Fast and easy data movement using DB2's LOAD FROM CURSOR feature

Dirk Fechner January 08, 2009

Ease the process of DB2® for Linux®, UNIX®, and Windows® data movement using the DB2 LOAD utility's FROM CURSOR option. This article introduces the LOAD FROM CURSOR feature and provides usage samples for two interfaces, the Command Line Processor and the ADMIN_CMD stored procedure.

Introduction

Typical DB2 data movement tasks involve three steps:

1. Exporting the data from the source database into a temporary data exchange file in binary or text format
2. Moving the generated file between systems
3. Importing or loading the data from the file into the target database.

Generating the data exchange file using the EXPORT utility is often a lengthy process in the case of large amounts of data. Besides, different database codepages and operating systems have to be considered when moving data in and out of a database.

Such problems can be avoided by using the LOAD utility's FROM CURSOR option. When specifying the FROM CURSOR option, the LOAD utility directly references the result set of a SQL query as the source of a data load operation, thus bypassing the need to produce a temporary data exchange file. This way, a LOAD FROM CURSOR is a fast and easy possibility to move data between different tablespaces or different databases. LOAD FROM CURSOR operations can be executed on the command line as well as from within an application or a stored procedure by utilizing DB2's ADMIN_CMD stored procedure. This article introduces the LOAD FROM CURSOR feature and provides usage samples for both interfaces, the DB2 Command Line Processor (CLP) and ADMIN_CMD stored procedure.

Moving a table into another tablespace

First, see how to move a table from one tablespace into another tablespace. This may become necessary if the table was created in a tablespace with an inadequate page size or when a separate bufferpool should be used for accessing the table. In DB2 versions prior to 9.1, tables

were often moved between tablespaces because the maximum size of a tablespace was reached. However, with DB2 9.1 and higher versions, this should be no longer an issue as the tablespace size limits have been significantly increased (a prerequisite is the usage of large tablespaces in contrast to the regular tablespaces used before).

This sample scenarios starts with creating the DB2 SAMPLE database. This can be accomplished by calling the `db2sampl` command on the command line as Listing 1 indicates.

Listing 1. Creating the SAMPLE database

```
C:\>db2sampl

  Creating database "SAMPLE"...
  Connecting to database "SAMPLE"...
  Creating tables and data in schema "FECHNER"...

  'db2sampl' processing complete.
```

Besides other tables, the SAMPLE database contains a table named SALES. By default, that table was created in tablespace USERSPACE1. This can be verified by executing a query against the DB2 catalog views SYSCAT.TABLES and SYSCAT.TABLESPACES.

Listing 2. Determining the tablespace of table SALES

```
C:\>db2 "CONNECT TO SAMPLE"

   Database Connection Information

 Database server        = DB2/NT 9.5.2
 SQL authorization ID   = FECHNER
 Local database alias   = SAMPLE

C:\>db2 "SELECT TABLES.TABSCHEMA, TABLES.TABNAME, TBSPACES.TBSPACE FROM SYSCAT.TABLES AS
TABLES, SYSCAT.TABLESPACES AS TBSPACES WHERE TABLES.TBSPACEID = TBSPACES.TBSPACEID AND
TABNAME = 'SALES'"

TABSCHEMA
         TABNAME
                  TBSPACE

------------------------------------------------------------------------------------------
-------- ---------------------------------------------------------------------------------
---------------- --------------------------------------------------------------------------
-------------------------
FECHNER
         SALES
                  USERSPACE1


  1 record(s) selected.
```

In addition to tablespace USERSPACE1, there is a second tablespace IBMDB2SAMPLEREL that is intended for storing user data, too. In this sample scenario, IBMDB2SAMPLEREL serves as the destination tablespace for moving table SALES. By executing the DB2 command `LIST TABLESPACES`, you can see all the tablespaces of a database. Listing 3 shows how to do this.

