



153

DB2

July 2005

In this issue

- [3 DB2 LUW – how to alter the columns of a table](#)
 - [10 Distributed Relational Database Architecture](#)
 - [19 DB2 LUW – using maintained tables in a federated environment](#)
 - [29 Managing DB2 for z/OS through WAP and Web environments](#)
 - [49 DB2 news](#)
-

© Xephon Inc 2005

update

DB2 Update

Published by

Xephon Inc
PO Box 550547
Dallas, Texas 75355
USA

Phone: 214-340-5690

Fax: 214-341-7081

Editor

Trevor Eddolls

E-mail: trevore@xephon.com

Publisher

Colin Smith

E-mail: info@xephon.com

Subscriptions and back-issues

A year's subscription to *DB2 Update*, comprising twelve monthly issues, costs \$380.00 in the USA and Canada; £255.00 in the UK; £261.00 in Europe; £267.00 in Australasia and Japan; and £265.50 elsewhere. In all cases the price includes postage. Individual issues, starting with the January 2000 issue, are available separately to subscribers for \$33.75 (£22.50) each including postage.

***DB2 Update* on-line**

Code from *DB2 Update*, and complete issues in Acrobat PDF format, can be downloaded from our Web site at <http://www.xephon.com/db2>; you will need to supply a word from the printed issue.

Disclaimer

Readers are cautioned that, although the information in this journal is presented in good faith, neither Xephon nor the organizations or individuals that supplied information in this journal give any warranty or make any representations as to the accuracy of the material it contains. Neither Xephon nor the contributing organizations or individuals accept any liability of any kind howsoever arising out of the use of such material. Readers should satisfy themselves as to the correctness and relevance to their circumstances of all advice, information, code, JCL, and other contents of this journal before making any use of it.

Contributions

When Xephon is given copyright, articles published in *DB2 Update* are paid for at the rate of \$160 (£100 outside North America) per 1000 words and \$80 (£50) per 100 lines of code for the first 200 lines of original material. The remaining code is paid for at the rate of \$32 (£20) per 100 lines. To find out more about contributing an article, without any obligation, please download a copy of our *Notes for Contributors* from www.xephon.com/nfc.

© Xephon Inc 2005. All rights reserved. None of the text in this publication may be reproduced, stored in a retrieval system, or transmitted in any form or by any means, without the prior permission of the copyright owner. Subscribers are free to copy any code reproduced in this publication for use in their own installations, but may not sell such code or incorporate it in any commercial product. No part of this publication may be used for any form of advertising, sales promotion, or publicity without the written permission of the publisher.

Printed in England.

DB2 LUW – how to alter the columns of a table

This article looks at changing a column in a table; changing in this context can mean altering its attributes or dropping it. It has long been the bane of the DBA's life to be requested to change the attributes of a column, rename it, or drop it. With DB2 UDB V8.2 this process has been made a lot simpler. What you had to do in the past was get the DDL for the table, unload it, make the changes to the DDL, drop and re-create the table (making sure all the views etc defined on the original table were re-created), edit the unload file, and then load the data back into the newly-defined table. This could be 'automated' up to a point by having a set of scripts available, but it was still best done as a manual (albeit infrequent) process. Now we can perform all of these tasks using the DB2 Control Center.

Let's look at a couple of examples (all of which were performed on a Windows 2000 Professional system running DB2 V8 FP8 and using the sysadm userid db2admin and the SAMPLE database).

The first example will involve removing a column from the middle of a table and the second example will involve changing the attributes of a column in a table.

So let's start with removing a column from the middle of a table. Create a table called hmtab with three integer columns called c1, c2, and c3:

```
>db2 connect to sample user db2admin using xxxxxxxx
```

```
>db2 create table hmtab (c1 int, c2 int, c3 int)
```

Populate the table:

```
>db2 insert into hmtab values(1,1,1)
```

```
>db2 insert into hmtab values(2,2,2)
```

Create the following: a view on just columns c1 and c3; a view based on all the columns; and two Materialized Query Tables (MQTs) based on the sum of column c2.

To create the view called vc1c3 based on columns c1 and c3 and the view vc1c2c3 based on the columns c1, c2, and c3 issue:

```
>db2 create view vc1c3 as select c1,c3 from hmtab
>db2 create view vc1c2c3 as select c1,c2,c3 from hmtab
```

We want to create the two MQTs. The first one (called shmtabd) is defined as refresh deferred, and the other one (called shmtabs) is defined as refresh immediate:

```
>db2 create table shmtabd as (select sum(c2) as tot from hmtab) data
initially deferred refresh deferred
```

```
>db2 refresh table shmtabd
```

```
>db2 select * from shmtabd
```

TOT
3

```
>db2 create table shmtabs as (select c2,count(*) as tot from hmtab group
by c2) data initially deferred refresh immediate maintained by system
```

```
>db2 set integrity for shmtabs immediate checked
```

```
>db2 select * from shmtabs
```

C2	TOT
1	1
2	1

2 record(s) selected.

Now we have finished setting up everything, let's drop column c2. Don't forget that the userid performing the alter needs the DBADM and LOAD authorities on the table (because we are using the db2admin userid we are OK). We drop the column using the Control Center by right-clicking on table name hmtab and then selecting **Alter....** You will get the screen shown in Figure 1 – highlight the line containing c2 and then press the **Remove** button.

Click on **Close** and then **OK**. You will see Figure 2.

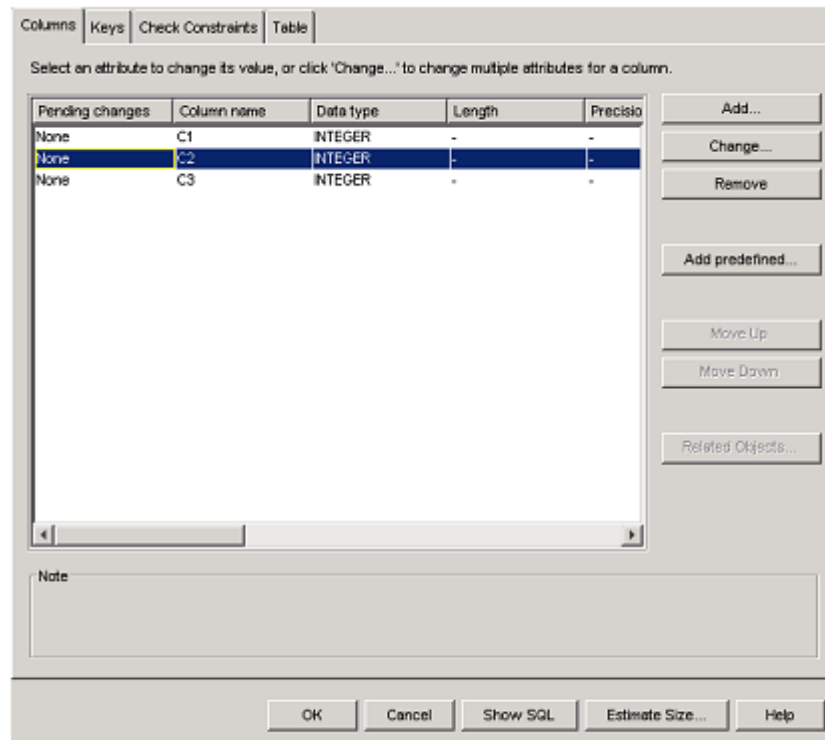


Figure 1: Changing an attribute

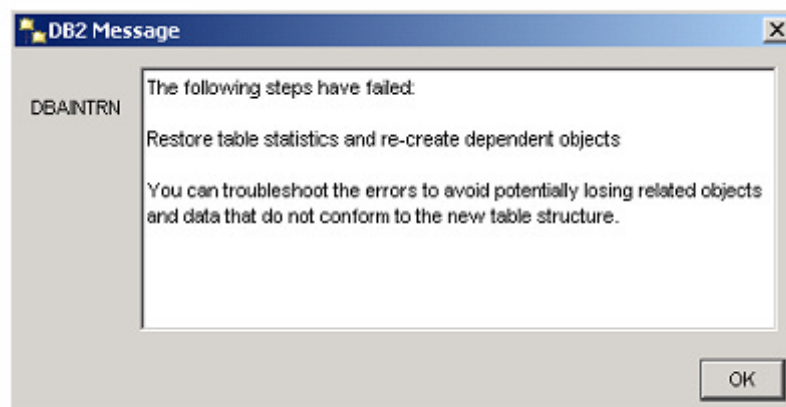


Figure 2: Error message

What has failed? Well, the view vc1c3 is re-created, but the creation of view vc1c2c3 failed because column c2 no longer

exists. The creation of the MQTs shmtabd and shmtabs also failed because they are based on column c2, which now no

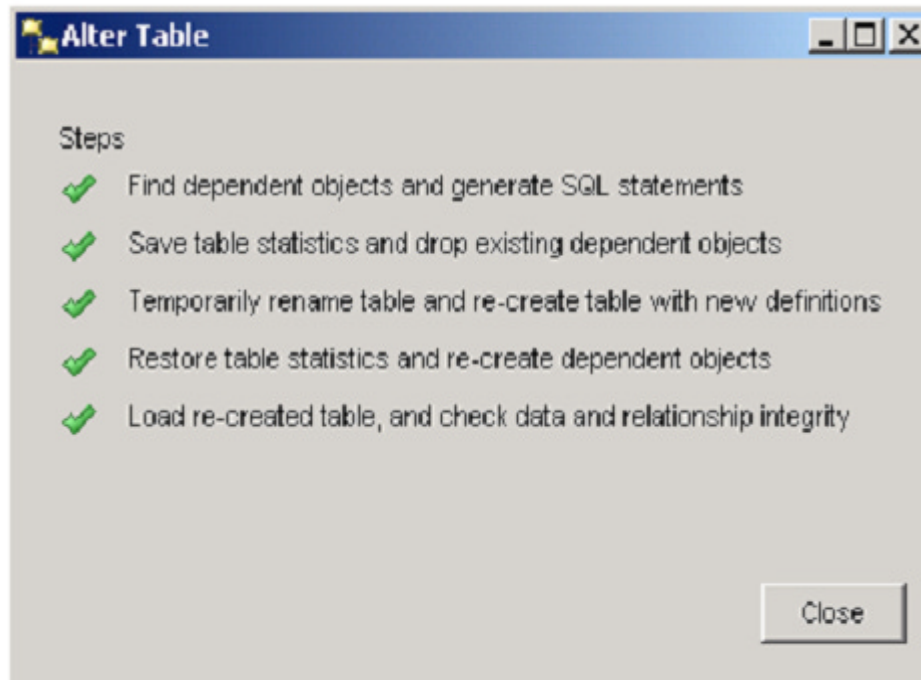


Figure 3: Successful conclusion

longer exists. You have the option to test each of the failed creations to determine why they failed, which is a very useful option.

If everything works out OK, then you would see something like Figure 3.

