# 154

# DB2 update

*August 2005*

## In this issue


3 DB2 UDB for LUW 8.2 – an INSERT/SELECT/DELETE scenario

6 DB2 Stinger and HADR

16 Managing DB2 for z/OS through WAP and Web environments – part 2

31 DB2 Web services

51 DB2 news




A itc PUBLICATION

## 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.



## 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.



*Printed in England.*

# DB2 UDB for LUW 8.2 – an INSERT/SELECT/ DELETE scenario

This article looks at the INSERT/SELECT/DELETE command, which was introduced in DB2 UDB V8.2.

There has always been a requirement to take rows from one table, insert them into another table, and delete those records from the first table. Before DB2 UDB V8.2 this was a two-step process, with all the inherent dangers of that. Now, I can achieve all of this with just one SQL statement.

So let's look at an example. If you follow all the commands, you should be able to reproduce the example. All the SQL was issued on a Windows 2000 Professional system running DB2 UDB 8.2.2.

We have a table, tabts, containing the following rows:

```
create table tabts (id int, name char(10));

insert into tabts values(1,'Anita');
insert into tabts values(2,'Helen');
insert into tabts values(3,'Chantal');
insert into tabts values(4,'Fred');
insert into tabts values(5,'John');
insert into tabts values(6,'Harry');
insert into tabts values(7,'Carrie');
insert into tabts values(8,'Scott');
```

And we need to create the output table:

```
create table newtab (id int, name char(10));
```

Now we want to write an INSERT/SELECT/DELETE statement to move the lines from table tabts, where the id value is between 3 and 5 inclusive, to table newtab. The statement would look like:

```
with fred (id,name) as
(
  select id,name from old table
  (delete from db2admin.tabts where id between 3 and 5)
)
```

```
select count(*) from new table
(insert into db2admin.newtab select id,name from fred);
```

Let's look at this statement in more detail. We use the same construct as with temporary tables – the *with … as* construct. We populate this 'temporary' table with rows from our 'from' table based on a predicate where id is between 3 and 5, and then we delete these rows. We finally insert the rows from our temporary table into our 'to' table (db2admin.newtab). We need an expression after the definition of the temporary table – I have used **select count(*)** to give me some idea of how many rows I am moving. If I wanted to see those rows, I could replace the **count(*)** with **id,name**.

If I had a timestamp in my tabts table I could move rows based on the timestamp value, or if I wanted to populate my 'to' table with a 10 minute delay, then my predicate would be current timestamp -10 minutes.

I put the above SQL in a file called sel01.sql and ran it as:

```
>db2 -tvf sql01.sql

1
-----------
          3
```

You see that the SQL returns the number of rows that it moved, and if we now look at table tabts, we see that we do not have the rows with an id of 3, 4, and 5:

```
>db2 select * from tabts

ID          NAME
----------- ----------
          1 Anita
          2 Helen
          6 Harry
          7 Carrie
          8 Scott

  5 record(s) selected.
```

And if we look at table newtab, we see the three rows we have moved:

```
>db2 select * from newtab

ID          NAME
----------- ----------
          3 Chantal
          4 Fred
          5 John

  3 record(s) selected.
```

You can see that we have achieved our aim – in one SQL statement we have moved rows from one table (our 'from' table) to another table (our 'to' table) and deleted those rows from the 'from' table.

So could I have used the MERGE statement to achieve a similar result? A single MERGE statement could not have achieved both the move operation and the delete operation. We would have to write two statements – a MERGE statement and a DELETE statement. Let's use our original tables, tabts and newtab, and see what our two statements would look like. The MERGE statement would look like:

```
merge into newtab t1
using (select * from tabts where id between 3 and 5 ) t2
on (t1.id = t2.id)
when not matched then insert (id,name) values(t2.id,t2.name);
```

And the DELETE statement would look like:

```
delete from tabts where id between 3 and 5;
```

The first statement would copy the lines where the id is between 3 and 5 from our 'from' table to our 'to' table, but the rows would not be deleted – therefore we have to write our delete statement, which brings us back to our two SQL statement process. The reason we can't put the delete operation in the MERGE statement is that you can specify DELETE only for matched rows and we are copying non-matched rows.

I hope I have shown the power of the INSERT/SELECT/DELETE statement and how it has considerably improved our

processing ability over the old two-step INSERT, SELECT, and DELETE statements.

*C Leonard*
*Freelance Consultant (UK)*


# DB2 Stinger and HADR

DB2 Universal Database V8.2 (formerly 'Stinger') has a broad range of enhancements in the areas of autonomic computing, high availability, and improved performance. This article explores IBM's High Availability Disaster Recovery (HADR) and automatic client reroute, which come as another aid in enabling the 24x7 information availability and resilience required by enterprises.

## BEFORE HADR

DB2 UDB has been offering various features towards meeting the recovery and availability requirements that are essential for any critical database server. While recovery aims at avoiding any data loss through system failures, high availability is aimed at achieving 24x7 information availability, which is fast becoming a norm, rather than an exception, in our Internet-based global village.

Before HADR, typically two distinct solutions were used to address high availability and disaster recovery requirements. High availability exploited the operating system's clustering services, while database features like log shipping, data replication, shadow copying, etc, were used for disaster recovery.

## WHAT IS HADR AND AUTOMATIC CLIENT REROUTE?

DB2 UDB V8.2 for Linux, Unix, and Windows provides HADR

as a single integrated hybrid solution that addresses both availability and recovery.

DB2 HADR is yet another replication-based solution – similar to SQL replication and Q Replication – and is supposed to have borrowed heavily from technologies available in the Informix Dynamix Server.

Automatic client reroute is another availability enhancement that reroutes the client connections to the standby database when the connection to the primary database fails. Automatic client reroute is supported only with TCP/IP. To enable this feature for a specific database, the alternate server hostname and the port number should be specified in the database directory catalog.

The HADR option is included at no extra charge with DB2 ESE and is also available as an add-on product for DB2 Express, DB2 WSE, and DB2 WSUE servers

## HOW DOES HADR WORK?

DB2 UDB HADR is a database replication feature that protects against data loss by replicating the changes from the source (or primary) database to the target (or standby) database.

Applications access the primary database only and updates are made to this database both at the data structure level (using DDL) as well as the data level (using DML). The log data that is generated on the primary database is shipped to the standby database and is used to update the standby database accordingly.

Log shipping via user exits has been asynchronous and hence there was a potential for data loss when the active logs of the primary server had not been shipped successfully when a failure occurred. HADR overcomes this limitation by implementing three synchronization modes – synchronous, near synchronous, and asynchronous.

Without HADR, customers had to implement high availability

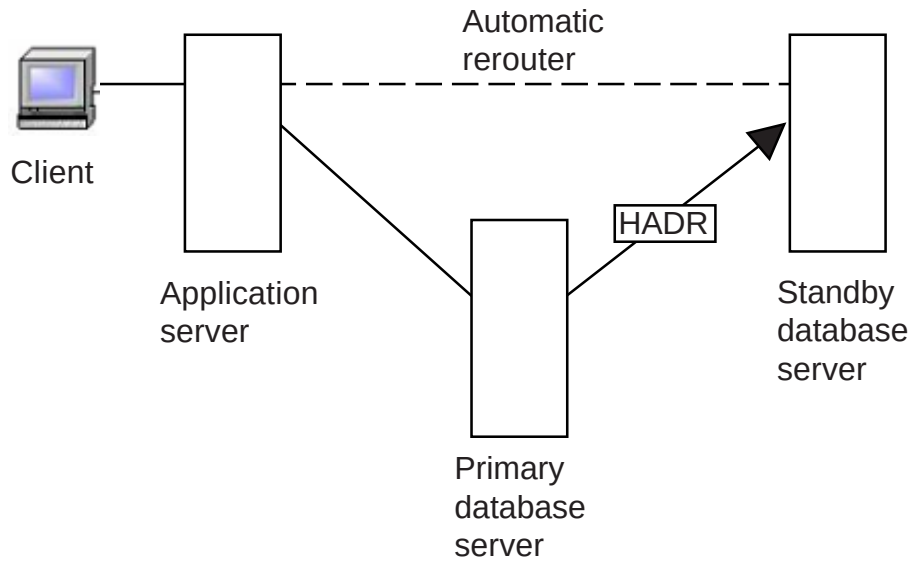


*Figure 1: DB2 with HADR and automatic reroute*

solutions based primarily on the clustering services provided by the operating systems. Now, by using the automatic client reroute feature with HADR, it can be ensured that the client application connections automatically failover to the standby server (which acts as a primary server for a certain period) and then failback to the primary server when it gets reactivated.

