The Federation: Database Interoperability (Part 1)

Marty Lurie

April 24, 2003

Introduction

If you have many flavors of databases, you've already discovered getting them to talk to each other is a chore. This article will make your life easier by addressing interoperability between databases. We'll use the IBM strategic federated database technology, the core of the IBM®DB2® Information Integrator. We'll examine how this technology works, the critical technical success factors, and how the IBM implementation stacks up when tested against these factors. In the second installation of this article we'll also look at insert performance and how to federate servers to generate XML from data stored in multiple sources.

Why do I need database federation, and by the way, what is it?

As a DBA you probably didn't volunteer to have many different brands of databases - I haven't yet met a DBA who said, "yes, please, I'd like one of every different flavor, I'm not busy enough." Significant productivity gains are to be had if you can work on all the different databases, including selects, inserts, updates, and deletes as if all the tables lived in a single database. Database federation does exactly that: it makes the tables look like they are all in the same database. The following sample SQL illustrates just how powerful this can be:

```
db2 => insert into remote_informix
       select a.customer, b.balance, c.limit
       from remote_db2 a, remote_oracle b, remote_sybase c
       where c.limit  > 10000
       and a.c_key = b.c_key
       and a.c_key = c.c_key;
```

Database federation may eliminate the need for building a datamart! An example of performance optimization for datamart replacement is shown below. If the volume of queries is not large, and if they can often be satisfied with summary tables, there is a huge productivity boost by eliminating the need for a datamart and the corresponding need to create a new server, move
significant quantities of data, and so on. Of course, for frequent heavy-duty queries that need access to the lowest level of detail, a datamart or data warehouse is the preferred solution.

**How does database federation work?**

Just dandy. Oh, sorry, time to get technical...

Consider the system diagram shown here:

**Figure 1. Members of a federation**

The "federator" system operates on the tables in the remote systems, the "federatees". The remote tables appear as virtual tables in the "Federator" database. Client application programs can perform operations on the virtual tables in the "Federator" database, but the real persistent storage is in the remote database. We'll look at a sample client program that does inserts in the next installment of this article.

Each "federatee" views the "federator" as just another database client connection. The "Federatee" is simply servicing client requests for database operations. The "federator" needs client software to access each remote database. Client software for IBM Informix®, Sybase, Oracle, etc. would need to be installed to access each type of federatee.

The application interface to the database federation is SQL. This provides a significant productivity boost over having to learn a new interface. The remote tables are accessed with the same syntax as local tables for select, insert, update, and delete. Not all table operations are possible, but Information Integrator in DB2 v8 takes a big step forward by providing insert and update functionality.

**Installation and configuration**

This is the summary and highlights of the install and configuration process. Don't forget to read the manual (RTM!). You'll need to install DB2 v8.1 with the Information Integrator options. When doing the install select **Custom**, and under the **Server Support** options enable "Informix Data Source Support". Informix wrappers currently ship with the 8.1 server; other wrappers are in beta and will be available shortly. For hints and tips on Linux kernel parameters, applicable to v7 and v8, see my previous article, **Simulating Massively Parallel Database Processing on Linux!** For best practices on configuring Informix remote servers see my article ** Winning Database Configurations: An IBM Informix Database Survey**.

For the impatient, take a look at **Appendix A**, which contains the complete SQL to set up a federated environment.
The steps for setting up federation are as follows:

**Step 1: Set up client connectivity between the federator and the remote database server**

Before attempting a federated configuration, you need to get the remote client software operational. Testing a simple client program, for example dem01.ec that comes with Informix, or the Oracle or Sybase equivalent, to validate connectivity. If the client software can't access the remote server you can't get out of the starting blocks.

**Step 2: Create the db2dj.ini file containing parameters required for the federator**

The `db2dj.ini` file is required for both DB2 v7 and DB2 v8. This file provides the parameters and paths for the remote databases. The sample shown here references two remote servers, one IBM Informix Extended Parallel Server™ (XPS), one Informix Dynamic Server™ (IDS). Here is a sample from a Windows® 2000 machine:

```
C:\Program Files\IBM\SQLLIB\cfg>type db2dj.ini
INFORMIXDIR=C:\PROGRA~1\Informix\CLIENT~1
INFORMIXSERVER=fliif
INFORMIXSQLHOSTS=C:\tmp\sqlhosts
INFORMIXSERVER=flxps
```

This example illustrates how to access two remote servers: one is IBM Informix version 9, and the other is IBM Informix XPS.