If you get the following error when trying to alter the table it means that the userid issuing the **alter** command did not have the DBADM and LOAD authorities:

```
[IBM][CLI Driver][DB2/NT] SQL0443N  Routine "SYSPROC.ALTOBJ" (specific  
name "ALTOBJ") has returned an error SQLSTATE with diagnostic text  
"DBA7904, DBAD".  SQLSTATE=38553
```

All the Control Center does is invoke a stored procedure. This is shown below:

```
CALL SYSPROC.ALTOBJ ( 'APPLY_CONTINUE_ON_ERROR', 'CREATE TABLE  
DB2ADMIN.HMTAB ( C1 INTEGER , C3 INTEGER ) IN USERSPACE1 ', -1, ? );
```

Can I run this as a command script? Yes you can. When I ran it I got:

```
SQL0443N  Routine "ALTOBJ" (specific name "") has returned an error  
SQLSTATE with diagnostic text "SQL0206  Reason code or token: C2".  
SQLSTATE=38553
```

If you select from the hmtab table you will see just columns c1 and c3 and the view vc1c3 exists, whereas vc1c2c3 doesn't. It seems that the command works and the error occurs because we are trying to create the objects like view v1c2c3, which cannot be created because column c2 has been dropped.

The Information Center tells us that we can use the ALTOBJ stored procedure to do the following:

- Rename a column.
- Increase or decrease the size of a column.
- Alter a column type and transform existing data using DB2 scalar functions.
- Change the precision or the scale of decimal values.
- Change the default value of a column.
- Change the nullability attribute of a column to nullable.
- Drop a column.

The stored procedure can be run with the following options: GENERATE, VALIDATE, APPLY_CONTINUE_ON_ERROR, APPLY_STOP_ON_ERROR, UNDO, and FINISH.

If you run the stored procedure with the GENERATE option, then you see the following (the file alter02.txt just contains the CALL SYSPROC lines shown):

```
>type alter02.txt  
CALL SYSPROC.ALTOBJ  
( 'GENERATE', 'CREATE TABLE DB2ADMIN.HMTAB ( C1 INTEGER , C3 INTEGER )  
IN USERSPACE1 ', -1, ? );
```



```
>db2 -tvf alter02.txt
```

```
CALL SYSPROC.ALTOBJ ( 'GENERATE', 'CREATE TABLE DB2ADMIN.HMTAB ( C1  
INTEGER , C3 INTEGER ) IN USERSPACE1 ', -1, ? )
```

```
Value of output parameters
```

```
-----
```

```
Parameter Name : ALTER_ID
```

```
Parameter Value : 10
```

```
Parameter Name : MSG
```

```
Parameter Value : SELECT OBJ_TYPE, OBJ_SCHEMA, OBJ_NAME,  
SQL_OPERATION, SQL_STMT, EXEC_SEQ FROM SYSTOOLS.ALTOBJ_INFO_V WHERE  
ALTER_ID=10 ORDER BY EXEC_MODE, EXEC_SEQ
```

```
Return Status = 0
```

And we could see the statements generated using the query:

```
>db2 select * from systools.altobj_info_v where alter_id = 10
```

Thirty-two rows are created. This isn't the easiest format to read, but it will tell you that all the steps in the stored procedure executed. The best way to look at this view is by using the Control Center. The main columns of interest are the SQL_OPERATION and SQL_STMT columns.

The second example involves changing the column attributes from char to integer of a table called hmtab2. Our table will have three columns defined as integer, character, and integer, and we want to change this to be three integer columns. Clearly, the character column could contain a name, which cannot be converted to an integer, as shown below:

```
>db2 connect to sample user db2admin using xxxxxxxx
```

```
>db2 create table hmtab2 (c1 int, c2 char(5), c3 int)
```

```
>db2 insert into hmtab2 values(1,'Helen',1)
```

Using the Control Center to change the attributes of column c2 from char to integer as in example 1, everything works up to the final step where we try to load into the re-created table. This step fails for obvious reasons (you cannot load a character value into a numeric field). The error panel is shown in Figure 4. The table has been created with the three integer columns

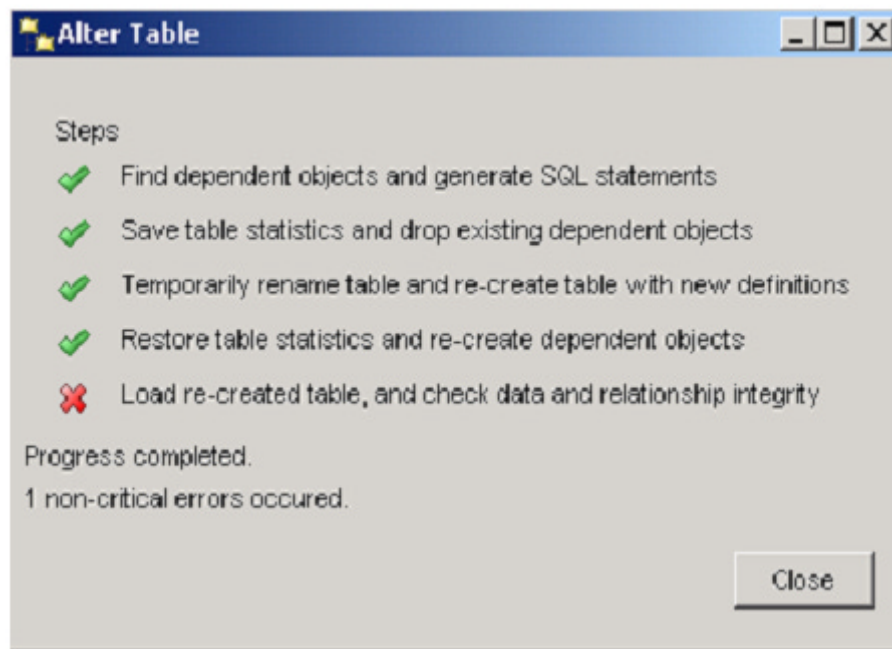


Figure 4: Error message

– it's just that it is empty. All is not lost though – a copy of the original table still exists and is called something like <userid>.T<date>_<hhmmss>. This contains the original table entries.

The stored procedure call is:

```
CALL SYSPROC.ALTOBJ ( 'APPLY_CONTINUE_ON_ERROR', 'CREATE TABLE
DB2ADMIN.HMTAB2 ( C1 INTEGER , C2 INTEGER , C3 INTEGER ) IN USERSPACE1
', -1, ? )
```

What would have happened if our character column had contained only numeric values? Suppose our insert statement had looked like this:

```
>db2 insert into hmtab2 values(1,'2',1)
```

If we now try to convert the table to have three integer values, then the conversion is successful.

I hope I have shown how easy it now is to change the column

attributes of a table, but, as ever on a production system, it's worth testing it first!

C Leonard
Freelance Consultant (UK)

© Xephon 2005

Distributed Relational Database Architecture

In this article I introduce the basic concepts of distributed databases and then describe how IBM developed DRDA to handle distributed databases.

A distributed database is a database that is not all stored at a single physical location, but is dispersed over a network of interconnected computers at different locations. The database can be viewed as a virtual object that physically consists of components located in different locations or on different computers in the same location. As far as a user is concerned, the database appears as a whole. A user can access any data in the network as if it is all stored on his local system – in the sense that he can access any database with a common access method. He does not have to use any special method according to the target database.

A distributed database has one or more of the following characteristics:

- Transparency of location.

Users and applications do not need to know where the data is stored. This should result in being able to move the database without rewriting the application.

- Scalability.

The maximum number of transactions and the size of the database are theoretically unlimited when you use a distributed database because you can extend your

distributed database without modifying existing databases and applications by adding more processing power.

- Growth can be phased.

In essence it is possible to add more hardware to extend a database at each location independently of other locations.

- Application portability.

It should be possible to develop an application at one location and migrate it to another without any modification.

- System independent.

Application programs and users should be independent of the operating system and protocols.

In 1988 IBM added the distributed database functionality to its Systems Application Architecture. To put this concept into practice the Distributed Data Facility was incorporated into DB2, but at first this only provided connectivity for MVS/ESA and DB2 on mainframe platforms. In 1990 IBM then designed and published Distributed Relational Database Architecture (DRDA). This defined rules and protocols that could be used to develop distributed databases across various networks. Most importantly it was designed to be platform independent. Throughout the '90s DRDA was adopted, and by 1998 the Open Group adopted it as an industry standard.

DRDA is an open vendor-independent architecture for providing connectivity between a client and database servers. By using DRDA it is possible for an application program to access various databases that support DRDA using SQL. One of the base concepts of DRDA is the Unit Of Work (UOW). A Unit of Work is a single logical transaction. It will normally consist of a sequence of SQL statements that are either all successfully performed, or the entire sequence of SQL statements is considered unsuccessful. Either will be a Unit of Work.

The second main concept is the two-phase commit protocol.

Two-phase commit is used by the Unit of Work and enables it to exist across multiple database management systems.

There are three well-known variants of two-phase commit protocols – presumed nothing, presumed abort, and presumed commit. You also need to understand that certain of these common protocols can only be used with certain network protocols. This is highlighted below:

- Presumed nothing – SNA supported – DRDA (TCP/IP) not supported.
- Presumed abort – SNA supported – DRDA (TCP/IP) supported.
- Presumed commit – SNA not supported – DRDA (TCP/IP) not supported.

So what is DRDA? Basically it is a standard for database interoperability protocol. It is not the only common interface for database access, but DRDA has proven to be a superior architecture in its performance and abundant functions. DRDA includes two-phase commit protocol, TCP/IP support, and support for stored procedures. Some of the major DBMS software products have adopted the DRDA architecture in their products. DRDA provides a common protocol, so an application program that uses DRDA can access any DRDA-supported databases.

The major features of DRDA are:

- DRDA is a database interoperability protocol using SQL as the standardized API. Both static and dynamic SQL are available.
- Remote bind support.
- Automatic data transformation.
- Unit of Work support.
- Stored procedure support.
- Superior performance, scalability, and availability.

- Support for data encryption.

The next concept to understand is that of function types. DRDA provides the following function types:

- Application Requester (AR) – functions support SQL and program preparation services from applications.
- Application Server (AS) – functions support requests that application requesters have sent and routes requests to database servers by connecting as an application requester.
- Database Server (DS) – this is to support requests from application servers.

DRDA also provides a number of protocols. These are:

- Application Support Protocol – provides connection between application requesters (AR) and application servers (AS).
- Database Support Protocol – provides connections between application servers (AS) and database servers (DS).

DRDA uses different architectures.

- Distributed Data Management (DDM) – the DDM architecture provides the command and reply structure used by the distributed databases.
- Formatted Data: Object Content Architecture (FD:OCA) – the FD:OCA provides the data definition architectural base for DRDA.
- Character Data Representation Architecture (CDRA) – CDRA provides the consistency of character data across the multiple platforms.