HADR uses TCP/IP for communicating between the primary and standby databases and hence these databases can be situated in different locations – in completely separate and independent storage. This can be used as an effective disaster recovery solution where your standby database is located in a different city or even a different country. The set-up is illustrated in Figure 1.

The *Manage HADR* interface in the DB2 Control Center lets you manage and check HADR status. This window will notify you if there are any problems with the HADR configuration.

## HADR DATA PROTECTION LEVELS

HADR offers three synchronization modes – synchronous, near-synchronous, and asynchronous – which control how log writing is managed between the primary and standby databases. This in turn provides an option for choosing the level of protection against potential data losses.

In synchronous mode, as the name suggests, the log writes are considered successful only after the primary database receives acknowledgement from the standby database that the logs have also been written to the log files on the standby database. Because the log data is guaranteed to be stored on both sides, this option provides the greatest protection. On the negative side, this mode results in the longest transaction response time.

Slightly less protection is offered by the near-synchronous mode, which also results in shorter response times. In this mode, the log writes are considered successful only after the primary database receives an acknowledgement from the standby database that the logs have also been written to main memory on the standby system. If the standby database crashes before it can copy the log records from memory, the log records will be lost. When the standby database restarts, it can get the log records again from the primary database. Data loss can occur only if both sides fail simultaneously before the standby copies all the log records from memory.

Asynchronous mode offers the least protection and also has the shortest transaction response time among the three modes. In this mode, the log write is considered successful once the log records have been delivered to the TCP layer of the primary system's host machine. The primary system doesn't wait for any acknowledgement from the standby. Log files in transit can be lost under a lot of conditions including failure of the primary database's host machine, the network, or on the standby database. If failover is needed when there are missing log files, permanent loss of transactions can occur.

## DATABASE CONFIGURATION FOR HADR

IBM recommends that the database and the database manager configuration be identical on the primary and standby databases to ensure optimal performance.

As HADR depends on log-shipping, non-logged operations like database configuration parameter changes, changes to the recovery history file etc are not replicated in the standby database. Though this restriction may seem obvious, it is essential that this key factor be kept in mind when establishing processes for maintaining both the databases.

Changes to the configuration parameters on the primary database must be replicated manually in the standby database also. While the dynamic parameters become effective immediately, the non-dynamic ones would require that the standby database be restarted.

To guarantee that the log file size of the standby is always the same as the primary one, IBM has designed the standby database to ignore the local LOGFILSIZ configuration and create the local log files to match the LOGFILSIZ configuration on the primary database.

The log receive buffer size (LOGBUFSZ) of the standby database is, by default, twice that of the value specified for the primary database. Though this size should be in general sufficient, there are times when there could be temporary peaks. With HADR in asynchronous mode and when the primary database experiences a high transaction load, the log shipping operation may stall, resulting in a larger buffer requirement on the standby side. Under such circumstances, the administrator can increase the standby log buffer size by modifying the DB2_HADR_BUF_SIZE registry variable.

The new database configuration parameters that support HADR include:

1 HADR_LOCAL_HOST – the local host name of the primary database must be same as the remote host name of the standby database.

2 HADR_LOCAL_SVC – the TCP service name or port number for which the local HADR process accepts connections.

3 HADR_REMOTE_HOST – the remote host name of the primary database must be same as the local host name of the standby database.

4. HADR_REMOTE_SVC – the TCP service name or port number that will be used by the remote HADR node.

5 HADR_SYNCMODE – HADR synchronization mode that determines how primary log writes are synchronized with the standby. Values are SYNC, NEARSYNC, and ASYNC.

6 HADR_TIMEOUT – specifies the time that the HADR process waits before considering that the communication attempt has failed.

The following are some of the key observations related to the HADR parameters:

- When the connection is established, a consistency check for the local and remote host names is performed to ensure that the remote host specified is the expected node.
- The synchronization and timeout parameters should be identical in both the primary and standby databases. When a HADR pair establishes the connection, a consistency check for this is performed.
- Usually the standby database is started first. If the primary database is started first and if the standby is not started within the HADR timeout limit specified, the start-up procedure will fail.
- Though the local host name and service name are relevant only for the primary database, it is essential that they be set appropriately in the standby database so that the standby is ready to take over when required.

- Changes made to HADR parameters are not effective until the database is shut down and restarted.

IBM has provided the set-up HADR wizard in the DB2 Control Center, which lets you easily configure primary and standby database servers with HADR.

## HADR COMMANDS

The following commands are provided to control the HADR:

- START HADR – starts HADR operations for a database. If specified as primary and with the 'NO FORCE' option, the HADR primary database will not wait for the standby database to connect to it. If started as standby, the database will attempt to connect to the primary database until the connection is successfully established.
- STOP HADR – used to convert the HADR database to a standard one. Stops HADR operations for a database. If this command is issued for an active primary database, it stops shipping logs to the standby and the database role changes to standard and remains online. This command returns errors when issued on an active standby database because it is necessary to deactivate the standby before converting it to standard.
- TAKEOVER HADR – used to switch the roles of the primary and standby databases. Instructs a HADR standby database to take over as the new HADR primary database for the HADR pair. The command can be issued only on the standby database.

## LOAD OPERATIONS AND HADR

If the load operation is executed on the primary database with the COPY YES option, the command will execute on the primary database and the data will be replicated to the standby database. It has to be ensured that the device or directory

specified in the load command can be accessed by the standby database using the same path, device, or load library. Otherwise, the standby table space in which the table is stored is marked as bad and any future log records to this tablespace get skipped.

Similarly, if the load operation is executed with the NONRECOVERABLE option, the table on the standby database is marked bad and the future log records get skipped.

Load operations with the COPY NO option specified are not supported in HADR. The load operation with the COPY NO can be automatically converted to COPY YES by setting the DB2_LOAD_COPY_NO_OVERRIDE registry in the primary database.

## CLUSTER MANAGERS AND HADR

The availability of a database can be enhanced by using HADR with the operating system cluster managers.

One way of configuring this is to set up a HADR pair such that the same cluster manager services the primary and standby databases. This configuration is most suited when the primary and standby databases are located at the same site and the fastest possible failover is required. The cluster manager can benefit from HADR to quickly detect the problem and initiate the take-over operation without waiting for failover to occur on the volume.

The other option is to set up a HADR pair where the primary and standby databases are not serviced by the same cluster manager. This is more suited to high availability under disaster recovery (in the event of complete site failure) where the databases are located at different sites.

## RESTRICTIONS WITH HADR

The following are the HADR restrictions:

1 HADR is not supported when multiple database partitions are used. In a partitioned database environment, though the HADR configuration parameters are visible and can be changed, they are ignored.

2 Reads on the standby database are not supported and hence the clients cannot connect directly to the standby.

3 Log archiving can be performed only by the primary database.

4 Back-up operations are not supported on the standby database.

5 Use of data links is not supported.

6 HADR does not interface with the DB2 Fault Monitor, which can be used to automatically restart a failed database.

7 HADR does not replicate stored procedures and UDF objects and library files. You must create the files on identical paths on both the primary and standby databases.

## RECOMMENDATIONS FOR EFFECTIVE USE OF HADR

IBM recommends the following to make HADR effective:

- Use identical computers and operating system versions including patch levels for both the primary and standby databases. This would ensure that the behaviour of the standby server during failover is as expected.
- A TCP/IP interface is a must, between the HADR host machines, while a high-speed high-capacity network is recommended.
- Use the same level of resources (hardware and software) for both the primary and standby database servers. This would ensure that there are no performance issues in the case of failover situations.
- Make sure that the database versions of the primary and

standby databases as well as the bit sizes of the database (32 or 64) are identical. During rollover upgrades, the version on the standby server can be later than that of the primary database. You should never have the primary database version higher than the standby database version.