You need to tell the database how to get to this file. Make sure you use the fully qualified path name. Don't use a relative path name--that would be bad. For Windows 2000 use:

```
DB2_DJ_INI=c:\progra~1\ibm\sqllib\cfg\db2dj.ini
```

For UNIX® or Linux use:

```
db2set DB2_DJ_INI=/home/db2inst1/sqllib/cfg/db2dj.ini
```

**Step 3: Create the "wrapper" for the remote database**

The "wrapper" defines a library file that understands how to communicate with the federatee database. It uses the client connectivity software you set up in Step 1 to access the remote database.

There are two ways to define the wrapper, using SQL, or in DB2 v8 using a graphical user interface (GUI).

An example of creating a wrapper using SQL is shown here:

```
CREATE WRAPPER "INFORMIX" LIBRARY 'db2informix.dll';
```
The GUI wizard is launched from the Control Center and is shown in Figure 2. Right click on the **Federated Database Objects** identifier in the tree navigation pane of the database you want to configure. This is shown in the illustration. I encourage you to use the button **show SQL** when you use the GUI. It will provide a much better understanding of what is happening behind the scenes (see Figure 3, for an example). Think of the **show SQL** button as your private DBA tutor.

**Figure 2. Using the Control Center to create a wrapper**

![Figure 2. Using the Control Center to create a wrapper](image)

**Figure 3. Behind the scenes of the GUI**

![Figure 3. Behind the scenes of the GUI](image)

**Step 4: Define the federated server - the remote database instance**

Use the CREATE SERVER statement (or the Control Center) to define the remote instance. Notice we use the wrapper created in the prior step. There are quite a number of options to specify the relative CPU and I/O speed, as well as network speed and a number of other parameters. IBM recommends accepting the defaults for most parameters. The parameters are shown only as an example of how these can be specified.

Here's an example of the SQL:

```
CREATE SERVER "rcfliif"
  TYPE INFORMIX VERSION '9.3'
  WRAPPER "INFORMIX" OPTIONS ( NODE 'fliif',
    DBNAME 'stores_demo',
    ADD CPU_RATIO '1',
    ADD IO_RATIO '1',
    ADD COMM_RATE '1',
    ADD DB2_MAXIMAL_PUSHDOWN 'Y');
```
The NODE specifies the remote database server and is not a TCP/IP hostname. The DBNAME defines the remote database.

PUSHDOWN is the default and it indicates that joins should take place on the remote server where possible. We'll test this and look at some database optimizer explain plans in a later section. DB2_MAXIMAL_PUSHDOWN is an undocumented parameter (documented here) that tells the federator to send SQL and do the join at the "federatees" even if the optimizer thinks it would be better off extracting the data and doing the join locally.

Other options you may want to use include FOLD_ID and FOLD_PW with a setting of 'N'. That causes federated to connect to data source with the user ID and password exactly as they are in the User Mapping (see Step 5). If not set, the federated server attempts the connection four times, with the first attempt with user ID and password both in upper case and the fourth try with user ID and password in lower case. Setting FOLD_ID and FOLD_PW to 'N' will make the connection data source a little faster if the userID and password are in lower case as is frequently the case for Unix, Linux, and Windows systems.

Step 5: Create user mapping for federator authentication

The federator wants to get data from the remote databases. It needs to authenticate just like any other client program. The "User Mapping" provides the mechanism to establish the credentials of the data request.

A local db2 user ID is mapped to a userid on the remote server. The example maps from the local ID "LURIE" to the remote "informix" ID.

```
CREATE USER MAPPING FOR "LURIE"
SERVER "rcfliif" OPTIONS( REMOTE_AUTHID 'informix', REMOTE_PASSWORD 'useyourown');
```

The DB2 Control Center also has a GUI definition mode for user mappings. This is illustrated in Figure 4:

Figure 4. Mapping user IDs using the Control Center
Step 6: Table nicknames - your ticket to the remote table

So far we've accomplished the following:

• The wrapper: the type of server is defined and the binaries to access it are identified.
• The server: the location of the remote server and the database on that server is defined.
• User mapping: the authentication to the remote server is defined.

