

特 別 講 座

-DB2実践ノウハウ-

CLPの歩き方

DB2では、ShellプロンプトからSQL文をDB2に流し込むための機能として、CLPが提供されています。CLPとは、コマンドラインプロセッサの略語です。

CLPはOracleのSQL*Plusに相当する機能ですが、考え方には大きな差があります。多機能でコマンド編集からデータベースに対するShell的な機能まで提供するSQL*Plusに対して、CLPでは必要最低限の機能しかありません。これは両社の考え方の違いを表しているようで、なかなかおもしろいです。

逆に言うと、CLPではUNIXなどのShellコマンドとの組合せが比較的自由に行えるように工夫されており、使い方次第ではSQL*Plusに負けない有効活用が可能です。この講座では、CLPの特徴と初歩的な使い方の説明を行い、バッチ系の作業を楽に組み立てられるようにすることを目標とします。



著者：岩城裕嗣（いわき・ゆうじ）
プロフィール： UDB
ジャパン株式会社システム一部 部長。1998年ニ
イウス株式会社入社。
2000年「UDB Japan
フェア」デモンストレー
ションを担当後、勢いで
(?)創立とともにUDBジャ
パンへ。DB2技術支援担
当として活動を始める。
明治学院大学 文学部
卒。1958年生まれ。

CONTENTS

- ▶ [はじめに](#)
- ▶ [ShellコマンドとしてのCLP](#)
- ▶ [CLPらしい使い方](#)
- ▶ [エラーが出たら？](#)
- ▶ [まとめ](#)



-DB2実践ノウハウ-

CLPの歩き方

CONTENTS

[はじめに](#)[ShellコマンドとしてのCLP](#)[CLPらしい使い方](#)[エラーが出たら？](#)[まとめ](#)

はじめに

[◀ INDEX](#) [NEXT ▶](#)

CLPIは、コマンドプロンプトからdb2と入力すれば立ち上げることが出来ます。

【例1】

```
db2:ax1153a:/db2/ap> db2
(c) Copyright IBM Corporation 1993,2001
DB2 SDK 7.2.2 コマンド行プロセッサ
コマンド・プロンプトからデータベース・マネージャのコマンド、およびSQLステートメントを呼び出せます。例:
db2 => connect to sample
db2 => bind sample.bnd
一般ヘルプには?をタイプしてください。
コマンドのヘルプには? command をタイプしてください。commandは、データベース・マネージャ・コマンドの主要なキーワードのいくつかです。例:
? CATALOG DATABASE は CATALOG DATABASE コマンドのヘルプです。
? CATALOGは全 CATALOG コマンドのヘルプです。
db2 対話モードを抜けるには、コマンド・プロンプトで QUIT とタイプしてください。対話モード以外では、すべてのコマンドに接頭語 'db2' を付ける必要があります。現在のコマンド・オプションの設定をリストするには、LIST COMMAND OPTIONSとタイプしてください。
詳細は、「オンライン解説書」を参照してください。
db2 =>
```

上記【例1】のようにプロンプトが返ってきますので、この状態でSQL文、メッセージコード、DB2コマンドを与えれば結果を表示してくれます。

【例2】

```
db2 => select count(*) from scratch
1
-----
15000
1 レコードが選択されました。
```

【例3】

```
db2 => ? sql0100
```

SQL0100W FETCH、UPDATE または DELETEの対象となる行がないか、または照会の結果が空の表です。

説明: 以下に示す条件の 1 つが成立しています。

- UPDATE または DELETEステートメントに指定された検索条件を満たす行が見つかりません。
 - SELECT ステートメントの結果が空の表でした。
 - 結果表の最後の行の後ろにカーソルを位置付けたときに、FETCHステートメントが実行されました。
 - INSERT ステートメントで使用された SELECT の結果が空です。データの検索、更新、または削除は実行されませんでした。
- ユーザーの応答: 処置は必要ありません。処理は続行されます。

```
sqlcode: +100
```

```
sqlstate: 02000
```

【例3】のメッセージコード表示機能は、なかなか便利です。いちいちマニュアルを見るのは面倒なのですが、コマンド一発で中味を参照できます。表示されるメッセージは導入バージョンの内容です。PTFをあてたりしてマニュアルと導入製品のバージョンが異なる場合は、このやり方では見られないメッセージがあります。

