Triton CONSULTING

DB2 CLP on z/OS

By James Gill

Table of Contents

Introduction

Command Line Processors (CLPs) have been a necessary feature of mid-range databases forever. Long before that smart GUI got written, back when the database was being developed in a basement somewhere, the most rudimentary interface to operate and interact with the product was something that could be driven from the command line.

Later it emerged that having something that you could run scripts of DDL and SQL with was quite handy. It meant that there was a form of easy repeatability available, which can be very handy if you have 50 copies of a large database to deploy.

Whilst we don't have these challenges on DB2 for z/OS – sample tools such as DSNTEP2 and DSNTIAUL help – there is a missing link in the form of Unix System Services (USS) and DB2 interfacing.

Enter the DB2 CLP, which resides in Unix System Services. Available since DB2 for z/OS V9, this tool is one of the more useful Java based offerings.

Why Should You Care?

Have you ever wanted to be able to drive a stored procedure from and see the results without having to write an application? Or describe a table like you can in DB2 LUW? Do you want to release the power of Unix shell scripting and make use of tool like awk and sed?

You can do all of this with the DB2 CLP – and the best thing is, you already have it – it's part of DB2.

Anyway, before we get into what we can do with it, we need to know how to use Unix System Services (USS).

How Do I..?

Access Unix System Services

There are a number of hurdles to jump over before you can start living the Unix dream on z/OS, and most of them are to do with security. Within RACF, this means that you need the administrator to setup an OMVS segment (OMVS is the old name for USS). This is made up of a (preferably) unique numeric ID, which is used with file permissions, a "home" directory (usually of the form /u/userid) and which shell (think of this as your general USS command line processor) – usually /bin/sh.

The other main hurdles can really be covered by "Do I know what I'm doing in Unix?" (be honest), and "How do I get there?". The first of these is not really in the scope of this blog, but a long time ago I found Linux very handy as a playpen and learning platform. If you set up a virtual machine (cf Oracle's freeware Virtualbox), then you can learn quite a lot quite quickly playing with the command line and reading from the internet and some of the excellent O'Reilly books. OpenSuse (free) is your friend, and try and stay out of root auth as "rm –rf /*" does exactly what it says on the tin!

The second question ("How do I get there?") depends on what you (and your z/OS sysprogs) are comfortable with:

TSO OMVS

From within TSO, issuing TSO OMVS will take you to the Unix shell. It's a little clunky as the usual line-mode operation of the shell has to be managed within a pseudo-ISPF TN3270 screen.

otelnet

This is a traditional Unix telnet server, run from within USS. The z/OS team should be able to tell you if it's been configured and which port it's running on. In our configuration it's running on port 1023, so can be accessed with the Windows telnet command like this:

> telnet <ip address> <port>
>
> telnet 192.168.200.1 1023

Configure DB2 CLP

Now that we're connected to USS and logged, we need to configure our session to run the DB2 CLP. This is implemented in Java, and makes use of JDBC type 4 drivers – also shipped with DB2 for z/OS.

Establishing .profile

To make these features generally available, we need to establish the PATH (load programs from here), the LIBPATH (load shared objects – DLLs – from here) and CLASSPATH (load Java classes from here) environment variables. This is best done from a shell script, and as we want to make these things available every time we use USS, we will apply them to the ".profile" file like this:

```
#!/bin/ksh  1
export JAVA_HOME=/usr/lpp/java/J6.0.1_64/J6.0.1_64 2

export PATH=/bin:"${JAVA_HOME}"/bin

export LIBPATH=/lib:/usr/lib:"${JAVA_HOME}"/bin
export LIBPATH="$LIBPATH":"${JAVA_HOME}"/lib
export LIBPATH="$LIBPATH":"${JAVA_HOME}"/lib/s390
export LIBPATH="$LIBPATH":"${JAVA_HOME}"/lib/s390/j9vm
export LIBPATH="$LIBPATH":"${JAVA_HOME}"/bin/classic

# jdbc
export JDBC=/u/db2b/sqllib/jdbc/classes 3
export CLASSPATH=/u/gillj:$JDBC/db2jcc.jar
export CLASSPATH=$CLASSPATH:$JDBC/db2jcc_license_cisuz.jar

# db2 clp:
export CLPHOME=/u/db2b/sqllib/base 4
export CLASSPATH=$CLASSPATH:$CLPHOME/lib/clp.jar 5
export CLPPROPERTIESFILE=$HOME/clp.properties 6
alias db2="java com.ibm.db2.clp.db2" 7

set -o vi
```