Now all we need are some tables!

Each table is defined to the Federator via a nickname. Once the nickname is defined we use it like a local table name. The example shows the remote table "card1" owned by user "lurie" on the remote server "rcfliif" is defined as nickname "rc9_card1".

```
create nickname rc9_card1 for "rcfliif"."lurie"."card1";
```

Congratulations! With the nickname in place we can now do exciting things like:

```
select * from rc9_card1;
```

Table names can provide useful identifiers. This naming convention is rather simple. The "rc" is short for Relational Connect. The "9" indicates the remote server is version 9 of Informix. Finally "card1" is the table name. The table has several generated columns of different cardinality. See Appendix B for the source code to create the table.

By now I'm sure the GUI fans are expecting a screen shot and far be it for me to disappoint you. First, a sample of the filter dialog to limit the tables presented as candidates for nicknames (Figure 5):

**Figure 5. Filtering tables**

![Filter Tables for Nicknames](image)

After the filtering process a list of tables are presented (Figure 6). In this case a single table meets the filter qualifications.
Figure 6. Tables that meet the filter criteria

Click OK, which brings up Figure 7, a sample of the data in the newly created nickname.

Figure 7. Tables that meet the filter criteria

Federated joins

After you have configured a federation of servers it is time to do some testing. The environment I used is shown in Figure 8. The SQL to create this environment is listed in Appendix A.

The first query we'll look at will use two tables "rc9_card1" and "rc9_card2" in the remote IDS9 server. We'll then optimize over all four remote tables using some advanced techniques with materialized data caching.
Where to perform the join?

The choice of where to process the table joins is made by the federator cost-based optimizer. This choice is one of the most critical aspects of federation performance. The optimizer can choose to send the SQL to the remote server and join the tables using the remote engine. This is known as pushdown. Alternately the federator can retrieve the individual rows from the remote server and perform the join using the federator join engine.

Let's work with the query:

```sql
select a.c3, a.c10, sum(a.c100 ) as sum_c100, sum(a.c1000) as sum_c1000
from rc9_card1 a, rc9_card2 b
where a.superkey = b.superkey
group by a.c3, a.c10;
```

*Figure 9* shows the explain plan for pulling the individual rows from the remote server and processing them in the federator engine. The total cost (timerons) of this join, is over 45 million.
Figure 9. Plan for a federated join without pushdown

There are several important factors to consider in deciding where the join is best performed:

- The relative speed of the federator CPUs versus the speed of the federatee CPUs
- The relative speed and throughput of the I/O subsystems
- The speed of the network
- The capabilities of the federatee database

The **Create Server** configuration in the Control Center shows a number of parameters that can be tuned based on the relative speeds of the servers (Figure 10).

Figure 10. Using the Control Center to tune performance parameters used with a particular federatee

As a general rule of thumb, pushdown is a good thing and should occur as much as possible when the federatee database is capable of punching it's way out of a heavy-duty paper bag. For the
feeble remote sources, such as a collection of flat files, joins are best performed (in some cases must be performed) by retrieving all the rows and joining at the federator server.

When pushdown occurs, the query plan looks quite different and is shown in Figure 11. The total cost (timerons) is significantly reduced from the non-pushdown case.

The optimizer is actually doing some very deep thinking about where to do the join. I've tried to keep this as simple as possible. Don't get caught up in the theory of distributed joins - there are too many interesting applications of federated technology to dwell on the theory. Up-to-date statistics at both the federated server and the remote server will improve the query plans.

Figure 11. Pushdown join performs much faster

![Pushdown join diagram](image)

Faster federated joins - materialized query tables

The fastest way to join two tables is not to join them at all. The Transaction Processing Council (TPC) TPC-D benchmark, now withdrawn, proved conclusively that if the answer to a query is pre-computed in a summary table, the query runs faster.

How can this summary table speed-up technique be applied to a federated environment? To really take advantage of summary tables, they have to be used whenever possible. A summary of data at a monthly interval can satisfy a query at a monthly, quarterly, or annual level of aggregation. But the optimizer must be smart enough to rewrite the query to operate on the summary table instead of seeking the detail. We'll test this capability.

Adding a summary table to the federator is very similar to a non-federated environment. In DB2 v8 these tables are called materialized query tables (MQTs). A powerful feature of the federator is to create the MQTs on nickname (remote) tables.