CLPではヘルプとしてメッセージ内容とコマンドシンタクスの参照に「？」を使用することができます。残念ながらSQLのシンタクスはオンライン表示できませんので、マニュアルかInformationCenterで調べてください。

[TOP](#)

ShellコマンドとしてのCLP

対話式だけでなく、shellコマンドの一部としてCLPを使用することもできます。標準入出力に対応していますので、さまざまなUNIXコマンドとの組み合わせが可能です。

【例4】

```
db2:ax1153a:/db2/ap> cat "list tables" | db2 +p
```

表/視点

スキーマ タイプ 作成時刻

```
SCRATCH           DB2           T       2001-11-06-09.46.12.421241
```

```
TAB01           DB2           T       2001-11-05-08.53.43.490832
```

2 レコードが選択されました。

「+p」は『-p 対話式入力プロンプトを表示する ON』を、OFFに変える意味を持っています。毎回『詳細は、オンライン解説書を参照してください』の表示を見るのが嫌になったら、これで消すことができます。

対話式のCLPには入力補助機能があまりありません。マニュアルには何も書いてありませんが、現場での活用は対話式ではなくUNIXコマンドの一部として利用する方が、より実践的だと思います。

ここではちょっとマニュアルから離れて、典型的なCLP利用パターンのいくつかを御紹介します。

【例5】

```
db2:ax1153a:/db2/ap> db2 -x "select count(*) from tab01 where jan like '%00000015%'"
150
```

上記【例5】はjan列の00000015の文字検索を実行していますが、もう少し範囲を広げて行単位に00000015を検索したい時は、以下のようにします。

【例6】

```
db2:ax1153a:/db2/ap> db2 "select * from tab01" | grep 00000015 | wc -l
150
```

【例6】はSQL文でも書けますが、実践では上記の方が手になじむので、筆者はShellコマンドの一部のつもりでCLPを使用しています。たとえば、「set-o vi」などとプロンプトからおまじないしておく、コマンドhistoryでSQL文の発行履歴をとることができます。

「-x」は、『-x 列見出しの印刷を抑制する OFF』をONに変える意味です。指定しないと、以下の【例7】のようになります。

【例7】

```
db2:ax1153a:/db2/ap> db2 "select count(*) from tab01 where jan like '%00000015%'"
1
-----
150
```

これだとSQL実行結果を if で判定する時に取り扱いが面倒なので、結果をShell変数などにセットする場合は「-x」を指定してください。

【例8】

```
db2:ax1153a:/db2/ap> db2 -x "select count(*) from tab01 where jan like '%00000015%'" |
read -r xx
db2:ax1153a:/db2/ap> echo $xx
150
db2:ax1153a:/db2/ap> db2 "insert into scratch values($xx,'test')"
DB20000I SQL コマンドが正常に終了しました。
1 レコードが選択されました。
```

Shell変数を有効活用することで、【例8】のようにデータの受け渡しが可能となります。簡単な処理ならこれで大丈夫です。

[TOP](#)

◀◀ INDEX NEXT ▶▶



-DB2実践ノウハウ-

CLPの歩き方

CONTENTS

[はじめに](#)[ShellコマンドとしてのCLP](#)[CLPらしい使い方](#)[エラーが出たら？](#)[まとめ](#)

CLPらしい使い方

◀◀ INDEX ◀ PREV NEXT ▶▶

ここまでは一つのコマンドやSQL文をCLPに流し込む方法を説明してきましたが、実践では複数の命令を一括してCLPに渡す方式がよく使われます。標準入出力に対応しているため、[【例3】](#)、[【例6】](#)のやり方で複数のSQL文をまとめてCLPに渡せば処理してくれます。

なぜ、まとめてSQL文を渡すかというと、SQL文とShellスクリプトを別々に作るためです。Shell書きができる人がSQL文を組み立てられるとは限りませんし、逆もそうです。現場では、往々にしてShellとSQLの分別管理を行いたい場合があります。

また、パフォーマンス上の問題もあります。コマンドごとにCLPを起動した場合と、SQL文をまとめてCLPに受け渡すのでは、後者の方が若干パフォーマンス面で有利な場合があるのです。コーディネータエージェントは最初にCLPからconnectした時に作成したものが使われるので、繰り返しCLPを呼び出してもセッション管理上は問題ないのですが、大量のSQL文を一個のShellにまとめた時などは、db2の起動と終了のオーバーヘッド分だけわずかに遅くなります。