Notes:

1. This sets the shell command processor to ksh (Korne Shell)
2. JAVA_HOME is required for Java to operate and is used in other assignments
3. /u/db2b/sqllib is a symbolic link (like an MVS alias) to the operational DB2 supplied filesystem (e.g. /usr/lpp/db2/db2b10), and the jdbc path is where the Universal Driver (type 2 and type 4) is delivered.
4. The CLPHOME environment variable points to the DB2 CLP path. This is usually delivered in SMP/E as …./base, e.g. /usr/lpp/db2/db2b10/base
5. Add the JAR for DB2 CLP to the CLASSPATH variable
6. The location of the file that contains properties and database connection aliases (more on this below)
7. Sets up a command alias of "db2"

Setting Up clp.properties

As mentioned above, we use a properties file to set the general behaviour of the CLP, and also to define aliases for DB2 connections that we may want to make:

```
#Specify the value as ON/OFF or leave them blank
DisplaySQLCA=ON
AutoCommit=ON
InputFilename=
OutputFilename=
DisplayOutput=
StopOnError=
TerminationChar=
Echo=
StripHeaders=
MaxLinesFromSelect=
MaxColumnWidth=255
IsolationLevel=
#Create your own alias name for DB2 servers
#SERVER1=<URL>,<username>,<password>
DB2B=localhost:2046/DB2B,gillj,pa55w0rd
```

Most of these are self-explanatory, but the last entry sets up a connection alias for our V11 service DB2B. Note that the names that you give are case sensitive.

Start Using DB2 CLP

At the Command Line

To start using the CLP, just give the command "db2":

```
$ db2

db2 => ?
ADD XMLSCHEMA DOCUMENT
BIND
CHANGE
CALL
```

```
CONNECT
COMMIT
COMPLETE XMLSCHEMA
DECOMPOSE XML DOCUMENT
DESCRIBE
DISCONNECT
DISPLAY RESULT SETTINGS
ECHO
LIST COMMAND OPTIONS
LIST TABLES
REGISTER XMLSCHEMA
REMOVE XMLSCHEMA
ROLLBACK
UPDATE COMMAND OPTIONS
TERMINATE

db2 =>
```

As can be seen, the "?" shows some of the available commands – above and beyond regular SQL. So to connect using our previously defined alias:

```
db2 => connect to DB2B

     Database Connection Information
 Database server        =DB2 DSN11015
 SQL authorization ID   =gillj
 JDBC Driver            =IBM DB2 JDBC Universal Driver Architecture
3.65.117

DSNC101I : The "CONNECT" command completed successfully.
```

And some other examples (NB changed MaxColumnWidth in clp.properties to 20 to make this format nicely on a page):

Describing a Table

```
db2 => describe table DSN81110.EMP
COLUMN_NAME          TABLE_SCHEM          TYPE_NAME            COLUMN_SIZE DECIMAL_DIGITS IS_NULLABLE
EMPNO                DSN81110             CHAR                 6           <null>         NO
FIRSTNME             DSN81110             VARCHAR              12          <null>         NO
MIDINIT              DSN81110             CHAR                 1           <null>         NO
LASTNAME             DSN81110             VARCHAR              15          <null>         NO
WORKDEPT             DSN81110             CHAR                 3           <null>         YES
PHONENO              DSN81110             CHAR                 4           <null>         YES
HIREDATE             DSN81110             DATE                 10          <null>         YES
JOB                  DSN81110             CHAR                 8           <null>         YES
EDLEVEL              DSN81110             SMALLINT             5           0              YES
SEX                  DSN81110             CHAR                 1           <null>         YES
BIRTHDATE            DSN81110             DATE                 10          <null>         YES
SALARY               DSN81110             DECIMAL              9           2              YES
BONUS                DSN81110             DECIMAL              9           2              YES
COMM                 DSN81110             DECIMAL              9           2              YES
  14 record(s) selected

DSNC101I : The "DESCRIBE" command completed successfully.
```

