SQLcl – SQL Developer trifft SQL*Plus

Rainer Willems, ORACLE Deutschland B.V. & Co. KG

SQL*Plus ist das meistverbreitete Datenbank-Werkzeug. Es ist universell, schnell und schlank, aber nicht immer besonders benutzerfreundlich. Insbesondere Entwickler, aber auch DBAs haben sich gefreut, als Oracle nach Jahren des Wartens mit dem SQL Developer ein grafisches Frontend zur Verfügung gestellt hat, das inzwischen sehr mächtig geworden ist. Ein Command-Line-Tool ist damit allerdings nicht komplett ablösbar. Mit SQLcl kommt nun ein neues Werkzeug von Oracle, das basierend auf dem SQL Developer einige Annehmlichkeiten mit sich bringt und diese mit dem Charme und den Vorteilen eines Command-Line-Tools verbindet.

Das Tool findet man einfach im Bereich des SQL Developer auf OTN. Den 14-MB-Download des Java-basierten Tools (erfordert Java-8-Runtime-Umgebung) auspacken, und man kann ohne Installation loslegen *(siehe Abbildung 1)*. Je nach Betriebssystem wird „sql" oder „sql.bat" aufgerufen und einfach per EZConnect-Syntax verbunden *(siehe Listing 1)*. TNS steht natürlich auch zur Verfügung.

Der HELP-Befehl ohne zusätzlichen Parameter hat die überaus nette Eigenschaft, bei der Auflistung aller Help-Topics anzuzeigen, welche Befehle man schon aus SQL*Plus kennt und welche neu dazugekommen sind *(siehe Abbildung 2)*.

Arbeiten mit SQLcl

Mit die schönsten Eigenschaften für den Command-Line-Liebhaber werden das flexible Editieren, eine Historie sowie das einfache Formatieren der Ausgaben sein. Typisches Szenario nach der Eingabe eines Befehls ist, dass man sowohl am Statement als auch an der Ausgabe etwas ändern möchte *(siehe Listing 2)*.

Fangen wir mit der Ausgabe an. Statt umständlicher „COL[UMN]"- oder „LINESIZE"-Kommandos verwendet man einfach „SET SQLFORMAT" und gibt das gewünschte Format an. Der HELP-Befehl zeigt die Optionen an *(siehe Listing 3)*.

Einfacher kann ein Überführen in CSV, JSON und Co. nicht sein. Erwähnenswert die Variante „Insert", die, wie der Name schon sagt, Insert-Statements aus der Ausgabemenge generiert − eine tol-

le Möglichkeit, um jemandem schnell ein paar Beispieldaten zum Einspielen zu geben *(siehe Listing 4)*.

Listing 5 zeigt die „Ansiconsole" für eine hübschere Ausgabe am Bildschirm. Zusätzlich wird eine Farbcodierung für die Ausgabe unterstützt. Die Möglichkeiten hierzu sind ebenso vielfältig wie die Beispiele, die man im Web dazu findet. Eine spezielle Syntax in der Ausgabe kann für die Formatierung (nicht nur Farben) verwendet werden. Ein Beispiel für „rot und fett" ist „select ‚@|red,bold ‚||<spalte>||'|@' from <tabelle>". Kombiniert mit dem CASE-Statement, lassen sich somit übersichtliche Ausgaben erzeugen. Das Ausgabeformat als solches kann man übrigens auch direkt im Statement mitgeben, was sinnvoll ist, wenn man öfter mal das Format wechseln möchte *(siehe Listing 6)*.

Das Statement ist schnell editiert, und zwar nicht mit Befehlen wie „l 3" und dann „c/2000/3000", sondern einfach im Prompt-Cursor nach oben (das bringt uns zum letzten Befehl), dann den Cursor nach links (jetzt ist man im letzten Befehl), und schon kann man munter mit dem Cursor durch das Statement wandern, inline editieren, was man möchte, und anschließend den geänderten Befehl wieder abschicken. Natürlich kann man sich jetzt auch beim Schreiben eines Statements beliebig im Statement (auch zeilenübergreifend) bewegen, und zwar nicht nur mit dem Cursor, sondern auch per CTRL-W, -S, -A und -E direkt springen.