CLPにはファイルからテキストを取り込むオプションがあります。意味合いは[【例3】](#)と同じで、この方法でも複数のSQL文を一括処理可能です。

【例9】

```
db2:ax1153a:/db2/ap> db2 -vf aa.sql
connect to db01a
データベース接続情報
データベース・サーバー = DB2/6000 7.2.2
SQL 権限 ID = DB2
ローカル・データベース別名 = DB01A
select count(*) from scratch
1
-----
15001
1 レコードが選択されました。
terminate
DB20000I TERMINATE コマンドが正常に終了しました。
db2:ax1153a:/db2/ap> cat aa.sql
connect to db01a
select ¥
count(*) ¥
from scratch
terminate
```

「-v」は、CLPが実行するSQL文をエコーバックするかどうかという指定です。実行SQL文をshellなどで組み立てた時や、お客様の本番JOBなどでは指定した方が無難です。「-f」は、ファイルからテキストを取り込む意味です。

「connect」と「terminate」をSQL文の前後につけているのは、お作法です。上記【例9】のケースでは、プロンプトで既にconnectされていればSQL文の実行は可能です。ただし、下記【例10】のような場合はShellがバックグラウンドで動き、プロンプトに紐付いたDB2セッションは使われませんので、注意が必要です。

【例10】

```
db2:ax1153a:/db2/ap> db2 connect to db01a
データベース接続情報
データベース・サーバー = DB2/6000 7.2.2
SQL 権限 ID = DB2
ローカル・データベース別名 = DB01A
db2:ax1153a:/db2/ap> ./aa.sh
SQL1024N データベース接続が存在しません。 SQLSTATE=08003
db2:ax1153a:/db2/ap> cat aa.sh
#!/bin/ksh
db2 "select count(*) from scratch ;"
```

デフォルトでは、文の区切り文字は改行文字になっています。長いSQL文で一行では見にくい場合などは、上記【例9】のように「¥」で区切ることができます。ただ、これだとSQL文が汚れてしまうので、筆者は「¥」マークで区切るのが、あまり好きではありません。そこで、いつもは【例11】のようにしています。

【例11】

```
db2:ax1153a:/db2/ap> db2 ?tvf aa.sql
connect to db01a
データベース接続情報
データベース・サーバー = DB2/6000 7.2.2
SQL 権限 ID = DB2
ローカル・データベース別名 = DB01A
select count(*) from scratch
1
-----
15001
1 レコードが選択されました。
db2:ax1153a:/db2/ap> cat aa.sql
connect to db01a ;
select
count(*)
from scratch ;
```

「-t」は、区切り文字をブランク以外に設定するオプションです。区切り文字のデフォルトは、Oracleと同じ「;」(セミコロン)です。「;」以外の区切り文字を使用したい場合は「d」オプションで任意に変更が出来ます。

通常、区切り文字の変更は任意に行ってよいのですが、DB2の都合上、区切り文字を変更しなければならない場合があります。

【例12】

```
db2:ax1153a:/db2/ap> cat proc01.sql
drop procedure proc01
@
create procedure proc01(
in XX int,
in YY char(20)
)
language sql
main : begin
insert into scratch values(XX,YY) ;
end main
@
db2:ax1153a:/db2/ap> db2 -td@ -f proc100.sql
DB20000I SQL コマンドが正常に終了しました。
```

SQL ProcedureのコンパイルやMacro PSMなどでは、仕様で「;」がプロシージャの区切り文字となっていますので、CLPの区切り文字とだぶってしまいます。このケースでは「-t+d+区切り文字」で実行する必要があります。

ところで、いろいろ説明してきましたが、オプション変更のやり方が「db2-xx」式しかないとすると、対話式でCLPを使用する場合はオプション変更ができないことになってしまいます。実は、対話式の場合に備えてオプション変更コマンドは別途用意されています。ここで、オプションの一覧とコマンドの説明をしましょう。

【例13】

```
db2 => ? update command options
UPDATE COMMAND OPTIONS USING {options ...}
オプション :
a {ON|OFF} SQLCA を表示する
c {ON|OFF} 自動コミット
e {ON {C|S} | OFF} SQLCODE/SQLSTATE を表示する
l {ON filename | OFF} 履歴ファイルにコマンドのログをとる
n {ON|OFF} 改行文字を除去する
o {ON|OFF} 出力を表示する
p {ON|OFF} db2 対話式プロンプトを表示する
r {ON filename | OFF} 出力報告をファイルに保管する
s {ON|OFF} コマンド・エラーで実行を停止する
v {ON|OFF} 現行コマンドをエコーにする
w {ON|OFF} 警告メッセージを表示する
z {ON filename | OFF} すべての出力をファイルに保管する
```




