Configuring session routines sysdbopen() and sysdbclose()

Chiranjeevi Vishnu Saran Sudha (chisudha@in.ibm.com), System Software Engineer, IBM
Harshavardhan Changappa (vardhan.harsha@in.ibm.com), System Software Engineer, IBM
Ravi Vijay (ravvijay@in.ibm.com), Staff Software Engineer, IBM

This feature introduces new built-in SPL procedures: sysdbopen() and sysdbclose(), which are executed automatically when a user connects to or disconnects from the database. These procedures are very useful in setting the session environment variables and performing tasks, such as activating a role for users of Information Management applications whose code cannot easily be modified. They are also helpful in automating operations that need to be performed after the application terminates mainly cleaning up the operations. You can include valid SQL or SPL language statements that are appropriate when a database is opened or closed. The general restrictions on SQL statements that are valid in SPL procedures also apply to these routines.

Introduction

A very common requirement for a database server is to allow the administrator to add hidden actions at the beginning and the end of a user connection in the database server. These actions could be related to default settings for the user, such as roles, isolation levels, memory settings, optimizer hints, or tracking activities. Similar activities are sometimes required at the end of a user session, such as deleting session specific tables or tracking accounting information. IBM Informix Dynamic Server (IDS) 11.10 provides two stored procedures sysdbopen() and sysdbclose() to serve these purposes.

Using sysdbopen() and sysdbclose() procedures

The sysdbopen() procedure is executed whenever users successfully issue the DATABASE or CONNECT statement to explicitly connect to a database where the procedures are installed. (But when a user who is connected to the local database calls a remote UDR or performs a distributed DML operation that references a remote database object by using the database:object or database@server:object notation, no sysdbopen() procedure is invoked in the remote database.)

These procedures are exceptions to the general rule that IDS ignores the name of the owner of a UDR when a routine is invoked in a database that is not ANSI-compliant. For UDRs other than sysdbopen and sysdbclose, multiple versions of UDRs that have the same SQL identifier, but that have different owner names, cannot be registered in the same database unless the CREATE DATABASE statement that created the database also includes the WITH LOG MODE ANSI keywords.

You can also create the sysdbclose SPL procedure, which is executed when a user issues the CLOSE DATABASE or DISCONNECT statement to disconnect from the database. If a PUBLIC.sysdbclose procedure is registered in the database, and no user.sysdbclose procedure is registered for the current user, then the PUBLIC.sysdbclose procedure is executed automatically when that user disconnects from the database.
Warning: If a sysdbclose() procedure fails, the failure is ignored. If a sysdbopen() procedure fails, however, the database cannot be opened.

- Sysdbopen() is automatically executed at connection time for a user
- Sysdbclose() is automatically executed at connection close time for a user

The database administrator can create these well known procedures within a database:

- public.sysdbopen
- <user>.sysdbopen
- public.sysdbclose

- <user>.sysdbclose

- <user>.sysdbopen is executed automatically each time user connects to data-base using DATABASE/CONNECT TO.
- <user>.sysdbclose is executed automatically each time user disconnects from database using CLOSE DATABASE/DISCONNECT.
- sysdbclose is executed on session exit even if it does not have an explicit CLOSE DATABASE/DISCONNECT statement
    - Example: User hits ^C at the dbaccess prompt to exit.

Creating sysdbopen() and sysdbclose()

These procedures are regular SPL procedures but with a few particularities:

- Only the database administrator can create or alter these procedures for public or user on the appropriate database(s).
- Cannot have arguments or return values.
- A failure during execution of sysdbopen() closes the current database and returns the error to the user thereby preventing connection to the database.
    - Example: 310 table already exists
    - Only the DBA can bypass the execution by setting the environment variable IFX_NODBPROC to 1 (works at session level without server recycle)
- Failures from sysdbclose() are ignored.
- Only the database administrator can drop these procedures.
- The owner has more meaning than usual. If you create a procedure called myuser.sysdbopen, only "myuser" will run it when he connects to the database.
- If you create a procedure called public.sysdbopen, then all users that do not have a matching user.sysdbopen procedure, will run public.sysdbopen