Communication protocols SNA and TCP/IP must be available. DRDA requires the ability to identify and authenticate the end user associated with the DRDA requests. Some network protocols, such as LU6.2, provide the ability to pass the

information for identification and authentication. However, when the network protocol does not provide this function, DRDA provides the security mechanisms instead, as follows:

- User ID and password.
- User ID, password, and new password.
- User ID only.
- User ID and password substitute.
- User ID and encrypted password.
- Encrypted user ID, encrypted password.
- Encrypted user ID, encrypted password, and encrypted new password.
- Kerberos support.

IBM DB2 uses the Distributed Data Facility (DDF) to provide the connectivity to and from other databases over the network. DB2 for z/OS and OS/390 DDF supports two network protocols (SNA and TCP/IP), as well as two database communication protocols (DB2 Private Protocol and DRDA). DDF implements a full DRDA application server and application requester.

DDF was first delivered by IBM in DB2 for MVS V2R2. At that time, the only supported protocol was the DB2 Private Protocol over SNA. Since V2R3, both the DB2 Private Protocol and DRDA protocol have been available. IBM recommends that all new applications be developed using DRDA, but does still support the DB2 Private Protocol.

DDF is DB2's transaction manager for distributed database connections. With DRDA, connections can come from anywhere on the SNA or TCP/IP network that DB2 is operating with. For this reason DDF has developed very mature thread management strategies to be able to handle thousands of connections from anywhere.

DDF runs as an additional address space in the DB2 subsystem. The address space name is xxxxDIST, where xxxx

is the DB2 subsystem name. DDF is an efficient connection handler that will use SRBs instead of TCBs, thus reducing CPU overhead.

When DDF is using TCP/IP as a network protocol it will execute TCP/IP services using the Unix System Services (USS) asynchronous I/O callable Assembler interface. However, whether the network protocol is SNA or TCP/IP, it is a DDF requirement that it has to open its VTAM ACB when it starts. DRDA is the only protocol that supports TCP/IP. You cannot use the DB2 Private Protocol when using TCP/IP as a network communication protocol.

IBM recommends using TCP/IP as the network protocol for your applications. TCP/IP does not have the same level of security implementation as SNA does, and there are some limitations when using TCP/IP as the network protocol for DRDA. However, as TCP/IP continues to overtake the traditional SNA networks, you should develop using it as the underlying protocol.

DDF, as mentioned previously, should be developed using DRDA. However, many sites still use the DB2 Private Protocol, so I will briefly mention it here. DB2 for z/OS and OS/390 can use this proprietary protocol. The DB2 Private Protocol can be used only to provide connectivity between DB2 for z/OS and OS/390 subsystems. It uses the VTAM Advanced Program-to-Program Communication (APPC) function for internal communications.

Each DB2 subsystem will execute as a VTAM LU and has a LUNAME that is unique in the network associated with it. Each DB2 subsystem also has a unique location name. In the case of a DB2 data sharing group, the location name is unique for the data sharing group – enabling the different members of the DB2 data sharing group to have the same location name but use a different VTAM LU name. In a distributed data environment, every table can be accessed using a unique name represented by a three-part name (location-name.table-owner.table-name).

There are many differences between DRDA and the DB2 Private Protocol. DRDA supports:

- Remote bind.
- The use of static SQL.
- Connectivity to DB2 and non-DB2, IBM and non-IBM databases.

With DB2 V2R3, the DRDA SQL CONNECT was introduced. This statement can be used to connect to a remote database instead of using a three-part name. Using the SQL CONNECT statement requires that the application be aware of where the data resides, because it first has to connect to that remote DBMS before issuing the SQL statement. For this reason, this is also called application-directed remote database access.

It is possible, when using DB2 Private Protocol, to use an alias to mask the actual location of the data from the application, making the location of the data transparent to the application. This is called system-directed remote database access.

Until DB2 for OS/390 V6, DRDA protocol did not support the use of aliases. Since DB2 V6, you can use aliases and three-part names in combination with the DRDA protocol.

DRDA is defined by using different levels, so that vendors can implement their applications step-by-step, depending on the level of DRDA.

DRDA level 1 supports Remote Unit of Work. DRDA level 2 supports Distributed Unit of Work, two-phase commit, and scrollable cursors. In DRDA level 3, in addition to the previous features, you can use TCP/IP communication manager as support for DRDA RUW or DUW. DCE is supported for the security mechanism. Stored procedures with result sets are supported.

In DRDA level 4, the following additional features are supported:

- Database-directed access.
- Support for DESCRIBE INPUT. This allows an application

<i>Function</i>	<i>DRDA</i>	<i>Private Protocol</i>
Network protocol	SNA, TCP/IP	SNA only
Accessible database	Any RDBMS implementing DRDA AS functions.	Only DB2 for z/OS and OS/390 subsystems.
Connection	Connect statement, 3-part name, alias	3-part name, alias.
CDB	Inbound name translation is not supported (when TCP/IP is used)	Support for inbound translation.
SQL	DML, DCL, and DDL are supported	Only DML such as INSERT, UPDATE, DELETE, SELECT, OPEN, and FETCH. In addition, certain clauses are not supported.
Stored procedures	Supported	Not supported.
Data type	No limitation	Large object (LOB) and user-defined data type (UDT) are not supported.
Remote bind	Supported	Not supported. Always dynamic SQL.
Block fetch	Limited block fetch (enhanced with extra query block support)	Continuous block fetch.
Static SQL	Supported	Not supported. (Executed as dynamic SQL).
Dynamic SQL	Supported	Supported.
Two-phase commit	Supported	Supported.
Inactive database access thread	Type 2 inactive threads supported (thread pooling)	Type 1 inactive thread supported.

Figure 1: DB2 Private Protocol and DRDA

requester to obtain a description of input parameters from a remote database in a consistent format.

- Password encryption.

- Data link data type support.
- Support for user-defined data types.
- Support for large objects.

Further information on the various levels can be found on The Open Group Web site.

Further information about which levels of DB2 support which levels of DRDA can be found on IBM's Web site. Figure 1 provides a comparison of features of DRDA and the DB2 Private Protocol.

Some of the advantages of using DRDA are:

- The message format of DRDA is simpler and more optimized (condensed) than that of the DB2 Private Protocol. This improves network traffic performance.
- Stored procedures can be used in DRDA while not supported by DB2PP. The use of stored procedures can help minimize the network traffic between the application requester and application server.
- Since the DB2 Private Protocol does not support remote bind, all SQL is treated as dynamic SQL, which may cause performance degradation. This is also known as deferred embedded SQL.
- DRDA supports TCP/IP.
- Thread pooling is available.
- The DRDA block fetching protocol was enhanced to make it perform more like the DB2 Private Protocol's continuous block fetch protocol. Actually, the DRDA implementation is better in a way, because it allows both partners (AR and AS) to negotiate the number of blocks that can be sent in a single message, to avoid flooding the requester with data that it cannot keep up with.

John Bradley
Systems Programmer
Meerkat Computer Services (UK)

© Xephon 2005

DB2 LUW – using maintained tables in a federated environment

This article looks at using maintained tables in a federated scenario. Maintained tables have been around for a long time (they were called Automatic Summary Tables previously), but what's new in V8.2, amongst other things, is that they now do not have to include a **group by** keyword and you can 'switch' a table between being maintained and not maintained. Why would I want to use maintained tables? Well, say I was accessing a federated non-relational source (a flat file), then I could try to access that source for every query. Or, if the file isn't that large and not updated frequently, I could create a local DB2-maintained table based on the federated source and then update this table on a regular basis. If you do this, don't forget that the data will only be as current as the last refresh time. To start with, let's create a nickname to our flat file source, run a query against it, and look at the EXPLAIN output.

The commands executed below were carried out on a Windows 2000 Professional system running DB2 UDB V8.1 FP8 using the SAMPLE database and the system administrator userid (db2admin).

Create the two flat files (c:\ii_ff\flatfile1.txt and c:\ii_ff\flatfile2.txt) to contain three columns (id, region, wid), with a delimiter of '|'.
'|'.

The file flatfile1.txt will contain:

```
0|Asia|100  
1|America|200  
2|Europe|300
```

The file flatfile2.txt will contain:

```
4|Aus|400  
5|Antarc|500
```

We will be using the SAMPLE database:

```
>db2 connect to sample user db2admin using xxxxxxxx
```

Create the flat file wrapper:

```
>db2 CREATE WRAPPER "FLATFILE" LIBRARY 'db2lsfile.dll' OPTIONS( ADD
DB2_FENCED 'N')
```

Create the flat file server:

```
>db2 CREATE SERVER FLATSERV WRAPPER "FLATFILE"
```

Create the flat file nicknames (note, that the files do not have to exist for the command to be successful):

```
>db2 "CREATE NICKNAME DB2ADMIN.FLATN1 ( ID INTEGER ,REGION CHARACTER
(10) ,wid INTEGER ) FOR SERVER FLATSERV OPTIONS(COLUMN_DELIMITER '|' ,
FILE_PATH 'c:\ii_ff\flatfile1.txt')"
```

```
>db2 "CREATE NICKNAME DB2ADMIN.FLATN2 ( ID INTEGER ,REGION CHARACTER
(10) ,wid INTEGER ) FOR SERVER FLATSERV OPTIONS(COLUMN_DELIMITER '|' ,
FILE_PATH 'c:\ii_ff\flatfile2.txt')"
```

Check that the nicknames were created successfully by selecting from them:

```
>db2 select * from flatn1
```

ID	REGION	WID
0	Asia	100
1	America	200
2	Europe	300

3 record(s) selected.

```
>db2 select * from flatn2
```

ID	REGION	WID
4	Aus	400
5	Antarc	500

2 record(s) selected.

(If you get a SQL1822N message, make sure that you have pressed the delete key immediately after the last line of the input file.)

We can see that the nicknames were successfully created. Let's create a view (flatview) over these two nicknames:

```
>db2 create view flatview as select * from flatn1 union all select *
from flatn2
```

Let's check the state of the view:

```
>db2 select substr(viewname,1,10),valid from syscat.views where viewname
= 'FLATVIEW'
```

```
1          VALID
-----
FLATVIEW   Y
```

And selecting from it:

```
>db2 select * from flatview
```

```
ID          REGION      WID
-----
      4 Aus              400
      5 Antarc           500
      0 Asia             100
      1 America          200
      2 Europe           300
```

So let's look at the EXPLAIN output for running this query. To check this I used the db2expln command with an input file called runsql01.txt containing:

```
select * from db2admin.flatview
```

And I ran the db2expln command as:

```
>db2expln -d sample -t -f runsql01.txt
```

(where **-d** specifies the database, **-t** that the output should be written to the screen, and **-f** for the input file name).

