></td>
</tr>
<tr>
<td>Servers</td>
<td>SYSCAT.SERVERS</td>
<td>These two views display registered remote data sources and their specific options.</td>
</tr>
<tr>
<td></td>
<td>SYSCAT.SERVEROPTIONS</td>
<td></td>
</tr>
<tr>
<td>User mappings</td>
<td>SYSCAT.USEROPTIONS</td>
<td>This view displays the registered user authentications for specific servers for a DB2 user. The password setting is stored encrypted.</td>
</tr>
<tr>
<td>Nicknames</td>
<td>SYSCAT.TABLES</td>
<td>This set of views display information related to registered nicknames. In SYSCAT.TABLES, nicknames are identified by the TYPE column set to '?N?'. SYSCAT.TABOPTIONS displays specific options about nicknames. SYSCAT.COLOPTIONS displays specific options about nickname columns. SYSCAT.INDEXCOLUSE lists the columns participated in an index. SYSCAT.KEYCOLUSE stores information about the primary key.</td>
</tr>
<tr>
<td></td>
<td>SYSCAT.TABOPTIONS</td>
<td></td>
</tr>
<tr>
<td></td>
<td>SYSCAT.COLUMNS</td>
<td></td>
</tr>
<tr>
<td></td>
<td>SYSCAT.COLOPTIONS</td>
<td></td>
</tr>
<tr>
<td></td>
<td>SYSCAT.INDEXES</td>
<td>These two views display index specifications created for nicknames.</td>
</tr>
<tr>
<td></td>
<td>SYSCAT.INDEXCOLUSE</td>
<td></td>
</tr>
<tr>
<td>Index specifications</td>
<td>SYSCAT.INDEXES</td>
<td>These two views display index specifications for nicknames.</td>
</tr>
<tr>
<td></td>
<td>SYSCAT.INDEXCOLUSE</td>
<td></td>
</tr>
<tr>
<td>Informational constraints</td>
<td>SYSCAT.TABCONST</td>
<td>This set of views display informational constraints defined for nicknames. SYSCAT.TABCONST displays each defined constraint. SYSCAT.CHECKS and SYSCAT.COLCHECKS display information about check constraints.</td>
</tr>
<tr>
<td></td>
<td>SYSCAT.CHECKS</td>
<td></td>
</tr>
<tr>
<td></td>
<td>SYSCAT.COLCHECKS</td>
<td></td>
</tr>
<tr>
<td></td>
<td>SYSCAT.CONSTDEP</td>
<td></td>
</tr>
<tr>
<td></td>
<td>SYSCAT.REFERENCES</td>
<td></td>
</tr>
</tbody>
</table>
SYSCAT.CONSTDEP lists dependent objects of constraints.
SYSCAT.REFERENCES lists referential constraints.

<table>
<thead>
<tr>
<th>Type mappings</th>
<th>SYSCAT.TYPEMAPPINGS</th>
<th>This view displays user-defined type mappings used in nickname registration and remote table creation. Default built-in type mappings are not stored in this catalog view.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Function templates</td>
<td>SYSCAT.FUNCTIONS</td>
<td>SYSCAT.ROUTINES</td>
</tr>
<tr>
<td>Function mappings</td>
<td>SYSCAT.FUNCMAPPINGS</td>
<td>SYSCAT.FUNCMAPOPTIONS SYSCAT.FUNCMAPPARMOPTIONS</td>
</tr>
<tr>
<td>Passthru privileges</td>
<td>SYSCAT.PASSTHRUAUTH</td>
<td>This view displays authorization to allow users to query a specific server using PASSTHRU.</td>
</tr>
</tbody>
</table>

**Conclusion**

With version 7 of DB2 Universal Database for Linux, UNIX, and Windows, IBM introduced the first commercial information management system capable of integrating both relational and nonrelational data. Federation takes you to the next level of information integration, allowing the DB2 federated system to act as a virtual database where remote objects can be queried like local tables. You can leverage the power of DB2 as well as remote data sources, and gain from the transparency, heterogeneity, high function, autonomy, extensibility, and optimization features that such a federated system enables.

WebSphere Information Integrator V8.2 further enhances federation technology, making it easier to build an enterprise federated system. Some of the important features include:

- Materialized query tables that can be defined over queries referencing both relational and nonrelational nicknames to locally cache the query results
- The cache table facility in the Control Center, which lets you set up local caches of nicknames using replication as the refresh mechanism
- The nickname statistics update facility, which allows you to update nickname statistics
- The federated health monitor, which monitors the health of your federated system
- The ability to import data into a relational nickname and export data from a query involving a nickname
- An enhanced process model, which takes better advantage of parallelism for queries involving nicknames in a federated system using the data partitioning feature (as described in “Parallelism in WebSphere Information Integrator V8.2” by S. Harris -- see Resources)
- A broader set of nonrelational wrappers that includes the WebSphere Business Integration (WBI) and Web Services wrappers
- A wrapper development kit in both C++ and Java, which allows you to write your own wrappers to access proprietary data sources from your federated system

With more and more customers adopting data federation, the challenges of enabling seamless federation continue to grow. We will continue to focus on making the federated systems...
more performant and ensuring better overall user experience. Stay tuned for the next set of enhancements in WebSphere Information Integrator from IBM.

© Copyright IBM Corporation 2005
Trademarks
(www.ibm.com/developerworks/ibm/trademarks/)