- Allocate the same amount of space for the log files on both the database servers.
- Ensure that the table spaces type (DMS or SMS), size, container path, container size, and container file type (raw device or file system) are identical on both the databases. If they are not, the log replay may fail with OUT OF SPACE or TABLE SPACE CONTAINER NOT FOUND error conditions, making the standby table space unavailable for takeover.
- Use relative container paths, which allow the same relative path to map to different absolute container paths on the primary and standby databases.
- It is necessary that the amounts of memory are the same on both the servers, because the bufferpool operations are also replayed on the standby databases.
- For HADR databases, set the LOGINDEXBUILD parameter to ON to ensure that the index creation, recreation, or reorganization information is logged. Although this would result in a longer index build time and more log space in the primary database, the indexes will be rebuilt on the standby system during log replay and will be available when the failover takes place.

*C Sasirekha*
*Tata Consultancy Services (India)*

# Managing DB2 for z/OS through WAP and Web environments – part 2

*This month we conclude the code to manage DB2 from WAP and Web environments.*

```
Const adRecInvalid = &H0000010
Const adRecMultipleChanges = &H0000040
Const adRecPendingChanges = &H0000080
Const adRecCanceled = &H0000100
Const adRecCantRelease = &H0000400
Const adRecConcurrencyViolation = &H0000800
Const adReegrityViolation = &H0001000
Const adRecMaxChangesExceeded = &H0002000
Const adRecObjectOpen = &H0004000
Const adRecOutOfMemory = &H0008000
Const adRecPermissionDenied = &H0010000
Const adRecSchemaViolation = &H0020000
Const adRecDBDeleted = &H0040000
'---- GetRowsOptionEnum Values ----
Const adGetRowsRest = -1
'---- PositionEnum Values ----
Const adPosUnknown = -1
Const adPosBOF = -2
Const adPosEOF = -3
'---- AffectEnum Values ----
Const adAffectCurrent = 1
Const adAffectGroup = 2
Const adAffectAll = 3
'---- FilterGroupEnum Values ----
Const adFilterNone = 0
Const adFilterPendingRecords = 1
Const adFilterAffectedRecords = 2
Const adFilterFetchedRecords = 3
'---- PropertyAttributesEnum Values ----
Const adPropNotSupported = &H0000
Const adPropRequired = &H0001
Const adPropOptional = &H0002
Const adPropRead = &H0200
Const adPropWrite = &H0400
'---- ErrorValueEnum Values ----
Const adErrInvalidArgument = &Hbb9
Const adErrNoCurrentRecord = &Hbcd
Const adErrIllegalOperation = &Hc93
Const adErrInTransaction = &Hcae
Const adErrFeatureNotAvailable = &Hcb3
```

```
Const adErrItemNotFound = &Hcc1
Const adErrObjectNotSet = &Hd5c
Const adErrDataConversion = &Hd5d
Const adErrObjectClosed = &He78
Const adErrObjectOpen = &He79
Const adErrProviderNotFound = &He7a
Const adErrBoundToCommand = &He7b
'---- ParameterAttributesEnum Values ----
Const adParamSigned = &H0010
Const adParamNullable = &H0040
Const adParamLong = &H0080
'---- ParameterDirectionEnum Values ----
Const adParamUnknown = &H0000
Const adParamInput = &H0001
Const adParamOutput = &H0002
Const adParamInputOutput = &H0003
Const adParamReturnValue = &H0004
'---- CommandTypeEnum Values ----
Const adCmdUnknown = 0
Const adCmdText = &H0001
Const adCmdTable = &H0002
Const adCmdStoredProc = &H0004
%>
```

## 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=*
```

## DB2COMMA

```
/* REXX */
/* PARSE ARG SSID COMMAND  */          /* Get the SSID to connect to */
                                       /* and the DB2 command to be  */
                                       /* executed                   */
SSID = 'DBØT'
COMMAND = '-DISPLAY DDF'
   /*********************************************************************/
  /*  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 <> Ø THEN CALL SQLCA */
say 'a2'
PROC = 'COMMAND'
RESULTSIZE = 327Ø3
RESULT = LEFT(' ',RESULTSIZE,' ')
   /*********************************************************************/
  /*  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 < Ø 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.Ø',',  */
/*          || SQLWARN.1',',    */
/*          || SQLWARN.2',',    */
/*          || SQLWARN.3',',    */
/*          || SQLWARN.4',',    */
/*          || SQLWARN.5',',    */
/*          || SQLWARN.6',',    */
/*          || SQLWARN.7',',    */
/*          || SQLWARN.8',',    */
/*          || SQLWARN.9',',    */
/*          || SQLWARN.1Ø       */
/* 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 O
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.Ø',',
            || SQLWARN.1',',
            || SQLWARN.2',',
            || SQLWARN.3',',
            || SQLWARN.4',',
            || SQLWARN.5',',
            || SQLWARN.6',',
```

```
            || SQLWARN.7',',
            || SQLWARN.8',',
            || SQLWARN.9',',
            || SQLWARN.1Ø
SAY 'SQLSTATE='SQLSTATE
/* EXIT 99 */
```

## 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,5,1ØØ)
   say 'COMMAND=' || COMMAND
/* SSID = 'DBØT' */
/* 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 <> Ø THEN CALL SQLCA */
say 'a2'
PROC = 'COMMAND'
RESULTSIZE = 327Ø3
RESULT = LEFT(' ',RESULTSIZE,' ')
   /*******************************************************************/
  /*  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 < Ø 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.Ø',',  */
/*          || SQLWARN.1',',    */
/*          || SQLWARN.2',',    */
/*          || SQLWARN.3',',    */
/*          || SQLWARN.4',',    */
/*          || SQLWARN.5',',    */
/*          || SQLWARN.6',',    */
/*          || SQLWARN.7',',    */
/*          || SQLWARN.8',',    */
/*          || SQLWARN.9',',    */
/*          || SQLWARN.1Ø       */
/* 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 O
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.Ø',',
```

```
             || 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 */
```

## DB2COMME_DB2_CREATE_PROCEDURE

```
//DB2COMME JOB   ,'DB2-DYNAMIC-SQL',MSGLEVEL=(1,1),MSGCLASS=X,USER=SDBA1
//*
//STEP1    EXEC PGM=IKJEFT01,DYNAMNBR=20
//SYSTSPRT DD SYSOUT=*
//SYSPRINT DD SYSOUT=*
//SYSUDUMP DD SYSOUT=*
//SYSTSIN  DD *
 DSN SYSTEM(DB1T)
 RUN PROGRAM(DSNTEP2) PLAN(DSNTEP71) LIB('DSN710.RUNLIB.LOAD')
//SYSIN    DD *
DROP PROCEDURE SYSPROC.DB2COMME RESTRICT;
COMMIT;
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;
```

## 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="0" 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>
<input TYPE="radio" VALUE="DISPLAY DB(*) SP(*) RESTRICT LIMIT(*)"
NAME="komut"
onclick="return radioClick (document.forms[0],2)"
<%if cint(request.form("myhidden"))=2 then response.write "CHECKED"%> >
DISPLAY DB(*) SP(*) RESTRICT LIMIT(*)
</TD></TR>
<TR><TD>
<input TYPE="radio" VALUE="DISPLAY UTILITY(*)"  NAME="komut"
onclick="return radioClick (document.forms[0],3)"
<%if cint(request.form("myhidden"))=3 then response.write "CHECKED"%> >
DISPLAY UTILITY(*)
<br>
<input TYPE="radio" VALUE="TERM UTILITY(TGNLLOAD)"  NAME="komut"
onclick="return radioClick (document.forms[0],4)"
<%if cint(request.form("myhidden"))=4 then response.write "CHECKED"%> >
TERM UTILITY(TGNLLOAD)
<br>
</TD></TR>
<TR><TD>
```