Listing 3. Listing all tablespaces of the SAMPLE database

```
C:\>db2 "LIST TABLESPACES"
```

```
          Tablespaces for Current Database

Tablespace ID                        = 0
Name                                 = SYSCATSPACE
Type                                 = Database managed space
Contents                             = All permanent data. Regular table space.
State                                = 0x0000
  Detailed explanation:
    Normal

Tablespace ID                        = 1
Name                                 = TEMPSPACE1
Type                                 = System managed space
Contents                             = System Temporary data
State                                = 0x0000
  Detailed explanation:
    Normal

Tablespace ID                        = 2
Name                                 = USERSPACE1
Type                                 = Database managed space
Contents                             = All permanent data. Large table space.
State                                = 0x0000
  Detailed explanation:
    Normal

Tablespace ID                        = 3
Name                                 = IBMDB2SAMPLEREL
Type                                 = Database managed space
Contents                             = All permanent data. Large table space.
State                                = 0x0000
  Detailed explanation:
    Normal
```

Before copying the contents of table SALES to tablespace IBMDB2SAMPLEREL, you have to create an empty table in the destination tablespace that has the same structure as table SALES. Because it is not possible to have two tables with identical names in the same schema, the new table is temporarily created with the name SALES_TMP. Creating an empty table with the same structure as an existing table can be done by specifying the `LIKE` option of the `CREATE TABLE` command (Listing 4). The tablespace for the new table SALES_TMP is explicitly defined through the `IN` option.

Listing 4. Creating the destination table SALES_TMP for the data movement operation

```
C:\>db2 "CREATE TABLE FECHNER.SALES_TMP LIKE FECHNER.SALES IN IBMDB2SAMPLEREL"
DB20000I  The SQL command completed successfully.
```

Now the data movement operation can take place. With the `DECLARE CURSOR` command, a cursor is defined that reads all data of the source table SALES using a trivial `SELECT` statement. The name of the cursor, C1 in the sample, can be freely chosen. Then this cursor is referenced in a `LOAD` command for filling the destination table SALES_TMP. The `LOAD` command in the sample writes its messages in a log file load_sales_tmp.msg. The LOAD operation is performed with the option `NONRECOVERABLE`. That means the LOAD operation cannot be redone during the rollforward phase of a database recovery. Thus a database backup, or at least a tablespace backup, should be executed following the data movement operation. There are other options of the

`LOAD` command that avoid such a situation, but these options are outside the scope of this article. See the description of the `LOAD` command in the DB2 Information Center for further information (see Related topics).

Listing 5. Executing a LOAD FROM CURSOR operation to copy all rows in table SALES into table SALES_TMP

```
C:\>db2 "DECLARE C1 CURSOR FOR SELECT * FROM FECHNER.SALES"
DB20000I  The SQL command completed successfully.

C:\>db2 "LOAD FROM C1 OF CURSOR MESSAGES C:\load_sales_tmp.msg INSERT INTO
FECHNER.SALES_TMP NONRECOVERABLE"

Number of rows read         = 41
Number of rows skipped      = 0
Number of rows loaded       = 41
Number of rows rejected     = 0
Number of rows deleted      = 0
Number of rows committed    = 41
```

After copying all rows in table SALES successfully to table SALES_TMP, the source table SALES can be dropped (`DROP TABLE` statement). Then the destination table SALES_TMP is renamed to SALES (`RENAME TABLE` statement). When working with `RENAME TABLE`, one should know that only the table name can be changed but not the schema name for the table. Thus it is important to create table SALES_TMP in the correct schema right from the start.

Listing 6. Dropping source table SALES and renaming destination table SALES_TMP

```
C:\>db2 "DROP TABLE FECHNER.SALES"
DB20000I  The SQL command completed successfully.

C:\>db2 "RENAME TABLE FECHNER.SALES_TMP TO SALES"
DB20000I  The SQL command completed successfully.
```

Using the already introduced query on the DB2 catalog views, you can check that table SALES has been moved from its original tablespace USERSPACE1 in its new tablespace IBMDB2SAMPLEREL as seen in Listing 7.