Calling a Stored Procedures

Note the use of the "?" as a parameter place holder:

```
db2 => call DSNWZP(?)

Value of output parameters
--------------------------
Parameter Name : P10
Parameter Value :
SYSPAUDT/DSN6SYSP/AUDITST/DSNTIPN/ 1/AUDIT TRACE/00000000000000000000000000000000
SYSPCDB/DSN6SYSP/CONDBAT/DSNTIPE/ 4/MAX REMOTE CONNECTED/0000010000
SYSPCT/DSN6SYSP/CTHREAD/DSNTIPE/ 2/MAX USERS/00200
SYSPDFRQ/DSN6SYSP/DLDFREQ/DSNTIPL1/10/LEVELID UPDATE FREQUENCY/ON
SYSPFRQ/DSN6SYSP/PCLOSEN/DSNTIPL1/ 8/RO SWITCH CHKPTS/00010
SYSPIDB/DSN6SYSP/IDBACK/DSNTIPE/ 6/MAX BATCH CONNECT/00050
SYSPIDF/DSN6SYSP/IDFORE/DSNTIPE/ 5/MAX TSO CONNECT/00050
SYSPLOGT/DSN6SYSP/CHKTYPE/DSNTIPL1/ 1/CHECKPOINT TYPE/SINGLE
SYSPLOGL/DSN6SYSP/CHKFREQ/DSNTIPL1/ 2/RECORDS\CHECKPOINT/0000000005
SYSPLOGR/DSN6SYSP/CHKLOGR/DSNTIPL1/ 2/RECORDS\CHECKPOINT/NOTUSED
SYSPLOGM/DSN6SYSP/CHKMINS/DSNTIPL1/ 3/MINUTES\CHECKPOINT/NOTUSED
SYSPMON/DSN6SYSP/MON/DSNTIPN/ 9/MONITOR TRACE/00000000000000000000000000000000
SYSPMONS/DSN6SYSP/MONSIZE/DSNTIPN/10/MONITOR SIZE/0001048576
SYSPSYNV/DSN6SYSP/SYNCVAL/DSNTIPN/ 7/STATISTICS SYNC/NO
SYSPRLFA/DSN6SYSP/RLFAUTH/DSNTIPP1/11/RESOURCE AUTHID/SYSIBM
SYSPRLF/DSN6SYSP/RLF/DSNTIPO/ 4/RLF AUTO START/NO
SYSPRLFN/DSN6SYSP/RLFERR/DSNTIPO/ 6/RLST ACCESS ERROR /NOLIMIT
         :
DECPDATE/DSNHDECP/DATE/DSNTIP4/ 5/DATE FORMAT/ISO
DECPTIME/DSNHDECP/TIME/DSNTIP4/ 6/TIME FORMAT/ISO
DECPDLEN/DSNHDECP/DATELEN/DSNTIP4/ 7/LOCAL DATE LENGTH/000
DECPTLEN/DSNHDECP/TIMELEN/DSNTIP4/ 8/LOCAL TIME LENGTH/000
DECPIMTZ/DSNHDECP/IMPLICIT_TIMEZONE/DSNTIP4/ 9/IMPLICIT TIME ZONE/CURRENT
DECPSQL/DSNHDECP/STDSQL/DSNTIP4/10/STD SQL LANGUAGE/NO
DECPPADN/DSNHDECP/PADNTSTR/DSNTIP4/11/PAD NUL-TERMINATED/NO
DECPSSID/DSNHDECP/SSID/DSNTIPM/ 1/SUBSYSTEM NAME/DB2B
DECPLCTP/DSNHDECP/LC_CTYPE/DSNTIPF/12/LOCALE LC_CTYPE /
DECPDDRM/DSNHDECP/DEF_DECFLOAT_ROUND_MODE/DSNTIPF/13/DECFLOAT ROUNDING MODE/ROUND_HALF_EVEN

DSNC101I : The "CALL" command completed successfully.
```