```
<input TYPE="radio" VALUE="DISPLAY PROCEDURE"  NAME="komut"
onclick="return radioClick (document.forms[Ø],5)"
<%if cint(request.form("myhidden"))=5 then response.write "CHECKED"%> >
DISPLAY PROCEDURE
<br>
<input TYPE="radio" VALUE="DISPLAY DDF"  NAME="komut"
onclick="return radioClick (document.forms[Ø],6)"
<%if cint(request.form("myhidden"))=6 then response.write "CHECKED"%> >
DISPLAY DDF
<br>
<input TYPE="radio" VALUE="DISPLAY GROUP"  NAME="komut"  onclick="return
radioClick (document.forms[Ø],7)" <%if cint(request.form("myhidden"))=7
then response.write "CHECKED"%> >  DISPLAY GROUP
</TD></TR>
<TR><TD>
<input TYPE="radio" VALUE="DISPLAY THREAD(*) TYPE(*)"  NAME="komut"
onclick="return radioClick (document.forms[Ø],8)"
<%if cint(request.form("myhidden"))=8 then response.write "CHECKED"%> >
DISPLAY THREAD(*) TYPE(*)
<br>
<input TYPE="radio" VALUE="DISPLAY THREAD (*) TYPE(INDOUBT)"
NAME="komut"
onclick="return radioClick (document.forms[Ø],9)"
<%if cint(request.form("myhidden"))=9 then response.write "CHECKED"%> >
DISPLAY THREAD (*) TYPE(INDOUBT)
<br>
<input TYPE="radio" VALUE="RECOVER INDOUBT (CICSTSA1) ACTION(ABORT)
ID(ENTRTTOHØØØ1)"  NAME="komut"
onclick="return radioClick (document.forms[Ø],1Ø)"
<%if cint(request.form("myhidden"))=1Ø then response.write "CHECKED"%> >
RECOVER INDOUBT (CICSTSA1) ACTION(ABORT) ID(ENTRTTOHØØØ1)
<BR>
</TD></TR>
</TABLE>
DB2 Command that will run:
<input size="1ØØ" maxlength="1ØØ" name="mycommand1"
value="<%response.write session("aktifcommand")%>" >
<BR><BR><br>
<input type="submit" name="submit" value="RUN THE DB2 COMMAND"></p>
</p>
</form>
<|-- #include file="constans.inc" -->
<%
if cstr(request.form("submit"))="RUN THE DB2 COMMAND" then
'beginnig 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))="DBØT" 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, 1Ø4,
MYINPUT1)
set ADO_Parm2 = CMD.CreateParameter("parm2", adChar,
adParamOutput,327Ø3,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>"
response.write "<br>"
%>
<table border="Ø" width="1ØØ%" >
<TR class=tarih align=left><td>
<%
 stringtowrite = cmd.parameters(1)
 i=1
 while i<= len(stringtowrite)-1
      if mid(stringtowrite,i,1)="@" then response.write "<br>"
      if mid(stringtowrite,i,1)=" " then response.write "  "
      if mid(stringtowrite,i,1)<>"@" then response.write
mid(stringtowrite,i,1)
     i=i+1
 wend
'end of the result
 end if
  %>
  </td></tr></table>
```

```
</font>
</body>
</html>
```

## 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,$ $)
   /*****************************************************************/
  /*  Set up the IFCA, return area, and output area for the        */
  /*  IFI COMMAND call.                                            */
   /*****************************************************************/
IFCA = SUBSTR('ØØ'X,1,18Ø,'ØØ'X)
IFCA = OVERLAY(D2C(LENGTH(IFCA),2),IFCA,1+Ø)
IFCA = OVERLAY($IFCA$,IFCA,4+1)
RTRNAREASIZE = 262144 /*1Ø48572*/
RTRNAREA = D2C(RTRNAREASIZE+4,4)LEFT(' ',RTRNAREASIZE,' ')
OUTPUT = D2C(LENGTH(CMD)+4,2)||'ØØØØ'X||CMD
BUFFER = SUBSTR($ $,1,16,$ $)
   /*****************************************************************/
  /*  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,2Ø+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 <> Ø THEN CALL SQLCA
ADDRESS DSNREXX $EXECSQL PREPARE S1 FROM :SQLSTMT$
IF SQLCODE <> Ø THEN CALL SQLCA
```

```
   /*****************************************************************/
  /*  Extract messages from the return area and insert them into  */
  /*  the temporary table.                                         */
   /*****************************************************************/
SEQNO = Ø
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 <> Ø THEN CALL SQLCA
  OFFSET = OFFSET + LEN
END
   /*****************************************************************/
  /*  Set up a cursor for a result set that contains 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 <> Ø THEN CALL SQLCA
ADDRESS DSNREXX $EXECSQL PREPARE S2 FROM :SQLSTMT$
IF SQLCODE <> Ø THEN CALL SQLCA
   /*****************************************************************/
  /*  Open the cursor to return the message output result set to  */
  /*  the caller.                                                  */
   /*****************************************************************/
ADDRESS DSNREXX $EXECSQL OPEN C2$
IF SQLCODE <> Ø 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.Ø','
            || 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';'
```