Listing 7. Verifying the tablespace of the new SALES table

```
C:\>db2 "SELECT TABLES.TABSCHEMA, TABLES.TABNAME, TBSPACES.TBSPACE FROM SYSCAT.TABLES AS
TABLES, SYSCAT.TABLESPACES AS TBSPACES WHERE TABLES.TBSPACEID = TBSPACES.TBSPACEID AND
TABNAME = 'SALES'"

TABSCHEMA
         TABNAME
                  TBSPACE

------------------------------------------------------------------------------------------
-------- ----------------------------------------------------------------------------------
---------------- --------------------------------------------------------------------------
--------------------------
FECHNER
         SALES
                  IBMDB2SAMPLEREL
```

```
  1 record(s) selected.

C:\>db2 "TERMINATE"
DB20000I  The TERMINATE command completed successfully.
```

Creating a table copy in another database by using a nickname

As well as a table can be moved between tablespaces within a database, a table can be moved between two different databases. That means: Using `LOAD FROM CURSOR`, a table can be copied from one database into another database, too. This can be accomplished in two ways:

- Approach 1 - From within the target database, the source database is accessed using DB2's mechanisms for federated databases.
- Approach 2 - The remote access feature of the `LOAD FROM CURSOR` command is used.

Both approaches offer certain advantages which will be explored in this article.

Approach 1 uses a federated access to the source database to copy table contents. Approach 1 requires that the target database is configured as federated database. Therefore the parameter `FEDERATED` of the corresponding DB2 instance has to be set to `YES` (`UPDATE DBM CFG`). After changing the `DBM CFG` parameter `FEDERATED`, the DB2 instance has to be restarted (`db2stop`/`db2start`). In this sample scenario, source and target database run within the same DB2 instance. As a target database, you create an empty, second database named MYSAMPLE. For a test database with no special requirements, this can be accomplished using a `CREATE DATABASE` command without further options.

Listing 8. Activating federated database support in the DBM CFG and creating the empty target database MYSAMPLE

```
C:\>db2 "UPDATE DBM CFG USING FEDERATED YES"
DB20000I  The UPDATE DATABASE MANAGER CONFIGURATION command completed
successfully.
SQL1362W  One or more of the parameters submitted for immediate modification
were not changed dynamically. Client changes will not be effective until the
next time the application is started or the TERMINATE command has been issued.
Server changes will not be effective until the next DB2START command.

C:\>db2stop
2008-09-22 14.55.36     0   0   SQL1064N  DB2STOP processing was successful.
SQL1064N  DB2STOP processing was successful.

C:\>db2start
2008-09-22 14.55.53     0   0   SQL1063N  DB2START processing was successful.
SQL1063N  DB2START processing was successful.

C:\>db2 "CREATE DATABASE MYSAMPLE"
DB20000I  The CREATE DATABASE command completed successfully.
```

As before, an empty table is required in the target database MYSAMPLE that has the same structure like the table SALES in the source database SAMPLE. Thus, you should extract the DDL of table sales in the source database using the `db2look` utility.

Listing 9. Extracting DDL of source table SALES using the db2look utility

```
C:\>db2look -d sample -e -z fechner -t sales -o sales.ddl
-- USER is: FECHNER
-- Specified SCHEMA is: FECHNER
-- The db2look utility will consider only the specified tables
-- Creating DDL for table(s)

-- Schema name is ignored for the Federated Section
-- Output is sent to file: sales.ddl
-- Binding package automatically ...
-- Bind is successful
-- Binding package automatically ...
-- Bind is successful
```

The result of the `db2look` call is a file sales.ddl containing the `CREATE TABLE` statement for table SALES. If there were constraints and/or indexes defined on table SALES, the corresponding definitions would also be extracted and written to the file sales.ddl. Listing 10 shows these results.