Aus dem Hinweis darauf, dass der erste Cursor-Befehl einen zum letzten Statement bringt, lässt sich schlussfolgern, dass man so auch weiter zurück in die Historie blättern kann. Das größte Manko von SQL*Plus ist damit also überwunden. Noch anschaulicher geht dies mit dem „HIST[ORY]"-Befehl. Dieser listet bis zu 100 Statements auf; man kann diese durch Angabe der Nummer auch direkt auswählen *(siehe Listing 7)*.

Sehr schön ist auch die in anderen Tools ja inzwischen gängige Code-Completion mittels TAB-Taste. Häufig benötigte Statements lassen sich auch mit einem Alias versehen, um so schnell wieder hervorgekramt werden zu können *(siehe Listing 8)*. Dies funktioniert natürlich auch ohne Bind-Parameter und lässt sich so hervorragend von DBAs für schnelle Status-Abfragen einsetzen.

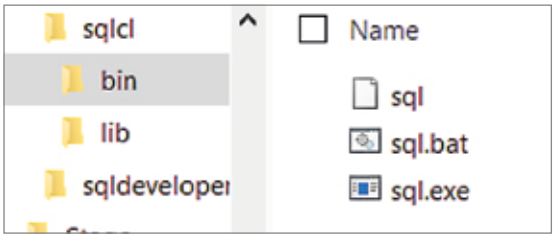

Abbildung 1: SQLcl nach dem Auspacken

```
Username? (''?) scott @localhost:1521/developer.de.oracle.com
Password? (**********?) *****
```

Listing 1

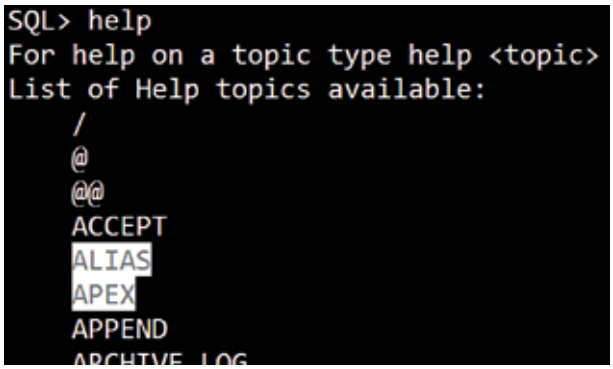

Abbildung 2: Das HELP-Kommando markiert Befehle

Rund um die Objekte

Auf der gerade beschriebenen ALIAS-Methode beruhen vordefinierte Abfragen, die unter „tables" und „tables2" abgelegt sind und einen schnellen Überblick über die Tabellen im Schema geben. Will man sich über eine Tabelle informieren, gibt es nun die ausführlichere Alternative zum „DESC[RIBE]" in der Form „INFO <tabellenname>". Noch ausführlicher geht es mit „INFO+ <tabellenname>". Das passende DDL-Kommando zum Erstellen ei-

```
SQL>  select *
  2   from emp
  3   where empno>2000
  4   order by job
  5   /

     EMPNO ENAME      JOB           MGR     HIREDATE       SAL     COMM       DEPTNO
---------- ---------- ---------  ---------- --------   ---------- ---------- ----------
      7788  SCOTT      ANALYST        7566   09.12.82       3000               20
      7902  FORD       ANALYST        7566   03.12.81       300                20
 …
```

Listing 2

```
SQL> help set sqlformat
SET SQLFORMAT
   SET SQLFORMAT {csv,html,xml,json,ansiconsole,insert,loader,fixed,default}
```