The bottom half of the screen output shows:

SQL Statement:

```
select *
from db2admin.flatview

| Ship Distributed Subquery #2
| | #Columns = 3
UNION
| Ship Distributed Subquery #1
| | #Columns = 3
)
```

```
Return Data to Application
| #Columns = 3
```

```
Nicknames Referenced:
  DB2ADMIN.FLATN1  ID = 32771
    Source File = c:\ii_ff\flatfile1.txt
```

```
Nicknames Referenced:
  DB2ADMIN.FLATN2  ID = 32770
    Source File = c:\ii_ff\flatfile2.txt
```

We can see our query select * from flatview and that the source tables c:\ii_ff\flatfile1.txt and flatfile2 are being accessed via the flatn1 and flatn2 nicknames.

So now let's create a maintained table for our flat files over the view summing up the *wid* column. This is a two step process – the first step is to create the maintained table and the second step is to change the value of the special registers CURRENT MAINTAINED TABLE TYPES FOR OPTIMIZATION and CURRENT REFRESH AGE so that the maintained table can be used by the optimizer.

So the first step we need to take is to create a maintained table (let's call it sumflat) based on the db2admin.flatfview view and summing up column *wid*. We do this as follows:

```
>db2 create table sumflat (total) as (select sum(wid) from flatview)
data initially deferred refresh deferred
```

Note that we don't have to specify the SUMMARY keyword.

Let's check how the table is registered in the catalog tables:

```
>db2 select substr(tabname,1,10),type from syscat.tables where tabname
like '%FLAT%'
```

1	TYPE
FLATN2	N
FLATN1	N
FLATVIEW	V
SUMFLAT	S

The flatn1 and flatn2 files have a type of 'N' (which means nickname), and the sumflat file has a type of 'S' (which means summary table) for backward compatibility.

Before you can select from this table you need to populate it (otherwise you will get a SQL0668N error). Populate the maintained table using the command:

```
>db2 refresh table sumflat
```

You would also use this command to ‘refresh’ it at whatever frequency you decide is appropriate.

We can select from our maintained table as below:

```
>db2 select * from sumflat
```

```
TOTAL
-----
      1500
```

But that’s not the point! We don’t want to specifically name the maintained table in any query – we want the optimizer to select it automatically when it matches the user query.

So, the second step is to change the value of the special registers. I can change the value of the special register CURRENT MAINTAINED TABLE TYPES FOR OPTIMIZATION (whose default value is SYSTEM) by connecting to the database (>db2 connect to sample user db2admin using xxxxxxxx) and then issuing the command:

```
>db2 SET CURRENT MAINTAINED TABLE TYPES FOR OPTIMIZATION ALL
```

The *ALL* means that all types of maintained tables controlled will be considered for dynamic SQL. What other options are there? You could specify NONE, FEDERATED_TOOL, SYSTEM, and USER. The NONE keyword means that you won’t be using maintained tables. The FEDERATED_TOOL keyword means that tables maintained by a federated tool will be supported. The SYSTEM and USER keywords refer to system and user maintained tables.

The CURRENT REFRESH AGE special register has to be set to a value other than zero (its default value) for us to use our maintained table in any query. We set it using the command:

```
>db2 set current refresh age any
```

Let’s bring it all together in our second example. Let’s create

a new input file (runsql02.txt) for the db2expln command, as below, where we set our special registers and issue our select query from our flat files view:

```
set current maintained table types for optimization all
set current refresh age any
select sum(wid) from db2admin.flatview
```

And rerun our db2expln command using the new input file:

```
>db2expln -d sample -t -f runsql02.txt
```

The bottom half of the screen output shows:

SQL Statement:

```
select sum(wid)
from db2admin.flatview
```

Access Materialized Query Table Name = DB2ADMIN.SUMFLAT ID = 4,31

We can see that the SQL statement is still select sum(wid) from db2admin.flatview, but that we are now using our maintained table sumflat.

Another perhaps simpler test to see whether we are using a maintained table is to rename one of the source files and rerun the select sum(wid) from db2admin.flatview query from the CLP. If we are not using the maintained table, the query will work only partially! We can issue the commands below from a CLP session:

```
>db2 set current maintained table types for optimization all
```

```
>db2 set current refresh age any
```

And check both values by using the db2 values command:

```
>db2 values current maintained table types for optimization
```

```
1
-----
ALL
```

And:

```
>db2 values current refresh age
```

```
1
```

9999999999999999.000000

The keyword ANY equates to a refresh age of 99999 (as shown).

So we can now issue our select command from this CLP session (try renaming the source file and see what happens – or change a row in it and see what happens – we will still be accessing our maintained table). If we opened a new CLP session and issued our select command we would pick up the values from the source table (and not the maintained table).

Note that both the special registers CURRENT MAINTAINED TABLE TYPES FOR OPTIMIZATION and CURRENT REFRESH AGE need to be set for maintained tables to be considered by the optimizer.

Can I selectively tell the optimizer to use/not use the maintained tables? You can control the use at the database level, but not at an individual query level within the same CLP session. The Information Center says [that a maintained table cannot be] ‘under transaction control’. What you have to remember is that the special register values are set at the database level for each CLP session – so if two users open two CLP sessions and one of them sets the special registers and the other one doesn’t, the first user will use the maintained tables and the second user won’t.

The examples I used were simple flat files, but you can use maintained tables for relational and non-relational sources. If you are going to use maintained tables then remind your end users about what that means with regard to the ‘freshness’ of the data.

What happens if I delete one of the underlying flat files? Let’s delete flatfile1.txt and check what happens if I try to select from the view:

```
>rem flatfile1.txt
```

```
>db2 select * from flatview
```

```
ID          REGION      WID
```

```

-----
          4 Aus                      400
          5 Antarc                   500
SQL1822N Unexpected error code "ERRNO = 2" received from data source
"FLATSERV". Associated text and tokens are "Unable to read file".
SQLSTATE=560BD

```

You can see that we still get the results from the flat file that still exists and an error from the one that was deleted.

If we check the view, we can see that it is still valid:

```
>db2 select substr(viewname,1,10),valid from syscat.views where viewname
= 'FLATVIEW'
```

```

1          VALID
-----
FLATVIEW   Y

```

Now what happens if I drop the nickname for flatfile1.txt?

```
>db2 drop nickname flatn1
```

If we now check the catalog for the view:

```
>db2 select substr(viewname,1,10),valid from syscat.views where viewname
= 'FLATVIEW'
```

```

1          VALID
-----
FLATVIEW   X

```

We can see that the view is not invalid. And if we check the catalog for the tables:

```
>db2 select substr(tabname,1,10),type from syscat.tables where tabname
like '%FLAT%'
```

```

1          TYPE
-----
FLATN2     N
FLATVIEW   V

```

we can see that not only has the entry for flatn1 been deleted but the summary table has been dropped. This is an important point to remember: if you drop a nickname then any maintained tables based on that nickname will also be dropped.

Note: you can use the ALTER TABLE command to change a

regular table to a summary table (and *vice versa*), as shown in the example below:

```
>db2 create table sumtab2 like flatview
```

```
>db2 select substr(tabname,1,10),type from syscat.tables where tabname =  
'SUMTAB2'
```

Check the table definition:

```
1          TYPE  
-----  
SUMTAB2    T
```

Now alter the table to make it a summary table:

```
>db2 alter table sumtab2 add materialized query (select * from flatview)  
data initially deferred refresh deferred maintained by user
```

And check the catalog entry:

```
>db2 select substr(tabname,1,10),type from syscat.tables where tabname =  
'SUMTAB2'
```

```
1          TYPE  
-----  
SUMTAB2    S
```

You can see that the table is now defined as 'S' (for summary). To change the table back again from summary to regular issue:

```
>db2 alter table sumtab2 set summary as definition only
```

And check the catalog entry:

```
>db2 select substr(tabname,1,10),type from syscat.tables where tabname =  
'SUMTAB2'
```

```
1          TYPE  
-----  
SUMTAB2    T
```

You can see that the table is again defined as 'T'.

If you want to drop the nickname without losing the summary table, you need to convert it to a normal table before dropping the nickname, as shown below:

```
>db2 select substr(tabname,1,10),type from syscat.tables where tabname  
like '%FLAT%'
```

1	TYPE
-----	----
FLATN1	N
FLATN2	N
FLATVIEW	V
SUMFLAT	S

We can see that the sumflat table is still a summary table. Now alter it to be a regular table:

```
>db2 alter table sumflat set summary as definition only
```

And let's check the definitions again:

```
>db2 select substr(tabname,1,10),type from syscat.tables where tabname
like '%FLAT%'
```

1	TYPE
-----	----
FLATN1	N
FLATN2	N
FLATVIEW	V
SUMFLAT	T

We can see that sumflat is now a regular table. So if we now drop the nickname:

```
>db2 drop nickname flatn1
```

And check the catalog:

```
>db2 select substr(tabname,1,10),type from syscat.tables where tabname
like '%FLAT%'
```

1	TYPE
-----	----
FLATN2	N
FLATVIEW	V
SUMFLAT	T

You can see that the nickname entry flatn1 has gone but the sumflat file is still there.

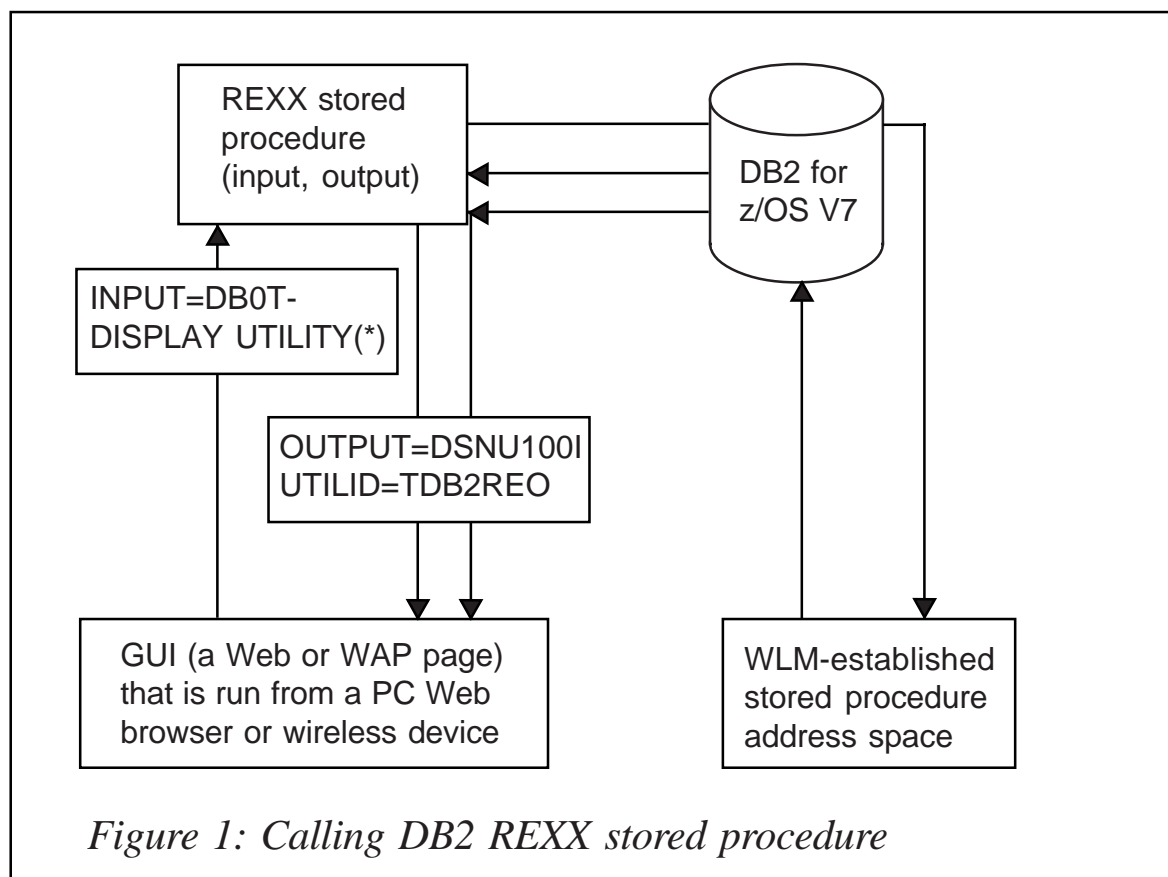
I hope I have shown how easy it is to set up and administer maintained tables in a federated environment and demonstrated some of the points to be aware of when dropping nicknames.