Listing 10. Result file sales.ddl from db2look call

```
-- This CLP file was created using DB2LOOK Version 9.5
-- Timestamp: 23.09.2008 07:35:10
-- Database Name: SAMPLE
-- Database Manager Version: DB2/NT Version 9.5.2
-- Database Codepage: 1208
-- Database Collating Sequence is: IDENTITY


CONNECT TO SAMPLE;

------------------------------------------------
-- DDL Statements for table "FECHNER "."SALES"
------------------------------------------------


CREATE TABLE "FECHNER "."SALES"  (
    "SALES_DATE" DATE ,
    "SALES_PERSON" VARCHAR(15) ,
    "REGION" VARCHAR(15) ,
    "SALES" INTEGER )
   IN "IBMDB2SAMPLEREL" ;


COMMIT WORK;

CONNECT RESET;

TERMINATE;
```

Before executing the statements in file sales.ddl against the target database MYSAMPLE, open the file in a text editor and make two changes to the generated statements:

- At the beginning of the file, `db2look` generated a `CONNECT` statement to the source database SAMPLE. As you want to execute the following statements against the target database MYSAMPLE, you alter the `CONNECT` statement accordingly.
- As there is not a tablespace IBMDB2SAMPLREL for user data in the target database MYSAMPLE, replace the tablespace name in the `CREATE TABLE` statement with USERSPACE1.

Listing 11. Modifications in the result file sales.ddl for creation of the destination table

```
CONNECT TO SAMPLE; -> CONNECT TO MYSAMPLE;

IN "IBMDB2SAMPLEREL" ; -> IN "USERSPACE1";
```

After changing the file sales.ddl as described above, the script is executed by calling the DB2 CLP (command line processor).

Listing 12. Creating the destination table in database MYSAMPLE

```
C:\>db2 -tf sales.ddl

   Database Connection Information

 Database server        = DB2/NT 9.5.2
 SQL authorization ID   = FECHNER
 Local database alias   = MYSAMPLE


DB20000I  The SQL command completed successfully.

DB20000I  The SQL command completed successfully.

DB20000I  The SQL command completed successfully.

DB20000I  The TERMINATE command completed successfully.
```

Until now, only an empty copy of table SALES in the target database MYSAMPLE has been created. The next step in preparing your data movement operation is cataloging the source database SAMPLE as remote database. Obviously, this is not a mandatory prerequisite for this sample scenario because both databases, source and target, run on the same server within the same DB2 instance. But in a real environment, the following `CATALOG` commands would have to be executed for the DB2 instance hosting the target database to allow TCP/IP access to the source database.

Listing 13. Creating entries in the node and database directory for accessing database SAMPLE

```
C:\>db2 "CATALOG TCPIP NODE SRCNODE REMOTE localhost SERVER 50000"
DB20000I  The CATALOG TCPIP NODE command completed successfully.
DB21056W  Directory changes may not be effective until the directory cache is
refreshed.

C:\>db2 "CATALOG DATABASE SAMPLE AS SRCDB AT NODE SRCNODE AUTHENTICATION SERVER"
DB20000I  The CATALOG DATABASE command completed successfully.
DB21056W  Directory changes may not be effective until the directory cache is
refreshed.

C:\>db2 "TERMINATE"
DB20000I  The TERMINATE command completed successfully.
```

The last step to prepare the LOAD FROM CURSOR operation is to configure the federated access to table SALES in the source database SAMPLE. This is done within the target database MYSAMPLE by creating several, special objects required for federated access to another database:

- Wrapper - A wrapper allows access to external data sources. The external data source could be another DBMS (database management system) like Oracle or SQL Server, or just an Excel sheet. Depending on the data source that should be accessed, a suitable wrapper is required. These wrappers are contained in the separate IBM product WebSphere® Federation Server. If one would like to access just another database of the DB2 product family, DB2 LUW or DB2 z/OS, the DRDA wrapper is needed. This wrapper is already contained in DB2 LUW, that is, WebSphere Federation Server is not necessary in that case. The DRDA wrapper is created by executing the following, trivial command in the target database: `CREATE WRAPPER DRDA`.
- Server - The term server is a little bit confusing in this case because what is really meant is the source database that takes the role of a (data) server. To make the source database known in the target database, a server object is created that specifies the type of the data source (DB2/UDB VERSION 9.5), the wrapper to use (DRDA), and a user/password combination to access the source database. The name of the source database itself is provided using the option `DBNAME`. Username and password have to be specified in quotation marks. To avoid that the quotation marks are removed by the command line interpreter, they are masked with a backslash (\). The name of the server object can be freely chosen. In this sample scenario, the name SRCSRV is used.
- User Mapping - A user mapping has to be defined for each user who wants to access the remote database using the previously defined server object. The user mapping defines how the authorization ID at the local database (MYSAMPLE) is mapped to an authorization ID at the remote database (SAMPLE). In this sample scenario, local and remote user are identical, nevertheless a user mapping has to be defined.
- Nickname - The nickname is a local alias for the remote table in the source database. Specifying the nickname, the remote table can be queried in SQL statements on the target database like any other local table.