Setting up a sysdbopen() and sysdbclose() procedure to configure session

1. Set the IFX_NODBPROC environment variable to any value, including 0, to cause the database server to bypass and prevent the execution of the sysdbopen() or sysdbclose() procedure.
2. Write the CREATE PROCEDURE or CREATE PROCEDURE FROM statement to define the procedure for a particular user or the PUBLIC group.
3. Test the procedure, for example, by using sysdbclose() in an EXECUTE PROCEDURE statement.
4. Unset the IFX_NODBPROC environment variable to enable the database server to run the sysdbopen() or sysdbclose() procedure.

Uses of sysdbopen() and sysdbclose()

- Can be used to do some kind of logging for connects and disconnects
- Can be used to change the isolation level when changing the application code is not an option
- Can be used to change the LOCK MODE in the same situations as above
- Can be used to stop the creation of new database sessions (inhibit connections during a maintenance period)
- Can be used to restrict certain users from connecting at certain hours, or from certain hosts
- Sysdbclose() can be used to gather session statistics (and save them in some history table)

Difference between regular UDRs and the sysdbopen() and

sysdbclose() procedures

- Regular UDRs can have arguments and return value(s) but sysdbopen and sysdbclose cannot have arguments or return values.
- With regular UDRs, any user with resource privilege on the database can create a routine, but on sysdbopen and sysdbclose only the database administrator can create these procedures.
- Sysdbopen(), sysdbclose(), and regular UDRs can contain any SET, DDL, or DML statements.
- Sysdbopen(), sysdbclose(), and regular UDRs cannot contain DATABASE, CONNECT, DISCONNECT statements.
- Objects created in procedure are owned by the procedure owner unless qualified with owner name for sysdbopen(), sysdbclose(), and regular UDRs
- With regular UDRs, almost all SET commands (except for PDQPRIORITY and OPTCOMPIND) within the UDR are persistent until the end of the session or until they are changed explicitly. If PDQPRIORITY and OPTCOMPIND are changed within the UDR, they are restored back after the procedure execution to the values at the start of the procedure execution. With sysdbopen() and sysdbclose(), all SET commands including PDQPRIORITY and OPTCOMPIND are made persistent.

Examples of sysdbopen() and sysdbclose()

The following example provides you with a general concept of how to use the sysdbopen() and sysdbclose() procedures. By setting the PDQPRIORITY, query scan can be parallelized.

```
Create procedure sysdbopen :
==========================

Database ksrtc_data;
Database selected.

> create procedure user1.sysdbopen()
SET PDQPRIORITY 80;
SET EXPLAIN FIL> > E TO "/tmp/sqexplain.out";
end procedure;
>
Routine created.

informix@ubuntu.in.ibm.com >dbaccess -e ksrtc_data pdqpriority.sql

Database selected.

database ksrtc_data;
Database closed.


Database selected.


create table customers (cust_no int, last_name char(10))
fragment by expression cust_no > 0 and cust_no < 5000 In dbspace1,
cust_no >=5000 and cust_no <10000 in dbspace2,
cust_no > 10000 and cust_no <=10200 in dbspace3,
REMAINDER IN dbspace4;
Table created.


insert into customers values (100,"Vardhan");
1 row(s) inserted.


insert into customers values (200,"Vijay");
1 row(s) inserted.


insert into customers values (5400,"Vishnu");
```

```
1 row(s) inserted.


insert into customers values (5700,"Sriram");
1 row(s) inserted.


insert into customers values (11000,"Jeevs");
1 row(s) inserted.


insert into customers values (14000,"Rams");
1 row(s) inserted.


grant resource to 'user1';
Permission granted.


grant select on customers to 'user1';
Permission granted.


close database;
Database closed.


Now a user 'user1' Queries the database ksrtc_data for listing customers,
whose cust_no is greater than 6212 from a table 'customers'
================================================================================
user1@ubuntu.in.ibm.com>dbaccess -e ksrtc_data pdq.sql

Database selected.

select * from customers where cust_no < 6212;

    cust_no last_name

       5400 Vishnu
       5700 Sriram
        100 Vardhan
        200 Vijay

4 row(s) retrieved.



Database closed.

Sqexplain file :
===============
> cat /tmp/sqexplain.out

QUERY: (OPTIMIZATION TIMESTAMP: 05-18-2010 22:32:28)
------
select * from customers where cust_no < 6212

Estimated Cost: 2
Estimated # of Rows Returned: 1
Maximum Threads: 3

  1) informix.customers: SEQUENTIAL SCAN  (Parallel, fragments: 0, 1, 3)
        Fragments Scanned: (0) dbspace1, (1) dbspace2, (3) dbspace4

        Filters: informix.customers.cust_no < 6212


Query statistics:
-----------------

  Table map :
  ----------------------------
  Internal name     Table name
  ----------------------------
  t1                customers
```

```
  type     table  rows_prod  est_rows  rows_scan  time       est_cost
  ---------------------------------------------------------------------
  scan     t1     4          1         6          00:00.00   2
```