C Leonard
Freelance Consultant (UK)

© Xephon 2005

Managing DB2 for z/OS through WAP and Web environments

24x7 availability is vital in production banking environments. Achieving high availability for DB2 depends on the accurate management of system and DB2 resources. Besides the usual TN3270 host session method, managing DB2 through WAP and Web environments, with the help of stored procedures running on DB2 for z/OS environment, helps database staff to control DB2 remotely. This article shows a useful Web page where users can choose (just click on an HTML radio button) the DB2 command they wish to run. This speeds up operation and decreases the likelihood of making errors while entering commands. This project also shows direct DB2 for z/OS access through wireless devices (mobile/cellular phones, palm tops etc). Hence, database staff can control DB2 directly when they are away from the office. It helps staff with



management and availability. By calling DB2 REXX stored procedures from the intranet and WAP environments, all REXX programs (even non-DB2 related) can be started from an open platform. For example, even JCL can be submitted from mobile (cell) phones. This article covers the implementation of this whole idea.

DB2 for z/OS commands can call JCL, IMS, CICS, application programs, and REXX programs. REXX programs can be run as DB2 stored procedures. DB2 stored procedures can be called through programming languages (Java, Visual Basic, .asp, .jsp, etc) that run SQL CALL statements. In this project, a DB2 command is sent to a DB2 REXX stored procedure via a Web browser. The stored procedure runs the command and sends back the result to the Web browser. The logic diagram is shown in Figure 1.

Here are the steps to implement this project:

1 Preparation of DB2 REXX stored procedure.

The stored procedure is defined to DB2.

```
CREATE PROCEDURE SYSPROC.DB2COMME
(
  IN      MYINPUT1          CHAR          (104),
  OUT     MYOUTPUT          VARCHAR      (32703 )
)
DYNAMIC RESULT SET 1 EXTERNAL NAME DB2COMME
LANGUAGE REXX PARAMETER STYLE GENERAL
NOT DETERMINISTIC  FENCED  CALLED ON NULL INPUT  MODIFIES SQL DATA
NO DBINFO WLM ENVIRONMENT DB0TWLM3  STAY RESIDENT NO
PROGRAM TYPE MAIN SECURITY DB2  COMMIT ON RETURN NO;
COMMIT;
GRANT EXECUTE ON PROCEDURE SYSPROC.DB2COMME TO PUBLIC;
COMMIT;
```

On the MVS side, WLM (Work Load Manager) and RRS (Resource Recovery) have to be installed. A WLM application environment named DB0TWLM3 is defined. DB0TWLM3 address space started task JCL is put into SYS2.PROCLIB. Note that the system REXX library is defined with SYSEXEC DD S000.COMM.REXX.

SYS2.PROCLIB(DB0TWLM3)

```

/*****
/*      JCL FOR RUNNING THE WLM-ESTABLISHED STORED PROCEDURES
/*      ADDRESS SPACE
/*      RGN      -- THE MVS REGION SIZE FOR THE ADDRESS SPACE.
/*      DB2SSN   -- THE DB2 SUBSYSTEM NAME.
/*      NUMTCB   -- THE NUMBER OF TCBS USED TO PROCESS
/*                  END USER REQUESTS.
/*      APPLENV  -- THE MVS WLM APPLICATION ENVIRONMENT
/*                  SUPPORTED BY THIS JCL PROCEDURE.
*****/
//DB0TWLM3 PROC RGN=0K,APPLENV=DB0TWLM3,DB2SSN=&IWMSSNM,NUMTCB=8
//IEFPROC EXEC PGM=DSNX9WLM,REGION=&RGN,TIME=NOLIMIT,
//          PARM='&DB2SSN,&NUMTCB,&APPLENV'
//STEPLIB DD DISP=SHR,DSN=DSN710.RUNLIB.LOAD
//          DD DISP=SHR,DSN=CEE.SCEERUN
//          DD DISP=SHR,DSN=DSN710.SDSNEXIT
//          DD DISP=SHR,DSN=DSN710.SDSNLOAD
//          DD DISP=SHR,DSN=T000.COMM.SPLOADBA
//SYSEXEC DD DISP=SHR,DSN=S000.COMM.REXX
//          DD DISP=SHR,DSN=D000.COMM.CLIB
//TRANFILE DD DISP=SHR,DSN=TCPIP.TCPIPT.STANDARD.TCPXLBIN
//SYSPRINT DD SYSOUT=*
//CEEDUMP DD SYSOUT=*
//SYSUDUMP DD SYSOUT=*
//SYSMDUMP DD SYSOUT=*
//SYSTSPRT DD SYSOUT=*
```

The DB2COMME and DB2REXX programs that will run the DB2 commands are put into the system REXX library dataset (S000.COMM.REXX)

S000.COMM.REXX(DB2COMME)

```

/* REXX */
  PARSE ARG SSID_COMMAND          /* Get the SSID to connect to */
  SAY 'SSID_COMMAND=' || SSID_COMMAND /* and the DB2 command to be */
  SSID = LEFT(SSID_COMMAND,4)      /* executed */
  say 'SSID=' || SSID
  COMMAND = SUBSTR(SSID_COMMAND,5,100)
  say 'COMMAND=' || COMMAND
/* SSID = 'DB0T' */
/* COMMAND = '-DISPLAY GROUP' */
/*****
/* Set up the host command environment for SQL calls. */
*****/
$SUBCOM DSNREXX$                /* Host cmd env available? */
IF RC THEN                       /* No--make one */
```

```

        S_RC = RXSUBCOM('ADD','DSNREXX','DSNREXX')
/*****
/*  Connect to the DB2 subsystem.
*****/
/*  ADDRESS DSNREXX $CONNECT$ SSID */
say 'a1'
/*IF SQLCODE <> 0 THEN CALL SQLCA */
say 'a2'
PROC = 'COMMAND'
RESULTSIZ = 32703
RESULT = LEFT(' ',RESULTSIZ,' ')
/*****
/*  Call the stored procedure that executes the DB2 command.
/*  The input variable (COMMAND) contains the DB2 command.
/*  The output variable (RESULT) will contain the return area
/*  from the IFI COMMAND call after the stored procedure
/*  executes.
*****/
ADDRESS DSNREXX $EXECSQL$ ,
$CALL$ DB2REXX $( :COMMAND, :RESULT)$
say 'a3'
IF SQLCODE < 0 THEN CALL SQLCA
/*SAY 'RETCODE ='RETCODE
/*SAY 'SQLCODE ='SQLCODE
/*SAY 'SQLERRMC ='SQLERRMC
/*SAY 'SQLERRP ='SQLERRP
/*SAY 'SQLERRD ='SQLERRD.1',',
/*      || SQLERRD.2',',
/*      || SQLERRD.3',',
/*      || SQLERRD.4',',
/*      || SQLERRD.5',',
/*      || SQLERRD.6
/*SAY 'SQLWARN ='SQLWARN.0',',
/*      || SQLWARN.1',',
/*      || SQLWARN.2',',
/*      || SQLWARN.3',',
/*      || SQLWARN.4',',
/*      || SQLWARN.5',',
/*      || SQLWARN.6',',
/*      || SQLWARN.7',',
/*      || SQLWARN.8',',
/*      || SQLWARN.9',',
/*      || SQLWARN.10
/* SAY 'SQLSTATE='SQLSTATE
/* SAY C2X(RESULT) '$'$||RESULT||'$'$ */
/*****
/*  Display the IFI return area in hexadecimal.
*****/
OFFSET = 4+1
TOTLEN = LENGTH(RESULT)

```

```

MYOUTPUT==$$
DO WHILE ( OFFSET < TOTLEN )
  LEN = C2D(SUBSTR(RESULT,OFFSET,2))
  SAY SUBSTR(RESULT,OFFSET+4,LEN-4-1)
  MYOUTPUT = MYOUTPUT || SUBSTR(RESULT,OFFSET+4,LEN-4-1) ||$@$
  OFFSET = OFFSET + LEN
END
/*MYOUTPUT = 11 */
/* MYOUTPUT =$1234567891234567$ */
RETURN MYOUTPUT
/*****
/* Routine to display the SQLCA */
*****/
SQLCA:
TRACE 0
SAY 'SQLCODE ='SQLCODE
SAY 'SQLERRMC ='SQLERRMC
SAY 'SQLERRP ='SQLERRP
SAY 'SQLERRD ='SQLERRD.1',' ,
      || SQLERRD.2',' ,
      || SQLERRD.3',' ,
      || SQLERRD.4',' ,
      || SQLERRD.5',' ,
      || SQLERRD.6
SAY 'SQLWARN ='SQLWARN.0',' ,
      || SQLWARN.1',' ,
      || SQLWARN.2',' ,
      || SQLWARN.3',' ,
      || SQLWARN.4',' ,
      || SQLWARN.5',' ,
      || SQLWARN.6',' ,
      || SQLWARN.7',' ,
      || SQLWARN.8',' ,
      || SQLWARN.9',' ,
      || SQLWARN.10
SAY 'SQLSTATE='SQLSTATE
/* EXIT 99 */

```

S000.COMM.REXX(DB2REXX)

```

/* REXX */
PARSE UPPER ARG CMD                      /* Get the DB2 command text */
                                          /* Remove enclosing quotes */
IF LEFT(CMD,2) = '$'$ & RIGHT(CMD,2) = '$'$ THEN
CMD = SUBSTR(CMD,2,LENGTH(CMD)-2)
ELSE
IF LEFT(CMD,2) = '$$$'$ & RIGHT(CMD,2) = '$$$$' THEN
CMD = SUBSTR(CMD,3,LENGTH(CMD)-4)
COMMAND    = SUBSTR($COMMAND$,1,18,$ $)
/*****