## WMLDB2COMME.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>
    <do type="options" label="Copyright">
      <go href="#copyright"/>
    </do>
    <p align="center">
   <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, 1Ø4,
MYINPUT1)
set ADO_Parm2 = CMD.CreateParameter("parm2", adChar,
adParamOutput,327Ø3,myinputoutputvar)
CMD.Parameters.Append ADO_Parm1
CMD.Parameters.Append ADO_Parm2
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; 2Ø04<br/>Akbank T.A.S.<br/>All rights
reserved.</small>
    </p>
  </card>
</wml>
```

*Kadir Güray Meriç*
*DB2 Systems Programmer*
*Akbank (Turkey)*

## DB2 Web services

Web services represent the next level of function and efficiency in e-business. The Web services infrastructure is based on the eXtensible Mark-up Language (XML). Web services are a series of standards and evolving standards that are being designed and specified by the Worldwide Web Consortium (W3C) to foster cross-platform program-to-program communications. More specifically, the W3C has currently specified a template (Web Services Description Language – WSDL) and a procedure call protocol (a programmatic interface called Simple Object Access Protocol – SOAP) as 'official' Web services standards.

The Web services architecture describes a framework in which e-business services may be described, published, discovered, and invoked dynamically in a distributed computing environment. Services (which are defined as a collection of operations that carry out some type of task) are implemented and published by service providers. They are discovered and invoked by service requesters. Information about a service may be kept within a service registry.

The three fundamental operations can be described in the

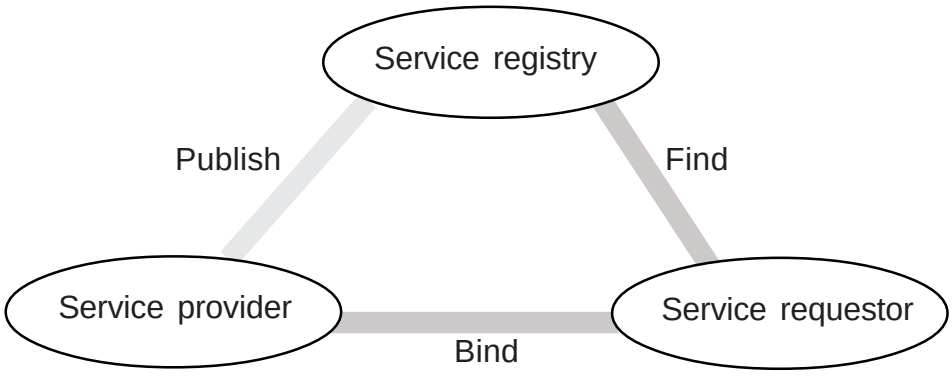


*Figure 1: Web services architecture*

following way:

- Publish – performed by the service provider to advertise the existence and capabilities of a service.
- Find – performed by the service requester to locate a service that meets a particular need or technology fingerprint.
- Bind – performed by the service requester to invoke the service being provided by the service provider.

A Web service can be defined as a modular application that can be:

- Described using Web Services Description Language (WSDL).
- Published using Uniform Description, Discovery, and Integration (UDDI).
- Found using UDDI.
- Bound using Simple Object Access Protocol (SOAP) .
- Invoked using SOAP.
- Composed with other services into new services using Web Services Flow Language (WSFL).

Web services often need to be integrated with relational data. To accomplish this, applications must access both Web services and database management systems.

## DB2 AS A WEB SERVICE PROVIDER

Web services Object Runtime Framework (WORF), which is shipped with DB2 V8.1, provides an environment to easily create simple Web services that access DB2. WORF uses Apache Simple Object Access Protocol (SOAP) 2.2 or later and the Document Access Definition Extension (DADX). A DADX document specifies how to create a Web service using a set of operations that are defined by SQL statements

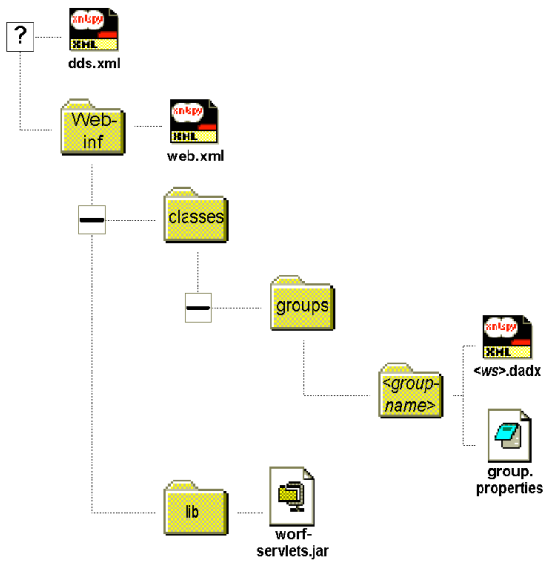


*Figure 2: WORF configuration files*

(including stored procedure calls) and, optionally, XML Extender DAD files. The Web services that are created from a DADX file are called DADX Web services or DB2 Web services.

WORF requires JDBC 2.0, which is the default in DB2 UDB Version 8. For DB2 UDB versions earlier than Version 8, select JDBC 2.0 by running the C:\Program Files\SQLLIB\java12\usejdbc2.bat file. To install WORF on WAS (WebSphere Application Server 5.0.2), complete the following steps:

1 Ensure that you have soap.jar in C:\Program Files\IBM\WebSphere\AppServer\lib.

2 Locate dxxworf.zip in your C:\Program Files\IBM\SQLLIB\samples\java\Websphere directory.

3 Unzip the dxxworf.zip to a directory, such as C:\worf.

4 Copy C:\worf\lib\worf.jar to C:\Program Files\IBM\WebSphere\AppServer\li directory.

5 Copy C:\worf\lib\worf-servlet.jar to C:\WORFSamp\WEB-

INF\lib directory (see Figure 2).

6 Start the WebSphere Administration Server.

A DADX file defines a Web service by specifying a set of operations. The definition of an operation consists of a list of parameters and an action to be performed. The action to perform is defined using SQL statements or DAD file references. DADX follows XML syntax to define the Web service. The operations in a DADX Web service can be defined by the following operation types:

- SQL operations:

  <query> – queries the database.

  <update> – performs an update, insert, or delete operation.

  <call> – calls stored procedures.

- XML collection operations (requires DB2 XML Extender):

  <retrieveXML> – generates XML documents.

  <storeXML> – stores XML documents.

For each set of related Web services (Web services that access the same database), we need to create a group and, as part of the group, we also create a connection configuration for the database (see group.properties file). The DADX file is stored in the directory defined for this group (List.dadx).

For each group we need to create a servlet (the role of this servlet is to inform the Web server at run-time where the appropriate DADX document is). This servlet is an instance of the DxxInvoker servlet that is part of worf-servlets.jar. The name of this servlet instance should be the same as the name of the Web service group, which is SampGroup in our case.

To associate the index.html file as the welcome file for the Web application insert the following lines into your web.xml file between the <webapp> and </webapp> tags:

```
<welcome-file-list>
   <welcome-file>index.html</welcome-file>
```

```
</welcome-file-list>
```

We can now proceed to package the Web application as a Web archive or war file by entering the following command. You will need to be in the WORFSamp directory:

```
c:\WORFSamp>"c:\Program Files\IBM\websphere\appserver\java\bin\jar" -cvf
SampleWorf.war *
```

Now you have created the Web application that can be deployed on an Application Server, start the WebSphere Administrative Web client by typing the following URL in the Web browser's address bar: http://localhost:9090/admin (assuming you have accepted the default option of port 9090 for the WebSphere Application Server admin server while installing Application Server and that the admin server is currently started).

Enter the system user id and login to administer your Application Server.

1 Click on the **Install New Applications** link listed under **Applications** and provide the following values on the appropriate screens:

- Local path: c:\WORFSamp\SampleWorf.war
- Context Root: SampleWorf
- Virtual Host: default_host
- Application Name: SampleWorf_war
- Reload Interval in Seconds: 0
- Put a tick in the box SampleWorf.war (Map Virtual Hosts...)
- Put a tick in the box SampleWorf.war (Map modules to application...)
- Click on the **Finish** push button
- After successfully installing the application, click on the link **Save to Master Configuration** and then click on the **Save** button.

2 Before we can start the application we need to ensure that the db2 JDBC drivers are included as part of the CLASSPATH. To do this, first click on the **Application Servers** link under **Servers** and then click on **server1**.

   On the next screen you will need to scroll down the page and click on the **Process definition** link. You will need to scroll down the next page as well and click on the **Java Virtual Machine** link. In the CLASSPATH field, enter C:\Program Files\IBM\SQLLIB\java\db2java.zip.

   Once you have done this you will need to scroll down the screen and click **OK**. This will return you to a previous screen, where you will need to click on **Save** to apply the changes to the master configuration.

3 Click on the **Enterprise Applications** link listed under **Applications**. Select the SampleWORF_war and click on the **start** button. You should be able to see that the red cross has changed to a green arrow, indicating that the Web application has started.

4 Open a new Web browser window and type the following URL in the address bar: http://localhost/SampleWorf/.

If you analyse the definition of our servlet 'SampGroup' you will see that the urspattern for this servlet is /sample/*. Hence when Application Server gets the request for /sample/List.dadx, it invokes the SampGroup servlet. This servlet instance looks for a directory called SampGroup (its own name) under /WEB-INF/classes/groups directory. In this directory it searches for List.dadx and invokes the Web service operation.

If you want to add another operation to the same Web service, all you need to do is add another operation element into the List.dadx file and save the file. WORF will automatically pick up the new operation when you invoke it. Also, if you want to create another related Web service with one or more operations in it, you can create a new DADX file similar to this one and place it under the directory in which the deployed List.dadx is placed (for example if your WAS root is C:\Program

Files\IBM\WebSphere\AppServer, this directory would be C:\Program Files\IBM\WebSphere\AppServer\installedApps\<YOUR-NODE-NAME>\SampleWORF_war.ear\SampleWORF.war\WEB-INF\classes\groups\SampGroup. You need not redeploy your Web application. WORF will pick up the new DADX automatically when the DADX is invoked).

Now look at the requirements DADX has to allow us to call a stored procedure.

We need to consider that stored procedures can return one or more result sets. You can include them in the output message. Metadata for a stored procedure result set must be defined explicitly in the DADX using the <result_set_metadata> element. At run-time, you obtain the metadata of the result set. The metadata must match the definition contained in the DADX file. Therefore, you can invoke only stored procedures that have result sets with fixed metadata. This restriction is necessary in order to have a well-defined WSDL file for the Web service. A single result set metadata definition can be referenced by several <call> operations, using the <result_set> element. The result set metadata definitions are global to the DADX and must precede all of the operation definition elements.

First look at the <result_set_metadata> element:

- Attributes:
  - name – identifies the root element for the result set.
  - Rowname – used as the element name for each row of the result set.
- Children:
  - <column> – defines the column. The order of the columns must match that of the result set returned by the stored procedure. Each column has a name, type, and nullability, which must match the result set.
  - Attributes:

    - name – required. This specifies the name of the column.
    - type – required if you do not specify element. It specifies the type of column.
    - element – required if you do not specify type. It specifies the element of the column.
    - as – optional. This provides a name for a column.
    - nullable – optional. Nullable is either true or false. It indicates whether column values can be null.

Next we can look at the <operation> element (the operation element and its children specify the name of an operation, and the type of operation the Web service performs):

- Attribute:
  - name – a unique string that identifies the operation. The string must be unique within the DADX file.
- Children (document the operation with the following element):
  - <documentation> – specifies a comment or statement about the purpose and content of the operation.
  - <call> – specifies a call to a stored procedure.
  - <SQL_call> – specifies a stored procedure call.
  - <parameter> – required when referencing a parameter in an <SQL_call> element. This specifies a parameter for an operation. Use a separate parameter element for each parameter referenced in the operation. Each parameter name must be unique within the operation.
  - Attributes:
    - name – the unique name of the parameter.
    - type – use the type attribute to specify a simple type.

- kind – specifies whether a parameter passes input data, returns output data, or does both. The valid values for this attribute are in, out, and in/out.
- <result_set> – this defines a result set and must follow any <parameter> elements. The result set element has a name that must be unique among all the parameters and result sets of the operation. It must refer to a <result_set_metadata> element. One <result_set> element must be defined for each result set returned from the stored procedure
- Attributes:
  - i name – a unique identifier for the result sets in the SOAP response.
  - ii Metadata – a result set metadata definition in the DADX file. The identifier must refer to the name of an element.

To invoke a stored procedure we need to do the following three things:

- Edit the DADX document.
- Create the stored procedure.
- Edit the index.html welcome page so that we can execute the new stored procedure.

## DB2 AS A WEB SERVICE CONSUMER

Web service consumer User-Defined Functions (UDFs) are provided as part of fix pack 2 for DB2 V8. These UDFs enable databases to directly invoke Web services using SQL. This eliminates the need to transfer data between Web services and the database, so the result is better performance. The Web services consumer converts existing WSDL interfaces into DB2 table or scalar functions.

The prerequisite is to enable DB2 XML Extender and register

five user-defined functions. From the DB2 Command Window enter:

- db2 connect to sample
- dxxadm enable_db sample
- db2enable_soap_udf -n SAMPLE -u db2admin -p db2admin.

To make sure that you enabled the database correctly ensure that you have the following UDFs defined:

- db2xml.soaphttpv (VARCHAR(256), VARCHAR(256), VARCHAR(3072))
- db2xml.soaphttpv (VARCHAR(256), VARCHAR(256), CLOB(1M))
- db2xml.soaphttpc (VARCHAR(256), VARCHAR(256), varchar(3072))
- db2xml.soaphttpc (VARCHAR(256), VARCHAR(256), CLOB(1M))
- db2xml.soaphttpcl (VARCHAR(256), VARCHAR(256), varchar(3072)).

Also ensure that db2soapudf.dll is copied to the sqllib/function directory of your DB2 installation.

The db2xml.soaphttp() is a DB2 UDF that composes a SOAP request, posts the request to the service endpoint, receives the SOAP response, and returns the content of the SOAP body. The function depends on the SOAP body being a VARCHAR() or a CLOB().