Listing 14. Creating the database objects required for federated access

```
C:\>db2 "CONNECT TO MYSAMPLE"

   Database Connection Information

 Database server        = DB2/NT 9.5.2
 SQL authorization ID   = FECHNER
 Local database alias   = MYSAMPLE

C:\>db2 "CREATE WRAPPER DRDA"
DB20000I  The SQL command completed successfully.

C:\>db2 "CREATE SERVER SRCSRV TYPE DB2/UDB VERSION 9.5 WRAPPER DRDA AUTHORIZATION
\"fechner\" PASSWORD \"password\" OPTIONS (DBNAME 'SRCDB')"
DB20000I  The SQL command completed successfully.

C:\>db2 "CREATE USER MAPPING FOR fechner SERVER SRCSRV OPTIONS (REMOTE_AUTHID 'fechner',
REMOTE_PASSWORD 'password')"
DB20000I  The SQL command completed successfully.

C:\>db2 "CREATE NICKNAME FECHNER.SRCTAB FOR SRCSRV.FECHNER.SALES"
DB20000I  The SQL command completed successfully.
```

Note: The steps described here for setting up the federated access are completely independent from the LOAD FROM CURSOR functionality. That means these steps can generally be used to create a nickname for a table in a remote database.

Now that federated access to the table in the source database is configured, the LOAD FROM CURSOR operation works exactly the same as already shown. First a cursor is defined that reads all rows in the remote table using the nickname created above. Then that cursor is referenced in the `LOAD` command.

Listing 15. Executing the remote LOAD FROM CURSOR operation using the nickname

```
C:\>db2 "DECLARE C1 CURSOR FOR SELECT * FROM FECHNER.SRCTAB"
DB20000I  The SQL command completed successfully.

C:\>db2 "LOAD FROM C1 OF CURSOR MESSAGES C:\load_sales.msg INSERT INTO FECHNER.SALES
NONRECOVERABLE"

Number of rows read         = 41
Number of rows skipped      = 0
Number of rows loaded       = 41
Number of rows rejected     = 0
Number of rows deleted      = 0
Number of rows committed    = 41

C:\>db2 "TERMINATE"
DB20000I  The TERMINATE command completed successfully.
```

As already mentioned, the LOAD FROM CURSOR operation combined with the federated access requires more effort in terms of configuration than the approach that is presented next. But the main advantage of the federated approach is the possibility to load data from data sources other than DB2. By using the federated approach, you can access data sources like Oracle, SQL Server, and many other relational and non-relational data sources and copy contents by creating a nickname and executing a LOAD FROM CURSOR operation referencing that nickname. The wrappers required to access other data sources than DB2 are shipped with the WebSphere Federation Server product.

An easier way to create a table copy in another database

Now that you know the remote LOAD FROM CURSOR approach based on usage of nicknames, you will evaluate the alternate approach that comes with less effort. To do so, you start with deleting all the rows that you have just imported in table SALES in the target database MYSAMPLE as Listing 16 shows.