Listing Tables In a Schema

```
db2 => list tables for schema DSN81110
TABLE_SCHEM          TABLE_NAME           TABLE_TYPE
DSN81110             DSN_QUERYINFO_AUX    AUXILIARY TABLE
DSN81110             DSN_QUERYINFO_AUX2   AUXILIARY TABLE
DSN81110             DSN_QUERY_AUX        AUXILIARY TABLE
DSN81110             DSN_STATEMENT_CACHE_ AUXILIARY TABLE
DSN81110             ACT                  TABLE
DSN81110             CATALOG              TABLE
DSN81110             CUSTOMER             TABLE
DSN81110             DEMO_UNICODE         TABLE
DSN81110             DEPT                 TABLE
DSN81110             DSN_COLDIST_TABLE    TABLE
DSN81110             DSN_DETCOST_TABLE    TABLE
DSN81110             DSN_FILTER_TABLE     TABLE
                     :
DSN81110             VPROJ                VIEW
DSN81110             VPROJACT             VIEW
DSN81110             VPROJRE1             VIEW
DSN81110             VPSTRDE1             VIEW
DSN81110             VPSTRDE2             VIEW
DSN81110             VSTAFAC1             VIEW
DSN81110             VSTAFAC2             VIEW
  81 record(s) selected

DSNC106I : Output has been truncated.
DSNC101I : The "LIST TABLES" command completed successfully.
```

Note the truncation message – this is because we have MaxColumnWidth set to 20 and some of the table names are longer than that.

In Batch / Scripts

Whilst the interactive CLP is handy (especially for DESCRIBE), it's more useful to run SQL as a batch tool, e.g.

```
db2 –tvf somefile.sql
```

The –t switch means standard (semicolon) statement termination, -v means verbose output (more information displayed) and –f means take input from the following file (somefile.sql) – which might look something like this:

```
connect to DB2B;
select count(*) as ct,
       substr(creator,1,30) as cr
from sysibm.systables
where type = 'T'
group by substr(creator,1,30)
order by 2
;
```

Resulting in:

```
$ db2 -tvf somefile.sql
connect to DB2B

     Database Connection Information
 Database server        =DB2 DSN11015
 SQL authorization ID   =gillj
 JDBC Driver            =IBM DB2 JDBC Universal Driver Architecture 3.65.117

DSNC101I : The "CONNECT" command completed successfully.
select count(*) as ct,
       substr(creator,1,30) as cr
from sysibm.systables
where type = 'T'
group by substr(creator,1,30)
order by 2
CT          CR
31          ADB
3           ALA
1           BOBBINS
1           DB2OSC
66          DB2PM
49          DSN81110
2           DSN811SA
4           DSN8BQRY
2           DSNRGCOL
9           GILLJ
1           IBMUSER
18          Q
1           STATS
23          SYSADM
163         SYSIBM
9           SYSIBMTS
1           SYSTOOLS
  17 record(s) selected

DSNC106I : Output has been truncated.
```

Conclusion

On the face of it, providing a Java based DB2 command line in Unix System Services might well elicit a "So what?" from most old hands on the z/OS platform. If, however, you've spent any time working on mid-range, and understand the power of scripting (cf awk, sed, grep, etc) or just like the ability to be able to CALL stored procedures and describe tables, you will find this a very useful tool.

Best of all, you've already got it – it comes as part of DB2!