```

```

/* Set up the IFCA, return area, and output area for the */
/* IFI COMMAND call. */
/*****/
IFCA = SUBSTR('00'X,1,180,'00'X)
IFCA = OVERLAY(D2C(LENGTH(IFCA),2),IFCA,1+0)
IFCA = OVERLAY($IFCA$,IFCA,4+1)
RTRNAREASIZE = 262144 /*1048572*/
RTRNAREA = D2C(RTRNAREASIZE+4,4)LEFT(' ',RTRNAREASIZE,' ')
OUTPUT = D2C(LENGTH(CMD)+4,2)||'0000'X||CMD
BUFFER = SUBSTR($ $,1,16,$ $)
/*****/
/* Make the IFI COMMAND call. */
/*****/
ADDRESS LINKPGM $DSNWLIR COMMAND IFCA RTRNAREA OUTPUT$
WRC = RC
RTRN= SUBSTR(IFCA,12+1,4)
REAS= SUBSTR(IFCA,16+1,4)
TOTLEN = C2D(SUBSTR(IFCA,20+1,4))
/*****/
/* Set up the host command environment for SQL calls. */
/*****/
$SUBCOM DSNREXX$ /* Host cmd env available? */
IF RC THEN /* No--add host cmd env */
  S_RC = RXSUBCOM('ADD','DSNREXX','DSNREXX')
/*****/
/* Set up SQL statements to insert command output messages */
/* into a temporary table. */
/*****/
SQLSTMT='INSERT INTO SYSIBM.SYSPRINT(SEQNO,TEXT) VALUES(?,?)'
ADDRESS DSNREXX $EXECSQL DECLARE C1 CURSOR FOR S1$
IF SQLCODE <> 0 THEN CALL SQLCA
ADDRESS DSNREXX $EXECSQL PREPARE S1 FROM :SQLSTMT$
IF SQLCODE <> 0 THEN CALL SQLCA
/*****/
/* Extract messages from the return area and insert them into */
/* the temporary table. */
/*****/
SEQNO = 0
OFFSET = 4+1
DO WHILE ( OFFSET < TOTLEN )
  LEN = C2D(SUBSTR(RTRNAREA,OFFSET,2))
  SEQNO = SEQNO + 1
  TEXT = SUBSTR(RTRNAREA,OFFSET+4,LEN-4-1)
  ADDRESS DSNREXX $EXECSQL EXECUTE S1 USING :SEQNO,:TEXT$
  IF SQLCODE <> 0 THEN CALL SQLCA
  OFFSET = OFFSET + LEN
END
/*****/
/* Set up a cursor for a result set containing the command */
/* output messages from the temporary table. */
/*****/

```

```

/*****/
SQLSTMT='SELECT SEQNO,TEXT FROM SYSIBM.SYSPRINT ORDER BY SEQNO'
ADDRESS DSNREXX $EXECSQL DECLARE C2 CURSOR FOR S2$
IF SQLCODE <> 0 THEN CALL SQLCA
ADDRESS DSNREXX $EXECSQL PREPARE S2 FROM :SQLSTMT$
IF SQLCODE <> 0 THEN CALL SQLCA
/*****/
/* Open the cursor to return the message output result set to */
/* the caller. */
/*****/
ADDRESS DSNREXX $EXECSQL OPEN C2$
IF SQLCODE <> 0 THEN CALL SQLCA
S_RC = RXSUBCOM('DELETE','DSNREXX','DSNREXX') /* REMOVE CMD ENV */
EXIT SUBSTR(RTRNAREA,1,TOTLEN+4)
/*****/
/* Routine to display the SQLCA */
/*****/

SQLCA:
SAY 'SQLCODE ='SQLCODE
SAY 'SQLERRMC ='SQLERRMC
SAY 'SQLERRP ='SQLERRP
SAY 'SQLERRD ='SQLERRD.1',' ,
      || SQLERRD.2',' ,
      || SQLERRD.3',' ,
      || SQLERRD.4',' ,
      || SQLERRD.5',' ,
      || SQLERRD.6
SAY 'SQLWARN ='SQLWARN.0',' ,
      || SQLWARN.1',' ,
      || SQLWARN.2',' ,
      || SQLWARN.3',' ,
      || SQLWARN.4',' ,
      || SQLWARN.5',' ,
      || SQLWARN.6',' ,
      || SQLWARN.7',' ,
      || SQLWARN.8',' ,
      || SQLWARN.9',' ,
      || SQLWARN.10
SAY 'SQLSTATE='SQLSTATE
SAY 'SQLCODE ='SQLCODE
EXIT 'SQLERRMC ='SQLERRMC';' ,
|| 'SQLERRP ='SQLERRP';' ,
|| 'SQLERRD ='SQLERRD.1',' ,
      || SQLERRD.2',' ,
      || SQLERRD.3',' ,
      || SQLERRD.4',' ,
      || SQLERRD.5',' ,
      || SQLERRD.6';' ,
|| 'SQLWARN ='SQLWARN.0',' ,
      || SQLWARN.1',' ,

```

```

|| SQLWARN.2',' ,
|| SQLWARN.3',' ,
|| SQLWARN.4',' ,
|| SQLWARN.5',' ,
|| SQLWARN.6',' ,
|| SQLWARN.7',' ,
|| SQLWARN.8',' ,
|| SQLWARN.9',' ,
|| SQLWARN.10';' ,
|| 'SQLSTATE='SQLSTATE';'

```

PREPARATION OF THE WEB ENVIRONMENT

DB2 stored procedures can be called from a Web browser via a .jsp (Java server page) or .asp (active server page) file. For our project, we will use an .asp page. In order to serve active server pages, a Web server (for our project IIS (Internet Information Server)) is installed on a Windows machine. An .asp is prepared and put into an IIS directory.

```

C:\Inetpub\wwwroot\test\db2command\Db2comme2.asp
<%
session("aktifortam")=request.form("ortam")
session("aktifcommand")=request.form("mycommand1")
session("aktifdb2")=request.form("myssid")
if session("aktifortam")="" then session("aktifortam")="TEST"
if session("aktifdb2")="" then session("aktifdb2")="DB1T"
if session("aktifcommand")="" then session("aktifcommand")="DISPLAY
DB(DTGNL*) SP(*) USE LIMIT(*)"
%>
<!DOCTYPE HTML PUBLIC "-//IETF//DTD HTML//EN">
<html>
<head>
<title>Akbank T.A.S@ 2004</title>
<META HTTP-EQUIV="Content-Type" CONTENT="text/html; charset=windows-
1254">
<META HTTP-EQUIV="Content-Type" CONTENT="text/html; charset=ISO-8859-9">
<META HTTP-EQUIV="content-language" content="TR">
<META HTTP-EQUIV="Copyright" CONTENT="Akbank T.A.S@ 2004">
<META NAME="Pragma" CONTENT="no-cache">
<META HTTP-EQUIV="cache-control" CONTENT="no-cache">
<style>
    .tarih{ font-size: 7pt; font-family: Courier New; color:blue }
    .tarih{ font-size: 8pt; font-family: Arial; color:blue }
    .firstcol{ font-size: 8pt; font-weight:bold; font-family: Arial;
color:blue }
    .firstcol1{ font-size: 8pt; font-weight:bold; font-family: Arial;
color:blue}

```



```

        .tarih1{ font-size: 8pt; font-weight:bold; font-family: Arial;
color:blue }
        .tarih2{ font-size: 8pt; font-weight:bold; font-family: Arial;
color:blue }
</style>
<script LANGUAGE="JavaScript">
<!--
function radioClick (f,i) {
    f.mycommand1.value = f.komut[i].value;
    f.myhidden.value=i;
    return true;
}
//-->
</script>
</HEAD>
<body bgcolor=white text="blue">
<form action="db2comme2.asp" method="post" name="myform">
<p align="left">
<input TYPE="hidden" VALUE="" NAME="myhidden">
Choose the environment:
<BR>
<input TYPE="radio" VALUE="TEST" NAME="ortam"
<%if session("aktifortam")="TEST" then response.write "CHECKED"%> >
TEST ENVIRONMENT
<BR>
<input TYPE="radio" VALUE="PRODUCTION" NAME="ortam"
<%if session("aktifortam")="PRODUCTION" then response.write "CHECKED"%>
> PRODUCTION ENVIRONMENT
<br>
Enter the name of DB2 datasharing group :
<input size="4" maxlength="4" name="myssid" value="<%response.write
session("aktifdb2")%>" >
<br><br>
You can either CLICK ON or WRITE DOWN the DB2 command that you would
like to run <br>
<% response.write session("selectedradiobutton") %>
<TABLE BORDER=1 width="60%">
<TR><TD><BR>
<input TYPE="radio" VALUE="DISPLAY DB(DTGNL*) SP(*) USE LIMIT(*)"
NAME="komut"
onclick="return radioClick (document.forms[0],0)"
<%if cint(request.form("myhidden"))=0 then response.write "CHECKED"%> >
DISPLAY DB(DTGNL*) SP(*) USE LIMIT(*)
<BR>
<input TYPE="radio" VALUE="START DB(DTGNL01) SP(STGNLG01) ACCESS(FORCE)"
NAME="komut"
onclick="return radioClick (document.forms[0],1)"
<%if cint(request.form("myhidden"))=1 then response.write "CHECKED"%> >
START DB(DTGNL01) SP(STGNLG01) ACCESS(FORCE)
<BR>

```

```

☐

```

```

</TD></TR>
</TABLE>
DB2 Command that will run:
<input size="100" maxlength="100" name="mycommand1"
value="<%response.write session("aktifcommand")%>" >
<BR><BR><br>
<input type="submit" name="submit" value="RUN THE DB2 COMMAND"></p>
</form>
<!-- #include file="constans.inc" -->
<%
if cstr(request.form("submit"))="RUN THE DB2 COMMAND" then
'beginning of the result
Set Connvs = Server.CreateObject("ADODB.Connection")
MYCOMMAND=request.form("mycommand1")
MYSSID=request.form("myssid")
if session("aktifortam")="PRODUCTION" then
    Session("ConnectionString") =
    "DSN=DB2PRODUCTION;UID=akbank;PWD=btvyg"
else
if cstr(UCASE(MYSSID))="DB0T" then Session("ConnectionString") =
"DSN=DB2MVS;UID=akbank;PWD=btvyg"
if cstr(UCASE(MYSSID))="DB1T" then Session("ConnectionString") =
"DSN=DB2MVS;UID=akbank;PWD=btvyg"
if cstr(UCASE(MYSSID))="DB2T" then Session("ConnectionString") =
"DSN=DB2DB3TS;UID=akbank;PWD=btvyg"
if cstr(UCASE(MYSSID))="DB3T" then Session("ConnectionString") =
"DSN=DB2DB3TS;UID=akbank;PWD=btvyg"
end if
Connvs.Open Session("ConnectionString")
set cmd = Server.CreateObject("ADODB.Command")
cmd.ActiveConnection = Connvs
set rs = Server.CreateObject("adodb.recordset")
RS.CursorType = 1
RS.LockType = 3
CMD.CommandText = "SYSPROC.DB2COMME"
CMD.CommandType = adCmdStoredProc
myinputoutputvar="STORED PROCEDURE OUTPUT RESULT WILL BE STORED IN THIS
VARIABLE"
MYINPUT1 = MYSSID & "-" & MYCOMMAND
set ADO_Parm1 = CMD.CreateParameter("parm1", adChar, adParamInput, 104,
MYINPUT1)
set ADO_Parm2 = CMD.CreateParameter("parm2", adChar,
adParamOutput,32703,myinputoutputvar)
CMD.Parameters.Append ADO_Parm1
CMD.Parameters.Append ADO_Parm2
cmd.Execute ()
response.write "<br>"
response.write "RESULT OF THE DB2 COMMAND THAT'S JUST BEEN RUN"
response.write "<br>"

```


prepared in .wml format and put into an IIS directory. A proper MIME type configuration for WML pages is set on IIS. (**Start/Settings/Control Panel/Administrative Tools/Internet Information Services/Default Web Site/HTTP Headers/MIME Map**). The MIME types below are added:

- Text/vnd.wap.wml for .wml files (WML source files).
- Application/vnd.wap.wmlc for .wmlc files (WML compiled files).
- Text/vnd.wap.wmlscript for .wmls files (WMLScript source files).
- Application/vnd.wap.wmlscriptc for .wmlsc files (WMLScript compiled files).
- Image/vnd.wap.wbmp for .wbmp files (wireless bitmaps).