Adding a materialized query table

The MQT contains a select statement to define how the data should be summarized. The SQL to implement an MQT is shown below:

```
--create a materialized query table or MQT
-- this will allow the optimizer to rewrite SQL
-- and not access the remote servers
```
drop table card_mqt;
create summary table card_mqt
as (
    select a.c3, a.c10, sum(a.c100 ) as sum_c100, sum(a.c1000)
    as sum_c1000
    from rc9_card1 a, rc9_card2 b, rc8_card1 c, rc8_card2 d
    where a.superkey = b.superkey
        and b.superkey = c.superkey
        and c.superkey = d.superkey
    group by a.c3, a.c10 )
data initially deferred refresh deferred;

-- populate the MQT with data
refresh table card_mqt;

-- VERY IMPORTANT - tell the optimizer the MQT
-- alive and well and open for business
set current refresh age=any;

So how much of an improvement does the MQT provide?
First consider a four-table query across two different servers. The most costly query is without pushdown. The query SQL is:

select a.c3,  sum(a.c100 ) as sum_c100
from rc9_card1 a, rc9_card2 b, rc8_card1 c, rc8_card2 d
where a.superkey = b.superkey
    and b.superkey = c.superkey
    and c.superkey = d.superkey
    group by a.c3;

The query was selected because it doesn't use the same level of aggregation as the MQT we just defined. This makes it harder for the optimizer - we want to see if the optimizer is smart enough to rewrite the query to use the MQT instead of pulling all the data over the network.

Without the MQT the query plan looks like Figure 12:

Figure 12. The query plan when MQTs are not used

Now lets add the MQT. Sure enough, the optimizer was smart enough to rewrite the query to use the summary instead of brute forcing the answer. The cost drops dramatically to only 25 timerons (Figure 13).
Figure 13. Query performance improves dramatically with use of MQTs

Another look at what happened with the MQT: the Visual Explain facility in the Control Center shows both the SQL as it was presented to the federator and the actual SQL that was executed.

The SQL query submitted references the remote tables on the federatees. The group by is performed on the column c3. This makes it more challenging to the optimizer since the MQT is aggregated on column c100.

Figure 14. Query that was submitted to the federator (before optimization)

Now, look at Figure 15. After optimization there is no join! A single table select from the MQT satisfies the query. The optimizer used the system meta-data to recognize that the table card_mqt needed simple aggregation instead of using the remote tables.

Figure 15. Query that was actually executed uses the MQT

Data mart elimination

The optimizer query rewrite working with the MQTs on federated servers are a very powerful combination. Rather than all the work of building a data mart, the MQT offers an alternative for many queries. This is not a substitute for a datamart when the queries use a combination of high volumes of data with functions that prevent use of the summary cache.
The MQTs should be populated and refreshed during off-hours to optimize performance and minimize the impact of this computation and communication traffic.

**Conclusion**

So that's it for your introduction to federation. In the next article we'll look at other ways to apply the federated technology, specifically:

- Federated inserts
- XML from federated joins, and
- XML and Websphere® MQ: Chocolate meets Peanut Butter.

**Appendix A**

```sql
-- This SQL creates a federated environment
-- for two different
-- Federatees

-- connect to the DB2 database
connect to sample;

drop wrapper "INFORMIX";

CREATE WRAPPER "INFORMIX" LIBRARY 'db2informix.dll';

-- create a server for a remote instance of Informix 9.3

CREATE SERVER "rcfliif"
    TYPE INFORMIX VERSION '9.3'
    WRAPPER "INFORMIX" OPTIONS ( NODE 'fliif',
        DBNAME 'stores_demo'
        -- , ADD CPU_RATIO '0.0000001'
        -- , ADD IO_RATIO '0.0000001'
        , ADD CPU_RATIO '1'
        , ADD IO_RATIO '1'
        , ADD COMM_RATE '1'
        -- no, still not pushing down join, ADD PUSHDOWN 'Y'
        , ADD DB2_MAXIMAL_PUSHDOWN 'Y'
        -- , ADD PUSHDOWN 'Y'
    );

-- create a server for a remote XPS Informix v8.3 instance

CREATE SERVER "rc_xps"
    TYPE INFORMIX VERSION '8.3'
    WRAPPER "INFORMIX" OPTIONS ( NODE 'flxps',
        DBNAME 'stores_demo'
        , ADD CPU_RATIO '1.0'
        , ADD IO_RATIO '1.0'
        , ADD COMM_RATE '2'
        , ADD PUSHDOWN 'Y'
    );

-- CREATE USER MAPPING FOR "LURIE"
CREATE USER MAPPING FOR "LURIE"
    SERVER "rcfliif" OPTIONS( REMOTE_AUTHID 'informix',
        REMOTE_PASSWORD 'useyourown');
CREATE USER MAPPING FOR "LURIE"
    SERVER "rc_xps" OPTIONS( REMOTE_AUTHID 'informix',
        REMOTE_PASSWORD 'useyourown2');

-- create nickname rc9_card1 for "rcfliif"."lurie"."card1"
create nickname rc9_card1 for "rcfliif"."lurie"."card1";
create nickname rc9_card2 for "rcfliif"."lurie"."card2";
```