Listing 3

```
SQL> set sqlformat insert
SQL> /
REM INSERTING into EMP
SET DEFINE OFF;
Insert into EMP (EMPNO,ENAME,JOB,MGR,HIREDATE,SAL,COMM,DEPTNO) values
('7788','SCOTT','ANALYST','7566',to_timestamp('09.12.82','DD.MM.RR HH24
:MI:SSXFF'),'3000',null,'20');
…
```

Listing 4

```
SQL> set sqlformat ansiconsole
SQL> /
EMPNO ENAME   JOB        MGR     HIREDATE   SAL     COMM       DEPTNO
7788  SCOTT   ANALYST    7566    09.12.82   3000               20
7902  FORD    ANALYST    7566    03.12.81   3000               20
…
```

Listing 5

```
SQL> select /*json*/ * from emp;
{"results":[{"columns":[{"name":"EMPNO","type":"NUMBER"},{"name":"ENAME
","type":"NUMBER"},{"name":"JOB","type":"NUMBER"},{"name":"MGR","type":
"NUMBER"},{"name":"HIREDATE","type":"NUMBER"},{"name":"SAL","type":"NUM
BER"},{"name":"COMM","type":"NUMBER"},{"name":"DEPTNO","type":"NUMBER"}
],"items":[{"empno":7839,"ename":"KING","job":"PRESIDENT","hiredate":"1
7.11.81","sal":5000,"deptno":10}, ……. }]}]}
```

Listing 6

```
SQL> help history
HISTORY
---------
history [<index> | FULL | USAGE | SCRIPT | TIME | CLEAR (SESSION)?] | FAILS
```

Listing 7

```
SQL> alias ma=select ename, job, sal, deptno
   from emp where sal>:b1 and deptno=:b2;

SQL> ma 1000 10
ENAME     JOB        SAL     DEPTNO
KING      PRESIDENT  5000    10
CLARK     MANAGER    2450    10
MILLER    CLERK      1300    10

SQL> ma 2000 10
ENAME     JOB        SAL     DEPTNO
KING      PRESIDENT  5000    10
CLARK     MANAGER    2450    10
```

Listing 8

nes Objekts erhält man einfach per DDL-Befehl *(siehe Listing 9)*. Mit „CTAS" kann man gleich im Schema eine Tabelle kopieren *(siehe Listing 10)*.

Dieses auf Basis von „DBMS_METADATA" generierte Statement wird nicht gleich ausgeführt, sondern kann zuvor manipuliert werden, um beispielsweise nur Teildaten in die neue Tabelle zu übernehmen. Da dies eben auf „DBMS_METADATA" basiert, kann man natürlich auch auf die Generierung Einfluss nehmen und etwa die Storage-Parameter oder den Schema-Namen weglassen oder durchsetzen, dass Constraints als eigenes Statement generiert werden.

Apex und mehr

Sehr praktisch ist der Apex-Befehl, mit dem nicht nur eine Liste aller Apex-Applikationen erzeugt, sondern insbesondere eine Apex-Applikation exportiert werden kann. Dies ist auch für REST-Services im ORDS möglich. Eine komplette SODA-Schnittstelle ist im SQLcl integriert. SODA steht für „Simple Oracle Document Access" und ermöglicht das schemalose Arbeiten mit der Oracle-Datenbank, basierend auf JSON-Dokumenten.

Weitere neue Kommandos in alphabetischer Reihenfolge

Mit **BRDIGE** kann nun Schema- und Datenbank-übergreifend gearbeitet werden, ohne Datenbank-Links anlegen zu müssen. So lässt sich beispielsweise mit „BRIDGE <targetTableName> as „<jdbcURL>"(<sqlQuery>);" eine Tabelle schnell auf Basis einer Query in einem anderen Schema anlegen.

CD erlaubt es, ohne SQLcl zu verlassen, das OS-Verzeichnis zu wechseln, ist also die Kurzvariante des Vorgehens bei SQL*Plus, bei der man erst ins OS und nach dem Verzeichniswechsel wieder zurück ins Tool springen musste.