- ```
  db2xml.soaphttpv returns VARCHAR():
  db2xml.soaphttpv (endpoint_url VARCHAR(256),
                            soap_action VARCHAR(256),
                            soap_body VARCHAR(3Ø72)) | CLOB(1M))
                            RETURNS VARCHAR(3Ø72)
  ```

- ```
  db2xml.soaphttpc returns CLOB():
  db2xml.soaphttpc (endpoint_url VARCHAR(256),
  ```

```
                              soapaction VARCHAR(256),
                              soap_body VARCHAR(3072) | CLOB(1M))
                              RETURNS CLOB(1M)
```

- ```
  db2xml.soaphttpcl returns CLOB() as locator:
  db2xml.soaphttpcl(endpoint_url VARCHAR(256),
                              soapaction VARCHAR(256),
                              soap_body varchar(3072))
                              RETURNS CLOB(1M) as locator
  ```

DB2 requires the following information to build a SOAP request and receive a SOAP response:

- Service endpoint, eg http://services.xmethods.net/soap/servlet/rpcrouter.
- SOAP action URI reference (it is optional and may be null string, ie ‘’).
- XML content of SOAP body, which is name of operation with request namespace URI, encoding style, and input arguments.

## HOW TO MAP THE ELEMENTS OF WSDL TO PARAMETERS OF WEB SERVICE CONSUMER FUNCTIONS

The WSDL for the ‘Delayed Stock Quote Request’ Web service is available at http://services.xmethods.net/soap/urn:xmethods-delayed-quotes.wsdl and we will use this file.

The service section of the WSDL contains the port definition for the SOAP interface:

```
<port name='net.xmethods.services.stockquote.StockQuotePort'
binding='tns:net.xmethods.services.stockquote.StockQuoteBinding'>
      <soap:address location='http://64.124.140.30:9090/soap'/>
</port>
```

The location of *soap:address* shows the service endpoint of the Web service. This is the first parameter of the Web service consumer functions . If there are multiple ports for different bindings, you have to find the port with a SOAP binding:

```
<soap:binding transport=http://schemas.xmlsoap.org/soap/http .../>.
```
```

The binding section of the WSDL lists all the operations of the service.

```
<binding  name='net.xmethods.services.stockquote.StockQuoteBinding'
type='tns:net.xmethods.services.stockquote.StockQuotePortType'>
  <soap:binding style='rpc'transport='http://schemas.xmlsoap.org/soap/
http'/>
  <operation name='getQuote'>
<soap:operation  soapAction='urn:xmethods-delayedquotes#getQuote'/>
...
...
  </operation>
</binding>
```

The *soapAction* of the desired Web service operation is the value of the second parameter of the Web service consumer functions.

The *portType* definition provides the structure of the soap body. The *portType* may define many operations. Each operation typically contains an input and an output message element. Occasionally it may contain fault elements too.

```
<portType  name='net.xmethods.services.stockquote.StockQuotePortType'>
  <operation name='getQuote' parameterOrder='symbol'>
    <input message='tns:getQuoteRequest1'/>
    <output message='tns:getQuoteResponse1'/>
  </operation>
</portType>
```

You can construct the structure of the soap body using the operation element of the *portType*. The name of the operation becomes the top-level node. You then need to flatten out the input or output message element to get the rest of the structure. Thus in our case the *getQuote* forms the top-level element and the flattening of input message *getQuoteRequest1* yields the second level element symbol. In many cases this flattening of the input or output message may yield quite a complex structure. Thus in our case the structure of the soap body becomes:

```
<stock:getQuote  xmlns:stock="urn:xmethods-delayed-quotes"
    SOAP-ENV:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/">
   <symbol xsi:type="xsd:string"> SYMBOL name to be supplied</symbol>
</stock:getQuote>
```

This is the third parameter of the Web service consumer functions for our example. Similarly you can find out the structure of the result by flattening out the output message element. In our case it looks as follows:

```
<n:getQuoteResponse xmlns:n="urn:xmethods-delayed-quotes">
   <Result xsi:type="xsd:float">QUOTE VALUE RETURNED</Result>
</n:getQuoteResponse>
```

The list below provides hints for finding the different parameter values for the Web service consumer functions. It specifies the parameter and the Xpath to look for:

1 Service endpoint URL – /definition/port/soap:address/@location.

2 SOAP action – /definitions/binding/soap:binding/operation/soap:operation/@soapAction.

3 SOAP body – /definitions/portType/operation.

So, the Web service consumer function for accessing the stock quote of IBM is:

```
DB2 VALUES  db2xml.soaphttpv (
'http://64.124.140.30:9090/soap',
'urn:xmethods-delayed-quotes#getQuote',
varchar ('<stock:getQuote xmlns:stock="urn:xmethods-delayed-quotes"
SOAP-ENV:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/">
<symbol xsi:type="xsd:string">IBM</symbol>
</stock:getQuote>' ));
```