Listing 16. Deleting all rows in the destination table for repetition of the LOAD FROM CURSOR operation

```
C:\>db2 "CONNECT TO MYSAMPLE"

   Database Connection Information

 Database server        = DB2/NT 9.5.2
 SQL authorization ID   = FECHNER
 Local database alias   = MYSAMPLE

C:\>db2 "DELETE FROM FECHNER.SALES"
DB20000I  The SQL command completed successfully.
```

For the alternate approach, it is not necessary to configure a federated access to the remote database. Instead, the remote database is just specified in the `DECLARE CURSOR` statement using the `DATABASE` option. For this to work, the remote database has to be cataloged in the system database directory of the local DB2 instance. The corresponding `CATALOG` commands were already presented above. In addition, the user name and password for the remote access are specified during definition of the cursor. The `LOAD` command itself remains unchanged.

Listing 17. Executing the remote LOAD FROM CURSOR operation without using a nickname

```
C:\>db2 "DECLARE C1 CURSOR DATABASE SRCDB USER fechner USING password FOR SELECT * FROM
FECHNER.SALES"
DB20000I  The SQL command completed successfully.

C:\>db2 "LOAD FROM C1 OF CURSOR MESSAGES C:\load_sales_2.msg INSERT INTO FECHNER.SALES
NONRECOVERABLE"

Number of rows read         = 41
Number of rows skipped      = 0
Number of rows loaded       = 41
Number of rows rejected     = 0
Number of rows deleted      = 0
Number of rows committed    = 41

C:\>db2 "TERMINATE"
DB20000I  The TERMINATE command completed successfully.
```

The possibility to perform a remote LOAD FROM CURSOR operation this way exists since DB2 9.1, the approach based on a federated access did already work in version 8. The new approach comes with two advantages -- ease of use and performance. It is obvious that the new approach is very easy to use. The performance benefit compared to the federated approach is related to the fact that less data transfer layers are involved. However, the advantage of the federated approach should not be forgotten, that is, the possibility to access other data sources than DB2.

Differences between CLP and ADMIN_CMD concerning LOAD FROM CURSOR

Many administrative commands can also be embedded in application code by executing the administrative commands through the special stored procedure ADMIN_CMD. This is true for a LOAD FROM CURSOR operation, too. Usage of the stored procedure ADMIN_CMD is independent from the location of the application code, that is, it does not matter if you talk about client-side code (for example, a Java application) or server-side code (for example, a SQL/PL stored procedure). The following sample demonstrates usage of the ADMIN_CMD stored procedure within a custom SQL/PL stored procedure. The file create_load_routine.sql contains the SQL/PL source code for our sample stored procedure called REMOTE_LOAD_FROM_CURSOR.

Listing 18. Contents of file create_load_routine.sql containing the sample stored procedure

```
CREATE PROCEDURE FECHNER.REMOTE_LOAD_FROM_CURSOR ()
    SPECIFIC REMOTE_LOAD_FROM_CURSOR
    LANGUAGE SQL
BEGIN

    DELETE FROM FECHNER.SALES;--

    CALL SYSPROC.ADMIN_CMD ('LOAD FROM (DATABASE SRCDB SELECT * FROM FECHNER.SALES) OF
CURSOR INSERT INTO FECHNER.SALES NONRECOVERABLE');--
END;
```

The first statement within the stored procedure is a `DELETE` to remove the existing rows in the local destination table SALES. Next, the remote LOAD FROM CURSOR operation is executed by calling ADMIN_CMD with a suitable `LOAD` command. The following differences exist compared to a LOAD FROM CURSOR operation that is executed on the command line:

- Definition of the required cursor does not occur separately by executing `DECLARE CURSOR`. Instead the cursor definition is implicitly done by providing the corresponding `SELECT` statement within the `LOAD` command. This syntax is only valid when the LOAD FROM CURSOR operation is embedded in an ADMIN_CMD call, but it is not valid on the command line.
- The remote database is defined via the `DATABASE` option within the `LOAD` command. The specification of a user/password combination for the remote access is not possible. Observe the implications of this restriction when testing your stored procedure.

But first you should create your stored procedure in the destination database MYSAMPLE.