FORMAT formatiert das SQL im SQLcl-Buffer oder in einem File. Dabei können Formatierungsregeln aus dem SQL Developer herangezogen werden, die als XML exportiert wurden *(siehe Listing 11)*.

Ähnlich zum bereits erwähnten ALIAS können mit **NET** unter Kurzbezeichnungen quasi Short-Cuts zu Netzwerk-Informationen abgelegt sein.

```
SQL> help ddl
DDL
---
DDL generates the code to reconstruct the object listed. Use the type op-
tion for materialized views. Use the save options to save the DDL to a file.
DDL [<object_name> [<type>] [SAVE <filename>]]
```

Listing 9

```
SQL> ctas emp newemp
CREATE TABLE "SCOTT"."NEWEMP"
   (   "EMPNO",
       "ENAME",
       …
       "DEPTNO",
       PRIMARY KEY ("EMPNO")
   USING INDEX PCTFREE 10 INITRANS 2 MAXTRANS 255 COMPUTE STATISTICS
 …
   BUFFER_POOL DEFAULT FLASH_CACHE DEFAULT CELL_FLASH_CACHE DEFAULT)
   TABLESPACE "USERS"
 as
select * from EMP
```

Listing 10

```
SQL> select ename, job, sal from emp where deptno=10 order by sal desc;
ENAME         JOB               SAL
----------    ---------  ----------
KING          PRESIDENT      5000
CLARK         MANAGER        2450
MILLER        CLERK          1300

SQL> format buffer
   1   SELECT
   2       ename,
   3       job,
   4       sal
   5   FROM
   6       emp
   7   WHERE
   8       deptno = 10
   9   ORDER BY
 10*       sal DESC
```

Listing 11

```
SQL> oerr ora 1200
01200. 00000 - "actual file size of %s is smaller than correct size of
%s blocks"
*Cause:   The size of the file as returned by the operating system
…
```

Listing 12

```
SQL> select to_char(sysdate,'HH24:MI:SS') Uhrzeit from dual;
UHRZEIT
--------
09:59:33

SQL> repeat 10 5

Running 3 of 10 @ 10:0:12.358 with a delay of 5s
UHRZEIT
10:00:12
```

Listing 12

LOAD ermöglicht das Laden einer CSV-Datei in eine Tabelle. Voraussetzung ist eine Kopfzeile, die mit den Spalten der Tabelle übereinstimmt.

Klein, aber fein ist **OERR**, um schnell mal nachzulesen, was eine Fehlermeldung bedeutet *(siehe Listing 12)*. Das funktioniert nicht nur für ORA-Fehler.

REPEAT wiederholt das letzte Statement „n" Iterationen mit einer Pause von „m" Sekunden *(siehe Listing 13)*.

Scripting mit **SCRIPT** ist jetzt auch im SQLcl möglich, und das in diversen JVM-Sprachen. Dies geschieht auf Basis des JSR-223. Damit lässt sich SQLcl individuell schön erweitern. Auf der DOAG Connect hat Dietmar Aust beispielhaft vorgeführt, wie er mit diesem Ansatz und JavaScript viele Bilder mit einem Script aus dem Filesystem in die Datenbank pumpt.

SSHTUNNEL und **TNSPING** machen das, was ihr Name halt schon sagt.

Wer übrigens seine Umgebung bezüglich Ausgabeformat und anderem vordefinieren will, der kann dies in der Datei „login.sql" tun. Dies bietet sich beispielsweise für die Einstellung des Prompts an: „SET SQLPROMT „_USER> „".

Weitergehende Informationen

- *http://krisrice.blogspot.co.uk*
- *http://www.thatjeffsmith.com*
- *http://www.oracle.com/technetwork/developer-tools/sqlcl/overview/index.html*
- *https://github.com/oracle/oracle-db-tools/tree/master/sqlcl*

Hinweis: Noch ist das Tool im „Early Adopter"-Status. Dies sollte sich mit der Datenbank-Version 12.2 ändern.

Rainer Willems
rainer.willems@oracle.com