A WAP gateway, like Nokia Wap GateWay, Ericsson Wap gateway etc, is installed on the Windows machine. A WAP gateway delivers HTTP data to wireless systems. A WAP Gateway has a WML Encoder and a WML Script Compiler. For our project, we will use the Nokia Wap GateWay. As the client GUI, a wireless phone or a simulation like Nokia Mobile Browser, Opera, etc, is installed on the Windows machine. For our project, we will use the Nokia Mobile Browser. This arrangement is illustrated in Figure 2.

```
C:\inetpub\wwwroot\test\db2command\wml\db2comme.asp
<!-- #include file="constans.inc" -->
<% Response.ContentType = "text/vnd.wap.wml" %>
<?xml version="1.0"?>
<!DOCTYPE wml PUBLIC "-//WAPFORUM//DTD WML 1.1//EN" "http://
www.wapforum.org/DTD/wml_1.1.xml">
<wml>
  <template>
    <do type="prev">
      <noop/>
    </do>
  </template>
  <card id="init" newcontext="true">
    <do type="options" label="Copyright">
      <go href="#copyright"/>
    </do>
    <p align="center">
```

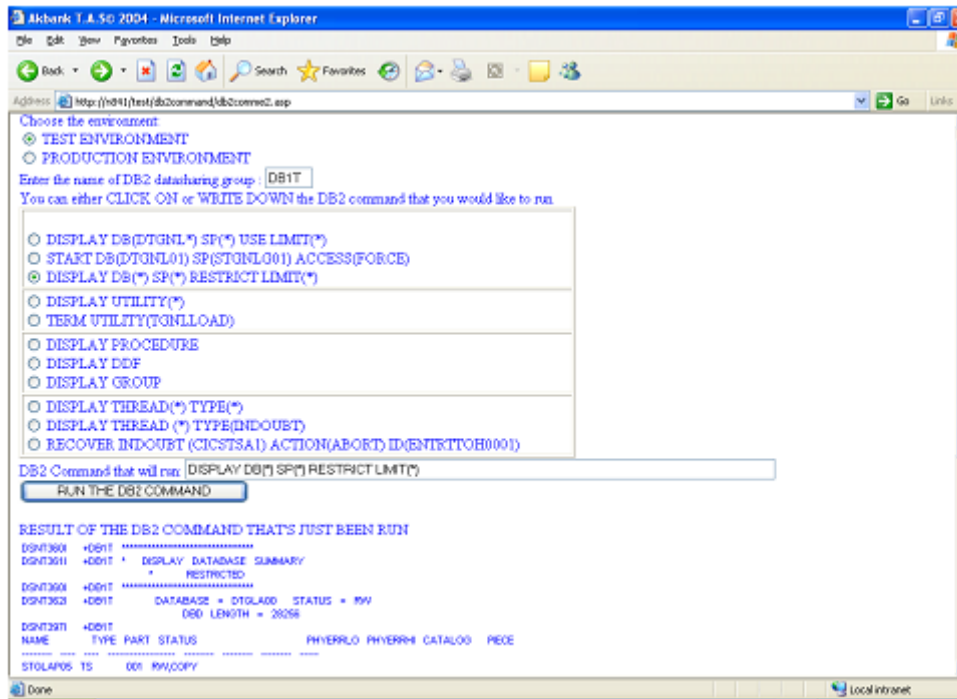


Figure 3: Running a DB2 command from a Web page

```

<br/>
<%
Set Connvs = Server.CreateObject("ADODB.Connection")
Session("ConnectionString") = "DSN=DB2MVS;UID=akbank;PWD=btvyg"
Connvs.Open Session("ConnectionString")
set cmd = Server.CreateObject("ADODB.Command")
cmd.ActiveConnection = Connvs
set rs = Server.CreateObject("adodb.recordset")
RS.CursorType = 1
RS.LockType = 3
CMD.CommandText = "SYSPROC.DB2COMME"
CMD.CommandType = adCmdStoredProc
myinputoutputvar="STORED PROCEDURE OUTPUT RESULT WILL BE STORED IN THIS VARIABLE"
MYCOMMAND="DISPLAY UTILITY(*)"
MYSSID="DB1T"
MYINPUT1 = MYSSID & "-" & MYCOMMAND
set ADO_Parm1 = CMD.CreateParameter("parm1", adChar, adParamInput, 104, MYINPUT1)
set ADO_Parm2 = CMD.CreateParameter("parm2", adChar, adParamOutput, 32703, myinputoutputvar)
CMD.Parameters.Append ADO_Parm1
CMD.Parameters.Append ADO_Parm2

```

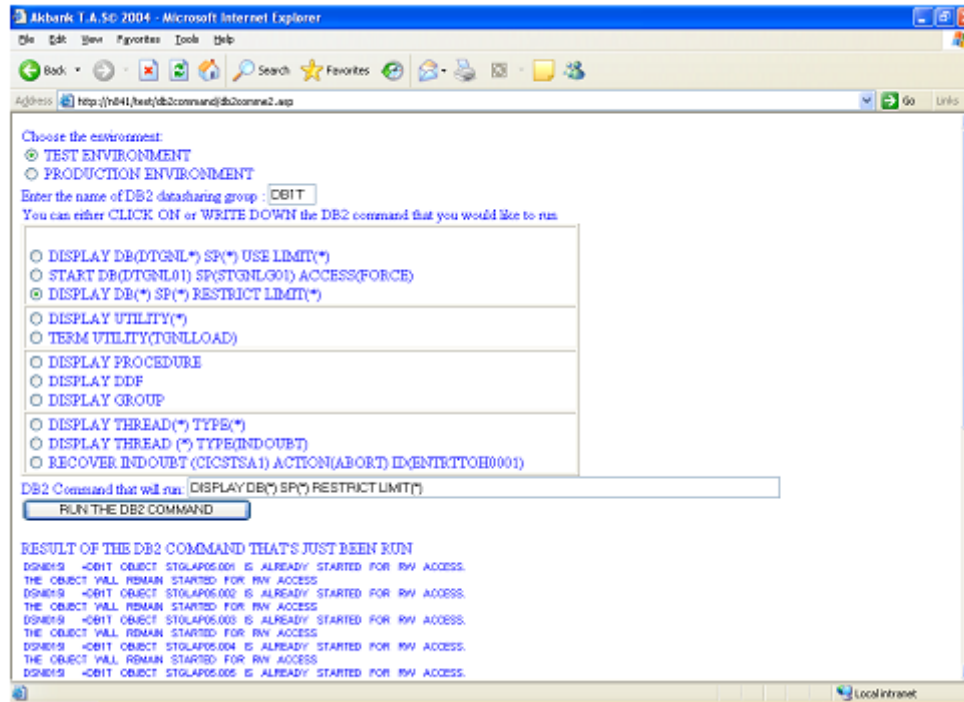


Figure 4: Running a DB2 command from a Web page

```
cmd.Execute ()
response.write "CALISTIRILAN KOMUT:DISPLAY UTILITY(*)"
stringtowrite = cmd.parameters(1)
i=1
while i<= len(stringtowrite)-1
    if mid(stringtowrite,i,1)<>"@" then response.write
mid(stringtowrite,i,1)
    i=i+1
wend
%>
</p>
</card>
<card id="copyright">
    <onevent type="ontimer">
        <prev/>
    </onevent>
    <timer value="25"/>
    <p align="center">
        <small>Copyright&#xA9; 2004<br/>Akbank T.A.S.<br/>All rights
reserved.</small>
    </p>
</card>
</wm1>
```

Now, let's see how the **DISPLAY DB(*) SP(*) RESTRICT LIMIT(*)** command is run through our intranet browser – see Figure 3.

When we run the DB2 command from an intranet browser, on the MVS side the DB0TWLM3 WLM address space is started. The same command output is also printed in both the intranet page and the DB0TWLM3 address space.

```

Display  Filter  View  Print  Options  Help
-----
----- SDSF OUTPUT DISPLAY DB0TWLM3 STC05525  DSID   105 LINE 1,851
COLUMNS 02- 81  COMMAND INPUT ==>
SCROLL ==> CSR  DSN9022I  +DB1T DSNTDDIS 'DISPLAY DATABASE' NORMAL
COMPLETION                      SSID_COMMAND=DB1T-DISPLAY DB(*) SP(*)
RESTRICT LIMIT(*)                      SSID=DB1T
COMMAND=-DISPLAY DB(*) SP(*) RESTRICT LIMIT(*)
DSNT360I  +DB1T *****
DSNT361I  +DB1T *  DISPLAY DATABASE SUMMARY
*  RESTRICTED
DSNT360I  +DB1T *****

```



Figure 5: Running a DB2 command from a WAP browser


```

DSNT362I  +DB1T      DATABASE = DTGLA00  STATUS = RW
DBD LENGTH = 28256
+DB1T
NAME      TYPE PART STATUS      PHYERRLO PHYERRHI CATALOG  PIECE
-----
STGLAP05 TS      001 RW,COPY

```

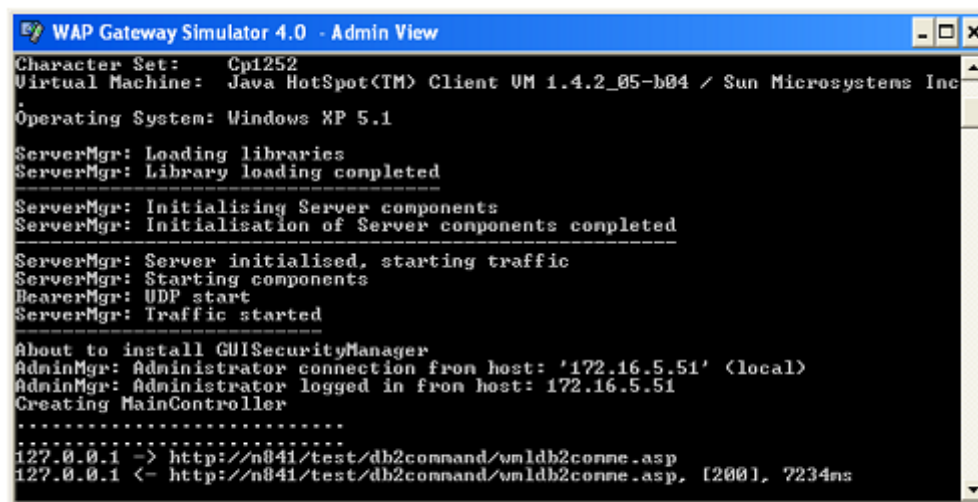
Now, let's see how the **START DB(DTGLA00) SP(STGLAP05) ACCESS(FORCE)** command is run through our intranet browser – see Figure 4.