If you are running DB2 UDB Version 8.1 you may as well use SQL/XML query to construct the SOAP body. The following example is the same as the previous one except that it is written in SQL/XML and uses the overloaded soaphttpv function that takes the soap body as a CLOB:

```
DB2 VALUES  db2xml.soaphttpv (
'http://64.124.140.30:9090/soap',
'urn:xmethods-delayed-quotes#getQuote',
xml2clob(XMLElement(name "getQuote",
XMLAttributes('urn:xmethods-delayed-quotes' AS "xmlns:stock"),
XMLElement(name "symbol",'IBM'))));
```

Instead of calling the db2xml.sopahttpv(...) every time we

make a stock quote request, we create a wrapper UDF that takes only the stock symbol as a parameter and internally calls the db2xml.soaphttpv(...). Let us define our wrapper UDF getStockQuote, which takes only one parameter, namely symbol, as a VARCHAR(256) and returns *Result* as FLOAT. The result of the call to db2xml.soaphttpv is a CLOB. Hence we extract the value contained in the Result element pointed to by the Xpath '/*/Result'.

```
CREATE FUNCTION db2admin.getStockQuote (symbol VARCHAR(100))
RETURNS DECIMAL(5,2) SPECIFIC xmethods_getQuote
LANGUAGE SQL CONTAINS SQL
EXTERNAL ACTION NOT DETERMINISTIC
RETURN
db2xml.extractREAL(
db2xml.xmlclob(
db2xml.soaphttpv(
'http://64.124.140.30:9090/soap',
'urn:xmethods-delayed-quotes#getQuote',
varchar('<m:getQuote  xmlns:m="urn:xmethods-delayed-quotes"
SOAP-ENV:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/">
<symbol xsi:type="xsd:string">' || symbol || '</symbol> </m:getQuote>'
))), '/*/Result');
```

This UDF can be invoked in the DB2 command line as follows:

```
DB2 VALUES db2admin.getStockQuote('IBM')
```

DB2 also allows the writing of UDFs that invoke Web services returning multiple results. In such cases we return a table with each row containing one result element.

In all the examples above we have used a text editor to create the DADX, XML, HTML, and properties files. There are, however, tools that can be used to make these tasks a little easier, namely WebSphere Studio Application Developer (WSAD).

WSAD can create the DADX files for you as well as check the WSDL to verify the external Web services. It can also be used to create the wrapper UDF. Also, it is worth noting that currently DB2 Web services are supported only with WebSphere Application Server and the Apache Tomcat Application Server.

Why choose DB2 Web services? You will be able to:

- Integrate data between two organizations without having to develop complex applications.
- Access a Web service from SQL.
- Consolidate Web services output in a DB2 database.
- Access Web services from any programming language that supports SQL – for example C, PL/I, COBOL.

## SOURCE CODE

### Table.txt (sample data for import in demo table)

```
"DB2",105,2001-07-01,"Nikola Lazovic","Simplifying Occasional, Regular
and Periodic Tasks of Database Administrator","One of main requirements
at our installation is full availability of data to our customers at
24x7 schedule.
 Under these conditions 'maintenance window' is really narrow and is
limited to the periods of the lowest system activity..."
"DB2",118,2002-08-01,"Nikola Lazovic","DB2 for OS390 V5 and Federated
Systems","A DB2 federated system is a special type of database
management system (DBMS) that provides heterogeneous distributed query
capability. With DB2 UDB V7.2 and separate product DB2 Relational
Connect, it is possible with single SQL query to access data located on
different platforms, both IBM and non-IBM. Distributed query can
reference data from multiple sources such as DB2 family database, OLE DB
source, Oracle, Sybase and/or Microsoft SQL Server database..."
"DB2",119,2002-09-01,"Nikola Lazovic","DB2 Recovery Log - Detailed
Analysis of User Update Activities","Our computer system serves large
number of customers, so online transactions triggered from a wide
network of classic terminals, Internet, and devices like answering
machines and ATMs, are processed round-the-clock, and the large-scale
batch work is carried out as well. In this environment there are times
when the need arises to find out who and when changed some concrete
data..."
"CICS",196,2002-03-01,"Nikola Lazovic","Maintenance of CICS DB2 entries
and transactions","At our installation we have CICS Transaction Server
for OS/390 V 1.2 and DB2 V 5.1. We have developed tool for
administration of DB2 entries and transactions. The REXX execs store
information from CSD file into DB2 tables, prepare job for migration
purposes and use generated ISPF tables allowing online update of CICS
resource definitions..."
```

## Table.sql

```
-- Before proceeding, you should:
-- Start the database manager (with the db2start command).
-- Create the "sample" database (with the db2sampl command).
-- This file can be invoked from Command Window with the following
command:
-- db2 -tvf Table.sql
CONNECT TO SAMPLE;
CREATE TABLE DB2ADMIN.XEPHON ( JOURNAL CHARACTER (1Ø)  NOT NULL ,
JOURNAL_NO
SMALLINT  NOT NULL , JOURNAL_DATE DATE  NOT NULL , AUTHOR CHARACTER (3Ø)
NOT
 NULL , TITLE CHARACTER (1ØØ)  NOT NULL , ABSTRACT VARCHAR (8ØØ)  NOT
NULL   ) ;
COMMENT ON TABLE DB2ADMIN.XEPHON IS 'Test table for web service';
COMMENT ON DB2ADMIN.XEPHON ( AUTHOR IS 'Contributor name', JOURNAL IS
'Kind of
 journal (DB2, CICS, ...)', JOURNAL_DATE IS 'When journal was published
(mm-Ø1-
yyyy)', ABSTRACT IS 'Abstract of article', TITLE IS 'Title of article',
JOURNAL_NO IS 'Journal number' ) ;
GRANT SELECT,INSERT,UPDATE,DELETE ON TABLE DB2ADMIN.XEPHON TO PUBLIC;
IMPORT FROM Table.txt OF DEL MODIFIED BY CHARDEL"" COLDEL, DATESISO
DECPT.
METHOD P (1, 2, 3, 4, 5, 6) MESSAGES Table.msg INSERT INTO
DB2ADMIN.XEPHON
(JOURNAL, JOURNAL_NO, JOURNAL_DATE, AUTHOR, TITLE, ABSTRACT);
CONNECT RESET;
```

## listauthor.sql

```
-- This file can be invoked from Command Window with the following
commands:
-- db2 connect to sample
-- db2 -td@ -vf listauthor.sql
-- db2 connect reset
CREATE PROCEDURE DB2ADMIN.listauthor ( IN AUTHOR CHAR(3Ø) )
SPECIFIC DB2ADMIN.listauthor
DYNAMIC RESULT SETS 1
LANGUAGE SQL
------------------------------------------------------------------------
-- SQL Stored Procedure
------------------------------------------------------------------------
P1: BEGIN
-- Declare cursor
DECLARE cursor1 CURSOR WITH RETURN FOR
SELECT JOURNAL, JOURNAL_NO, JOURNAL_DATE, TITLE, ABSTRACT
FROM DB2ADMIN.XEPHON
WHERE AUTHOR = listauthor.AUTHOR
```

```
ORDER BY JOURNAL ASC, JOURNAL_NO ASC;
-- Cursor left open for client application
OPEN cursor1;
END P1 @
```