The following example provides you with a general concept of how to use sysdbopen and sysdbclose for tracking user activities. Sysdbopen and sysdbclose for accounting.

```
/* database has the following table
create table tracking (
  id serial(100),
  username char(30),
  logintime datetime year to second,
  logouttime datetime year to second
)

/* create the following sysdbopen sp for user1 */
CREATE PROCEDURE "user1".sysdbopen()
  DEFINE GLOBAL sessionid int DEFAULT 10;
  DEFINE GLOBAL user char(30) DEFAULT “informix”;
  DEFINE global tracking int default 1;
-- Tracking
  let user="";
  let sessionid =0;
  SELECT dbinfo("sessionid") INTO sessionid FROM systables WHERE tabid=1;
  SELECT username INTO user FROM sysmaster@demo_on:syssessions
  WHERE sid=sessionid;
  INSERT INTO tracking VALUES ( 0,user , current, NULL ) ;
-- get the serial value from the last insert
  SELECT dbinfo("sqlca.sqlerrd1") INTO tracking FROM systables WHERE tabid=1;
END PROCEDURE;

/* create the following sysdbclose sp for user1 */

  CREATE PROCEDURE "user1".sysdbclose()
  DEFINE global tracking int default 1;
  UPDATE tracking SET logouttime=current
  WHERE id=tracking;
END PROCEDURE;

Output in the tracking table after disconnect :
select * from tracking;

id          100
username    user1
logintime   2008-06-24 09:00:56
logouttime 2008-06-24 09:10:24

1 row(s) retrieved.
```

The following procedure sets the role and the PDQ priority for a specific user.

```
create procedure oltp_user.sysdbopen()
  set role to oltp;
  set pdqpriority 5;
end procedure;
```

The following procedure sets the role and the PDQ priority for the PUBLIC group.

```
create procedure public.sysdbopen()
  set role to others;
  set pdqpriority 1;
end procedure
```

The following procedure can be used for debugging.

```
create procedure public.sysdbopen()
          SET DEBUG FILE TO "/tmp/tracefile" WITH APPEND;
          TRACE '--In sysdbopen';
          TRACE ON;
end procedure;
```

The following procedure gathers session statistics using sysdbclose.

```
create table user_session ( sess_id int, user_name char(32), user_id smallint,
process_id integer, host_name char(15), time_connected integer);

create procedure public.sysdbclose()
insert into user_session select  sid,
        username,
        uid,
        pid,
        hostname,
        connected
from    sysmaster:syssessions;
end procedure
```