The Federation: Database Interoperability (Part 1)
create nickname rc8_card1 for "rc_xps"."lurie"."card1";
create nickname rc8_card2 for "rc_xps"."lurie"."card2";

-- create a materialized query table or MQT
-- this will allow the optimizer to rewrite SQL to
-- the remote servers

drop table card_mqt;
create summary table card_mqt
as ( select a.c3, a.c10, sum(a.c100 ) as sum_c100,
     sum(a.c1000) as sum_c1000
from rc9_card1 a, rc9_card2 b, rc8_card1 c, rc8_card2 d
where a.superkey = b.superkey
and b.superkey = c.superkey
and c.superkey = d.superkey
group by a.c3, a.c10 )
data initially deferred refresh deferred;

refresh table card_mqt;
set current refresh age=any;

-- run various queries and capture explain plans

set current explain snapshot=yes;

values current timestamp;
select a.c3, a.c10, sum(a.c100 ) as sum_c100,
     sum(a.c1000) as sum_c1000
from rc9_card1 a, rc9_card2 b
where a.superkey = b.superkey
group by a.c3, a.c10;
values current timestamp;

select a.c3, a.c10, sum(a.c100 ) as sum_c100,
     sum(a.c1000) as sum_c1000
from rc9_card1 a, rc9_card2 b, rc8_card1 c,
     rc8_card2 d
where a.superkey = b.superkey
and b.superkey = c.superkey
and c.superkey = d.superkey
group by a.c3, a.c10;
values current timestamp;

select a.c3,  sum(a.c100 ) as sum_c100
from rc9_card1 a, rc9_card2 b, rc8_card1 c,
     rc8_card2 d
where a.superkey = b.superkey
and b.superkey = c.superkey
and c.superkey = d.superkey
group by a.c3;
values current timestamp;
set current explain snapshot=no;

Appendix B

-- (c) Copyright 2003 Martin Lurie and IBM Corp

drop table card1;
create table card1 ( superkey serial,
c3 int,
create table card2 ( superkey int,
c3 int,
c10 int,
c100 int,
c1000 int )
in dbspci;

-- Stored procedure to populate the
-- table drop procedure pop_card;

c10 int,
c100 int,
c1000 int )
in dbspci1;
drop table card2;
create table card2 ( superkey int,
c3 int,
c10 int,
c100 int,
c1000 int )
in dbspci1;

create procedure pop_card (tot_rows int)
define rows_in integer;
define c3cnt integer;
define c10cnt integer;
define c100cnt integer;
define c1000cnt integer;

let rows_in=0;
for c1000cnt = 1 to 1000 step 1
  for c100cnt = 1 to 100 step 1
    for c10cnt = 1 to 10 step 1
      for c3cnt = 1 to 3 step 1
        insert into card1 values ( 0,
c3cnt,
c10cnt,
c100cnt,
c1000cnt);
      let rows_in = rows_in + 1;
      if rows_in > tot_rows then
        exit for;
      end if;
    end for; -- c100cnt
    if rows_in > tot_rows then
      exit for;
    end if;
  end for; -- c100cnt
  if rows_in > tot_rows then
    exit for;
  end if;
end for; -- c10cnt
end for; -- c3cnt

end procedure;

execute procedure pop_card ( 50000 );

select count(*) from card1;