Listing 19. Creating the sample stored procedure

```
C:\>db2 "CONNECT TO MYSAMPLE"

   Database Connection Information

 Database server        = DB2/NT 9.5.2
 SQL authorization ID   = FECHNER
 Local database alias   = MYSAMPLE

C:\>db2 -tf create_load_routine.sql
DB20000I  The SQL command completed successfully.
```

A first test call fails returning the message SQL30082N Security processing failed with reason "3" ("PASSWORD MISSING"). SQLSTATE=08001.

Listing 20. First test of the sample stored procedure that fails

```
C:\>db2 "CALL FECHNER.REMOTE_LOAD_FROM_CURSOR"
SQL30082N  Security processing failed with reason "3" ("PASSWORD MISSING").
SQLSTATE=08001

C:\>db2 "TERMINATE"
DB20000I  The TERMINATE command completed successfully.
```

The cause of the error message is the way the connection to the database was established: `db2 "CONNECT TO MYSAMPLE"`. When executing the `CONNECT` statement, no user and password were given so that the connection was established with the username that was used to log in to the operating system. In that case, DB2 does not know about the password of the user connected. Now when performing the LOAD FROM CURSOR operation within the stored procedure, DB2 tries to establish the remote connection to database SAMPLE with the authorization ID of the local user. However, as DB2 does not know about the corresponding password because of the implicit `CONNECT`, remote access fails. Thus the reasons for the error are the following:

- Because a LOAD FROM CURSOR operation executed via ADMIN_CMD does not allow the definition of a user for remote access, the LOAD operation automatically tries to establish a connection to the remote database with the authorization ID belonging to the local database connection.
- If the locally connected user has done an implicit `CONNECT` without specifying a password, her/his password is unknown to DB2 and therefore not available when trying to establish the connection to the remote database.

Another implication of the described behavior for a remote LOAD FROM CURSOR via ADMIN_CMD - that is, the restriction that no explicit user can be specified for remote access - is that the user currently connected to the local database must have access to the remote database using the same authorization ID. This restriction does not apply to the approach using a federated access to the remote database as in that case there is an additional layer of abstraction in form of the user mappings that have to be defined.

Having determined the cause of the error, reconnect to the local database, this time explicitly specifying a username and a password. Afterwards a second call to your stored procedure should work fine (Return Status = 0) as Listing 21 indicates.

Listing 21. Second test of the sample stored procedure that succeeds

```
C:\>db2 "CONNECT TO MYSAMPLE USER fechner"
Enter current password for fechner:

   Database Connection Information

 Database server        = DB2/NT 9.5.2
 SQL authorization ID   = FECHNER
 Local database alias   = MYSAMPLE

C:\>db2 "CALL FECHNER.REMOTE_LOAD_FROM_CURSOR"

  Return Status = 0

C:\>db2 "TERMINATE"
DB20000I  The TERMINATE command completed successfully.
```

Summary

With the help of sample scenarios, the article demonstrates how DB2's LOAD FROM CURSOR feature can be used to copy data fast and simple within a database and also between different

databases. The article also explains the specialties when LOAD FROM CURSOR operations are executed within application code using the ADMIN_CMD stored procedure. Additionally, you learned how to configure a federated access to another DB2 database so that tables/views in a remote database can be transparently read and written as if they were local tables or views.

Related topics

- Read Moving data using the CURSOR file type to get more in-depth information about the topics covered in this article:
- Visit the DB2 9.5 Information Center for Linux, UNIX, and Windows to get the complete DB2 9.5 LUW documentation online in HTML format.
- The DB2 9 for Linux UNIX and Windows Support Site lets you search for APARs, download fixpacks, get DB2 LUW documentation in PDF format, and so on.
- Check out the Best practices for DB2 for Linux, UNIX, and Windows papers. These papers are designed to provide practical guidance for the most common DB2 9 product configurations. By applying these recommendations, you may improve the value of your DB2 data servers and align yourself with IBM's technical direction for DB2. These papers are authored by leading experts in IBM's development and consulting teams, and have been extensively tested.
- Download IBM product evaluation versions and get your hands on application development tools and middleware products from DB2®, Lotus®, Rational®, Tivoli®, and WebSphere®.