-DB2実践ノウハウ-

CLPの歩き方

CONTENTS

[はじめに](#)[ShellコマンドとしてのCLP](#)[CLPらしい使い方](#)[エラーが出たら？](#)[まとめ](#)

エラーが出たら？

[◀◀ INDEX](#) [◀ PREV](#)

ここまで、ファイルからまとまったSQL文をCLPに渡す方法をご説明しました。それでは次に、SQL文の中でエラーが発生した場合はどうすればよいのか、見ていきましょう。

【例14】

```
db2:ax1153a:/db2/ap> ./imp01.sh
データベース接続情報
データベース・サーバー = DB2/6000 7.2.2
SQL 権限 ID = DB2
ローカル・データベース別名 = DB01A
ok !
DB20000I TERMINATE コマンドが正常に終了しました。
db2:ax1153a:/db2/ap> cat imp01.sh
#!/bin/ksh
db2 connect to db01a
rc=`db2 -ec +o ¥
"import from data02 of del insert into scratch"
if [[ $rc -eq 0 ]];then
echo "ok !"
else
echo "error !"
fi
db2 terminate
exit $rc
```

「+o」とは、標準出力には何も返さないというオプションです。「-ec」はSQLCODEを返せ、という意味なのでrcにはSQL文のリターンコードがセットされます。後はkshのifで判定して、次のステップに進むかどうかを判断します。

上記【例14】は単一のSQL文を実行した場合の判定方法ですが、ファイルから複数のSQL文を読みこんだ場合は、以下のようにします。

【例15】

```
db2:ax1153a:/db2/ap> ./atm_sql3.sh
データベース接続情報
データベース・サーバー = DB2/6000 7.2.2
SQL 権限 ID = DB2
ローカル・データベース別名 = DB01A
ok!
DB20000I TERMINATE コマンドが正常に終了しました。
db2:ax1153a:/db2/ap> cat atm_sql3.sh
db2 connect to db01a
cat /dev/null > atm_sql1.log
rc=`db2 +c +o -td@ -s -ec << EOF >> atm_sql3.log
drop table scratch @
create table scratch (
pk int ,
text varchar(30)
)
in DATASP1
index in DATASP1
not logged initially @
begin atomic
declare a int default 0 ;
declare b int default 0 ;
l : for row as
select jan from tab01
do
set b = b + 1 ;
insert into scratch values( b,jan) ;
end for l ;
end @
create unique index ix1_scratch
on scratch (
pk asc
) @
EOF`
if [[ $rc -eq 0 ]];then
echo "ok!"
else
echo "error!"
echo $rc
fi
db2 terminate
```

【例15】のオプションのうち、「+c」は自動コミットを行わない、「-s」はエラーがあったらすぐ抜ける、という意味です。これに「-ec」と「+o」を組み合わせるとエラーコードだけをもらうように設定しておけば、エラー時の判定が可能となります。

たとえば、【例15】のcreate tableでエラーが発生したとすると、以降のPSMとcreate indexは処理されずに、結果をrollbackしてrcにSQLCODEがセットされ、次にifが実行されます。

もちろん「-f」と組み合わせると、SQL文の本体を別のファイルとすることもできます。

【例16】

```
rc=`db2 +c +o -td@ -s -ec 「ファイル名」 >> atm_sql3.log`
```

【例15】のSQL文は【例16】に書き換えることができます。

まとめ

CLPIは必要最低限の機能のため、逆に使い方のバリエーションに創意・工夫の余地が広いツールです。開発現場での自由裁量を活かして、より有効にDB2を活用しましょう。

最後に（本題とは直接関係ないのですが）、[【例15】](#)のSQL文について説明します。よく聞かれる話題ですが、create文には「not logged initially」というオプション構文があります。これは最初にテーブルをcreateした時のLUWでinsertやupdateが発生した場合、logをとらないという意味です。そこを抜けてしまえばlogは取得されます。また、alter文でlog取得のon/offを切替えるためには、最初にこれを指定しておく必要があります。指定して損はないオプションですので、お勧めです。

[TOP](#)[◀◀ INDEX](#) [◀ PREV](#)