## index.html

```
<! ********************************************************* >
<! DB2 as a Web Service provider - welcome page              >
<!    Folder: C:\WORFSamp                                    >
<! ********************************************************* >
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">
<html>
<head>
<META http-equiv="Content-Type" content="text/html; charset=windows-
1250">
<META name="GENERATOR" content="IBM WebSphere Studio Homepage Builder
V6.0.0 Trial for Windows">
<META http-equiv="Content-Style-Type" content="text/css">
<title> DB2 as a Web Service provider </title>
</head>
<body>
<H1 align="center"> DB2 as a Web service provider </H1>
<HR width="100%">
<p>
<br>
<B><FONT color="#000099">1. Test the "List Xephon" Web Service</FONT></
B> &gt;
<a href="sample/List.dadx/listXephon"> list_Xephon</a>
<br>
<FORM method="post" action="sample/List.dadx/insertXephon">
  <TABLE border="1">
      <TBODY>
        <TR>
         <TD colspan=6><FONT color="#000099"><B>2. Test the &quot;Insert
Xephon&quot; Web Service</B></FONT><BR>
           (enter a Journal, Journal number, Journal
           date, Author, Title and Abstract and click the <B>Execute</B>
                  button)
    </TD>
        </TR>
        <TR>
          <TD>Journal</TD>
          <TD>Journal number</TD>
          <TD>Journal date</TD>
          <TD>Author</TD>
          <TD>Title</TD>
          <TD>Abstract</TD>
        </TR>
        <TR>
```

```
<TD height="25" valign="top"><INPUT type="text" name="journal" size="10"
                  maxlength="10" value="DB2"></TD>
      <TD height="25" valign="top"><INPUT type="text" name="journalno"
size="5"
                  maxlength="5" value="666"></TD>
      <TD height="25" valign="top"><INPUT type="text" name="journaldate"
                  size="10" maxlength="10" value="2005-01-01"></TD>
 <TD height="25" valign="top"><INPUT type="text" name="author" size="30"
                  maxlength="30" value="Nikola Lazovic"></TD>
      <TD height="25" valign="top"><TEXTAREA rows="2" cols="30"
name="title">DB2
                  Web Services</TEXTAREA></TD>
      <TD height="25" valign="top"><TEXTAREA rows="5" cols="50"
       name="abstract">Web services represent the next level of function
        and efficiency in e-business. The Web services infrastructure is
       based on the eXtensible Markup Language (XML). Web services are a
      series of standards and evolving standards that are being designed
and specified by the Worldwide Web Consortium (W3C) to foster
cross-platform program-to-program communications. More pecifically,
the W3C has currently specified a template (Web Services
Description Language - WSDL) and a procedure call protocol (a
programmatic interface called Simple Object Access Protocol - SOAP)
as 'official' Web services standards...
</TEXTAREA></TD>
    </TR>
        <TR>
          <TD height="35"><INPUT type="submit" name="Submit"
                 value="Execute"><INPUT type="reset" value="Reset"></TD>
        </TR>
       </TBODY>
  </TABLE>
</FORM>
<FORM method="post" action="sample/List.dadx/ListAuthor">
  <TABLE border="1">
      <TBODY>
        <TR>
    <TD><FONT color="#000099"><B>3. Test the &quot;List Author&quot; Web
         Service</B></FONT><BR>    (enter a name of a
                  contributor and click the <B>Execute</B> button)</TD>
        </TR>
        <TR>
          <TD height="25"><INPUT type="text" name="author" size="30"
                  maxlength="30" value="Nikola Lazovic"></TD>
        </TR>
        <TR>
          <TD height="35"><INPUT type="submit" name="Submit"
                 value="Execute"><INPUT type="reset" value="Reset"></TD>
        </TR>
      </TBODY>
  </TABLE>
```

```
</FORM>
<FORM method="post" action="sample/List.dadx/updateXephon">
    <TABLE border="1">
      <TBODY>
        <TR>
         <TD colspan=4><FONT color="#000099"><B>4. Test the &quot;Update
                  Xephon&quot; Web Service</B></FONT><BR>
    (enter a Abstract, a Journal, a Journal
number, a Author and click the <B>Execute</B> button)</TD>
        </TR>
        <TR>
          <TD>Abstract</TD>
          <TD>Journal</TD>
          <TD>Journal number</TD>
          <TD>Author</TD>
        </TR>
        <TR>
          <TD height="25" valign="top"><TEXTAREA rows="5" cols="50"
       name="abstract">Web services represent the next level of function
       and efficiency in e-business. The Web services infrastructure is
       based on the eXtensible Markup Language (XML). Web services are a
      series of standards and evolving standards that are being designed
           and specified by the Worldwide Web Consortium (W3C) to foster
           cross-platform program-to-program communications. More
          specifically, the W3C has currently specified a template (Web
     Services Description Language - WSDL) and a procedure call protocol
        (a programmatic interface called Simple Object Access Protocol -
                  SOAP) as 'official' Web services standards...
</TEXTAREA></TD>
          <TD height="25" valign="top"><INPUT type="text" name="journal"
                  size="10" maxlength="10" value="DB2"></TD>
        <TD height="25" valign="top"><INPUT type="text" name="journalno"
                  size="5" maxlength="5" value="666"></TD>
          <TD height="25" valign="top"><INPUT type="text" name="author"
 size="30" maxlength="30" value="Nikola Lazovic"></TD>
      </TR>
      <TR>
        <TD height="35"><INPUT type="submit" name="Submit"
                 value="Execute"><INPUT type="reset" value="Reset"></TD>
      </TR>
    </TBODY>
  </TABLE>
</FORM>
<FORM method="post" action="sample/List.dadx/deleteXephon">
    <TABLE border="1">
      <TBODY>
        <TR>
         <TD colspan=3><FONT color="#000099"><B>5. Test the &quot;Delete
                  Xephon&quot; Web Service</B></FONT><BR>
           (enter a Journal, a Journal number, a
```

```
Author and click the <B>Execute</B> button)</TD>
  </TR>
  <TR>
    <TD>Journal</TD>
    <TD>Journal number</TD>
    <TD>Author</TD>
  </TR>
  <TR>
    <TD height="25"><INPUT type="text" name="journal" size="10"
            maxlength="10" value="DB2"></TD>
    <TD height="25"><INPUT type="text" name="journalno" size="5"
            maxlength="5" value="666"></TD>
```

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

*Nikola Lazovic*
*DB2 System Administrator*
*Postal Savings Bank (Serbia and Montenegro)*

IBM has introduced DB2 Anonymous Resolution, which allows corporate users to share information with each other and government agencies without having to reveal private details. The technology is designed to address the broad range of security problems involved in handling personal information in markets such as health care, financial services, and national security.

The technology is an extension of IBM's analytics software, which uses irreversible digital signatures and other techniques for correlating the data while it remains in an "anonymized" form. This enhances privacy and prevents data from being observed in its original form.

For further information contact:
URL: www-306.ibm.com/software/data/db2/eas/anonymous.

* * *

AdventNet has announced an update for its Web applications management software, ManageEngine Applications Manager, which provides support for DB2 monitoring and JBoss 4 monitoring.

Applications Manager is Web application management software providing integrated application, server, and systems monitoring. It provides in-depth monitoring of application servers (WebLogic, WebSphere, JBoss, Tomcat), databases (DB2, SQL Server, Oracle, MySQL), custom Java, JMX, J2EE applications, and Web sites with fault management and notification capabilities.

Applications Manager is available in three editions: a free edition, which can manage ten applications; a professional edition, which helps small and medium sized businesses; and an enterprise edition with an unlimited users pack.

For further information contact:
URL: manageengine.adventnet.com/products/applications_manager/download.html.

* * *

HiT Software has announced Version 4.1 of Allora, its bidirectional database to XML transformation engine.

New in Allora 4.1 is a multiple SELECT feature. Moving on from XML transformations using a single SQL query, it now offers an option to work with multiple sub-maps that are then joined in real-time with XSL, a W3C language for transforming XML documents.

Allora 4.1 is certified to work with over 20 different databases, including DB2

For further information contact:
URL: www.hitsw.com/products_services/xmlplatform.html.

* * *

SoftTree Technologies has released Version 2.5 of DB Mail for DB2, Oracle, Microsoft SQL Server, Sybase ASE, and Sybase ASA databases.

DB Mail automates common e-business functions and provides a unified messaging platform for sending e-mail, voice, fax, network, and SMS messages. It enables messages to be sent directly from a database system.

For further information contact:
URL: www.softtreetech.com.

* * *