The following example uses an external UDR.

```
create procedure public.sysdbopen()
external name ‘sysdbopen.udr language c;


#include <milib.h>

void
sysdbopen()
{
mi_string *name  = NULL;
mi_string *stmt1 = "insert into t1 values (1);";

    MI_CONNECTION *conn;        /* connection handle used in this routine */
    mi_integer    res;          /* result of all the mi_** routines       */
    mi_integer    ret;
    MI_STATEMENT  *stmt_hdl;    /* statement handle                       */
    ret = 1548;

    conn = mi_open(NULL, NULL, NULL);

    if ((stmt_hdl = mi_prepare(conn, stmt1 , name)) == NULL)
        ret = -1;
    res = mi_exec_prepared_statement(stmt_hdl, MI_SEND_READ, 0,0,
              NULL, 0, 0, NULL,  0, NULL)) == MI_ERROR)
        ret = -2;

    if ((res = mi_drop_prepared_statement(stmt_hdl)) == MI_ERROR)
        ret = -3;

    mi_close(conn);
}
```

The following procedure can be used for auditing.

```
create procedure db2inst1.sysdbopen()
system("onaudit -l 1");
system("onaudit -e 0");
system("onaudit -p /opt/IBM/informix/demo/server");
system("onaudit -s 50000");
system("onaudit -n");
system("onaudit -a -u db2inst1 -e +ACTB,GRTB,UPRW");
end procedure;
```

The following procedure can be used for restricting the number of user sessions.

```
CREATE PROCEDURE public.sysdbopen()

DEFINE login_sid INTEGER; -- Session ID
DEFINE login_curr_sessions INTEGER;

LET login_sid = DBINFO('sessionid');

SELECT count(*)
INTO login_curr_sessions
FROM sysmaster:syssessions t1, sysmaster:syssessions t2
WHERE t1.sid = login_sid
AND t2.sid != login_sid
AND t1.username = t2.username
AND t1.uid = t2.uid
AND t1.hostname = t2.hostname;


IF login_curr_sessions > 10 THEN
RAISE EXCEPTION -746,0,
'Too many coonections with the same user. Access Denied, try after some time.';
END IF;

END PROCEDURE;
```

Conclusion

The Informix Dynamic Server is a powerful database server equipped with many powerful features — built-in SPL procedures: sysdbopen( ) and sysdbclose( ) is one such feature. The article has explained the need for these procedures and provides the working examples .These procedures are very useful in setting the session environment variables and performing tasks, such as activating a role for users of Information Management applications whose code cannot easily be modified. These procedures are also helpful in automating operations that need to be performed after the application terminates mainly cleaning up the operations.

Downloads

Description	Name	Size	Download method
Sample Perl scripts for this article	DoNotLeaveThisLink.zip	10KB	**HTTP**
A related PDF (not of the article)[1]	DoNotLeaveThisLink.pdf	50KB	**HTTP**

More downloads

- Demo: How to code a widget

- Presentation: Why code widgets instead of whatnots[2]

Notes

1. This is a sample note about the PDF.
2. This is a sample note about the presentation.

Resources

Learn

- Get help from IBM Informix Dynamic Server v11.50 Information Center
- Find articles and tutorials, and connect to other resources to expand your Informix skill from developerWorks Informix page

Get products and technologies

- Informix Dynamic Server: Download a free trial version of IDS.

Discuss

- Participate in the discussion forum.
- Check out developerWorks blogs and get involved in the developerWorks community.

About the authors

Vishnu Saran is a System Software Engineer at IBM India Software Labs. He works for the IDS Integration Team across various IDS Products and also works on the OpenAdmin Tool for IDS 11. He is certified in System Administration for IBM Informix Dynamic Server V11.

Harshavardhan Changappa works for the IDS Integration Team at IBM India Software Labs. He is involved in developing automated test cases for various components of IDS like SQL, Security, and Replication. Currently, he is involved in Continuous availability features, Mach11 and ER. He is certified in System Administration for IBM Informix Dynamic Server V11.

Ravi Vijay works for Informix Development Team at IBM India Software Labs. He is involved with the development of features of Informix Dynamic Server and Informix Extended Parallel Server. Currently, he is involved with the development of warehousing features of IDS. He also worked to develop automated test cases for various features.