When we run the DB2 command from an intranet browser, on MVS the DB0TWLM3 WLM address space is started. The same command output is also printed in both the intranet page and the DB0TWLM3 address space.

```

Display Filter View Print Options Help
-----
----- SDSF OUTPUT DISPLAY DB0TWLM3 STC05525  DSID   105 LINE 1,894
COLUMNS 02- 81  COMMAND INPUT ==>
SCROLL ==> CSR  SSID_COMMAND=DB1T-START DB(DTGLA00) SP(STGLAP05 )
ACCESS(FORCE)          SSID=DB1T
COMMAND=-START DB(DTGLA00) SP(STGLAP05 ) ACCESS(FORCE)
DSNI015I  +DB1T OBJECT STGLAP05.001 IS ALREADY STARTED FOR RW ACCESS.
THE OBJECT WILL REMAIN STARTED FOR RW ACCESS
DSNI015I  +DB1T OBJECT STGLAP05.002 IS ALREADY STARTED FOR RW ACCESS.
THE OBJECT WILL REMAIN STARTED FOR RW ACCESS
DSNI015I  +DB1T OBJECT STGLAP05.003 IS ALREADY STARTED FOR RW ACCESS.

```



```

WAP Gateway Simulator 4.0 - Admin View
Character Set: Cp1252
Virtual Machine: Java HotSpot(TM) Client UM 1.4.2_05-b04 / Sun Microsystems Inc
Operating System: Windows XP 5.1
ServerMgr: Loading libraries
ServerMgr: Library loading completed
ServerMgr: Initialising Server components
ServerMgr: Initialisation of Server components completed
ServerMgr: Server initialised, starting traffic
ServerMgr: Starting components
ServerMgr: UDP start
ServerMgr: Traffic started
About to install GUISecurityManager
AdminMgr: Administrator connection from host: '172.16.5.51' (local)
AdminMgr: Administrator logged in from host: 172.16.5.51
Creating MainController
.....
127.0.0.1 -> http://n841/test/db2command/wmlldb2conne.asp
127.0.0.1 <- http://n841/test/db2command/wmlldb2conne.asp, [200], 7234ns

```

Figure 6: Network Traffic on WAP Gateway

THE OBJECT WILL REMAIN STARTED FOR RW ACCESS
 DSN1015I +DB1T OBJECT STGLAP05.004 IS ALREADY STARTED FOR RW ACCESS.

Now, let's see how the **DISPLAY UTILITY(*)** command is run through a mobile browser (see Figure 5):

```

    Display Filter View Print ----- SDSF OUTPUT
DISPLAY DB0TWLM3 STC04429 DSID 105 LINE 0 COLUMNS 02- 81 COMMAND
INPUT ==>                                SCROLL ==>CSR
***** TOP SSID_COMMAND=DB1T-DISPLAY
UTILITY(*)                                SSID=DB1T
COMMAND=-DISPLAY UTILITY(*)
DSNU100I +DB2T DSNUGDIS - USERID = STCUSR
MEMBER = DB1T
UTILID =STCUSR.TISTGL25
PROCESSING UTILITY STATEMENT 1
UTILITY = LOAD
PHASE = UTILINIT COUNT = 0
NUMBER OF OBJECTS IN LIST = 1
LAST OBJECT STARTED =1
STATUS = STOPPED                                DSNU100I
+DB2T DSNUGDIS - USERID = STCUSR MEMBER = DB1T

```

When we run the DB2 command from a mobile browser, the same command output is also printed in both the mobile browser .wml page and the DB0TWLM3 address space. The network traffic on the WAP Gateway during DB2 command call is shown in Figure 6.

CONSTANS INC

<%

```

'---- CursorTypeEnum Values ----
Const adOpenForwardOnly = 0
Const adOpenKeyset = 1
Const adOpenDynamic = 2
Const adOpenStatic = 3
'---- CursorOptionEnum Values ----
Const adHoldRecords = &H000000100
Const adMovePrevious = &H000000200
Const adAddNew = &H01000400
Const adDelete = &H01000800
Const adUpdate = &H01008000
Const adBookmark = &H00002000
Const adApproxPosition = &H00004000
Const adUpdateBatch = &H00010000
Const adResync = &H00020000
'---- LockTypeEnum Values ----

```

```

Const adLockReadOnly = 1
Const adLockPessimistic = 2
Const adLockOptimistic = 3
Const adLockBatchOptimistic = 4
'---- CursorLocationEnum Values ----
Const adUseClient = 1
Const adUseServer = 2
Const adUseClientBatch = 3
'---- DataTypeEnum Values ----
Const adEmpty = 0
Const adTinyInt = 16
Const adSmallInt = 2
Const adInteger = 3
Const adBigInt = 20
Const adUnsignedTinyInt = 17
Const adUnsignedSmallInt = 18
Const adUnsignedInt = 19
Const adUnsignedBigInt = 21
Const adSingle = 4
Const adDouble = 5
Const adCurrency = 6
Const adDecimal = 14
Const adNumeric = 131
Const adBoolean = 11
Const adError = 10
Const adUserDefined = 132
Const adVariant = 12
Const adIDispatch = 9
Const adIUnknown = 13
Const adGUID = 72
Const adDate = 7
Const adDBDate = 133
Const adDBTime = 134
Const adDBTimeStamp = 135
Const adBSTR = 8
Const adChar = 129
Const adVarChar = 200
Const adLongVarChar = 201
Const adWChar = 130
Const adVarWChar = 202
Const adLongVarWChar = 203
Const adBinary = 128
Const adVarBinary = 204
Const adLongVarBinary = 205
'---- ConnectPromptEnum Values ----
Const adPromptAlways = 1
Const adPromptComplete = 2
Const adPromptCompleteRequired = 3
Const adPromptNever = 4
'---- ConnectModeEnum Values ----
Const adModeUnknown = 0

```

```

Const adModeRead = 1
Const adModeWrite = 2
Const adModeReadWrite = 3
Const adModeShareDenyRead = 4
Const adModeShareDenyWrite = 8
Const adModeShareExclusive = &Hc
Const adModeShareDenyNone = &H10
'---- IsolationLevelEnum Values ----
Const adXactUnspecified = &Hffffffff
Const adXactChaos = &H00000010
Const adXactReadUncommitted = &H00000100
Const adXactBrowse = &H00000100
Const adXactCursorStability = &H00001000
Const adXactReadCommitted = &H00001000
Const adXactRepeatableRead = &H00010000
Const adXactSerializable = &H00100000
Const adXactIsolated = &H00100000
'---- XactAttributeEnum Values ----
Const adXactPollAsync = 2
Const adXactPollSyncPhaseOne = 4
Const adXactCommitRetaining = &H00020000
Const adXactAbortRetaining = &H00040000
Const adXactAbortAsync = &H00080000
'---- FieldAttributeEnum Values ----
Const adFldBookmark = &H00000001
Const adFldMayDefer = &H00000002
Const adFldUpdatable = &H00000004
Const adFldUnknownUpdatable = &H00000008
Const adFldFixed = &H00000010
Const adFldIsNulllable = &H00000020
Const adFldMayBeNull = &H00000040
Const adFldLong = &H00000080
Const adFldRowID = &H00000100
Const adFldRowVersion = &H00000200
Const adFldCacheDeferred = &H00001000
'---- EditModeEnum Values ----
Const adEditNone = &H0000
Const adEditInProgress = &H0001
Const adEditAdd = &H0002
'---- RecordStatusEnum Values ----
Const adRecOK = &H00000000
Const adRecNew = &H00000001
Const adRecModified = &H00000002
Const adRecDeleted = &H00000004
Const adRecUnmodified = &H00000008

```

Editor's note: this article will be concluded next month.

Kadir Güray Meriç
DB2 Systems Programmer
Akbank (Turkey)

© Kadir Güray Meriç 2005

BMC Software has announced Version 3.0 of SmartDBA Performance Solution for DB2 UDB, its product for managing and tuning DB2 UDB.

The product works with DB2 UDB running on Unix, Linux, and Windows, and provides database analysis and identification and resolution of problems.

SmartDBA Performance Solution for DB2 UDB is designed to allow DBAs to maintain desired performance levels, reduce maintenance needs and timing, and better prepare for future growth of applications. Its built-in intelligence reduces costs traditionally associated with this process by automating data collection and analysing transactions flowing through databases, identifying trends, and detecting abnormalities.

For further information contact:
URL: www.bmc.com/datamanagement.

* * *

Computer Associates has announced Unicenter Database Management r11 for DB2 UDB for Linux, Unix, and Windows.

The product, they claim, enables organizations to optimize and streamline database operations across multi-platform environments. Customers are able to unload data from large databases and transport it quickly to other databases or applications, thereby accelerating business processes and increasing the scalability of their data centre architectures.

A browser-based management console simplifies the unloading of large databases for increased operational efficiency.

For further information contact:
URL: www3.ca.com/press/PressRelease.aspx?CID=69647.

* * *

SAP AG and IBM have announced an optimized version of DB2 UDB to help customers ease configuration, enhance performance, and increase availability of their SAP solutions running on DB2.

SAP and IBM have integrated new features into DB2 Version 8.2.2 that enhance deployment, maintenance, and availability in a scalable architecture for SAP customers.

The new version streamlines the installation and configuration process, and has improved self-managing features, such as DB2's 'SAP tuner' feature developed to auto-configure DB2 in a SAP solutions environment.

The optimized DB2 offering for SAP solutions, including SAP Enterprise Services Architecture (SAPESA), is tailored for both new and existing SAP application developers and customers.

For further information contact:
URL: www.sap.com/services/servsuptech/smp.

* * *

