die Herausforderungen in den Details. So sind die MS-AD-Integration und Kerberos-Konfiguration bei komplexen AD-Architekturen um einiges aufwändiger. Zudem braucht es für den unternehmensweiten Einsatz zwingend ein geeignetes Benutzer- und Rollen-Konzept. Dafür muss häufig mehr Zeit eingeplant werden als für die einfache Konfiguration von OUD. Die verschiedenen Bugs im Kerberos-, LDAP- beziehungsweise SSL-Umfeld erschweren einem das Leben zusätzlich. Sind diese Hürden erst einmal gemeistert, hat man aber ein verlässliches und sicheres System für die zentrale Benutzerverwaltung von Oracle-Datenbanken.

Stefan Oehrli
stefan.oehrli@trivadis.com

Scripting mit SQLcl – Batch Scripts auf einem neuen Level

Robert Marz, its-people GmbH

SQLcl steht bereit, das gute alte SQL*Plus als Kommandozeile für die Datenbank abzulösen und dessen Funktionsumfang auf die Höhe der Zeit bringen. Die spannendste Neuerung ist das Scripting: Batch Scripts können in JavaScript verfasst werden. Aber auch andere Sprachen wie Python oder Lua sind möglich. Dadurch eröffnet sich ein wahres Füllhorn an Möglichkeiten, von denen „viele erstmal nicht offensichtlich sind.

SQLcl steht für „Oracle SQL Developer Command Line". Wie der Name vermuten lässt, wurde der komplette Funktionsumfang des SQL-Developer-Worksheet als Kommandozeile ausgekoppelt. SQLcl wird aktiv entwickelt: Zur Oracle Open World im September 2016 wurde ein Produktions-Release veröffentlicht, das seitdem alle sechs bis acht Wochen ein Update erhält.

SQLcl möchte das neue SQL*Plus werden und ist seit der Datenbank-Version 12c Release 2 im „$ORACLE_HOME/bin"-Verzeichnis zu Hause. Wer die neueste Datenbank-Version bereits im Einsatz hat, kann das Tool direkt benutzen. Alle anderen können es von der SQL-Developer-Seite des Oracle-Technologie-Networks herunterladen. Der Download ist mit einer Größe von rund 19 MB überschaubar. Die Installation ist mit dem Entpacken in ein beliebiges Verzeichnis abgeschlossen. Winzige Voraussetzung ist ein aktuelles Java im Suchpfad des Betriebssystems. Ein zusätzlicher Oracle-Client ist nicht nötig.

SQL*Plus „plus"

SQLcl deckt bis auf einige gewollte Ausnahmen den kompletten Funktionsumfang von SQL*Plus ab und enthält außerdem die Erweiterungen des SQL-Developer-Worksheet. Die Entwickler haben dem Tool darüber hinaus die Möglichkeit gegeben, Scripts in anderen Programmiersprachen auszuführen. Einzige Bedingung: Es muss eine Implementierung der Sprache existieren, die dem Standard „JSR-223" folgt und damit in der

Java Virtual Machine läuft. Java 1.8 bringt von Haus aus die Nashorn-Library mit. Das ist eine Implementierung von JavaScript für die JVM. Daher ist JavaScript die Standard-Sprache für solche Scripts.

SQLcl wird genau wie SQL*Plus aufgerufen. Der einzige Unterschied: Das Kommando heißt „sql". Man spart also beim Tippen die vier Tastendrücke für das „plus" *(siehe Abbildung 1)*. Leider wird diese Einsparung durch die verlängerte Startzeit des Java-Programms gegenüber dem in C implementierten SQL*Plus mehr als wieder aufgefressen. Nach dem erfolgten Start ist aber kein Geschwindigkeitsunterschied mehr festzustellen.

*Abbildung 1: SQLcl wird wie SQL*plus aufgerufen*

Die Dokumentation der neuen Scripting-Features ist äußerst spärlich. Es gibt noch kein eigenes Handbuch, sondern lediglich das Readme und die Beispiele im offiziellem Oracle-DB-Tools-GitHub-Repository *(siehe „https://github.com/oracle/oracle-db-tools/tree/master/sqlcl")*. SQLcl erweitert die verwendete Sprache um globale Objekte, die dann direkt im Script genutzt werden können:

- *ctx*
  Mit „ctx.write(„Hallo Welt")" wird Text in die Konsole geschrieben. „ctx.cloneCLIConnection()" eröffnet eine neue Datenbanksitzung mit den Zugangsdaten der aktuellen SQLcl-Verbindung.

- *sqlcl*
  „sqlcl.setStmt(„prompt Hallo Welt")" füllt den Kommandozeilen-Buffer von SQLcl, als würde in die Tastatur getippt, „sqlcl.run()" führt den Buffer anschließend aus. Alle Befehle, die auf diese Weise ausgeführt werden, können anschließend mit den Cursor-Tasten wieder aus der Historie von SQLcl hervorgeholt werden.

- *util*
  Schließlich gibt es noch „util.execute();" in verschiedenen Varianten, mit denen, ähnlich dem „execute immediate" aus PL/SQL, mit Bindevariablen versehene SQL-Statements abgesetzt werden können.

Aufwärmen

Neben dem „@"-Kommando zum Starten von althergebrachten SQL-Scripts kennt SQLcl das neue Schlüsselwort „script". Gefolgt von einem Dateinamen lädt es diese Datei und führt sie aus. Folgt nach „script" ein Zeilenumbruch, kann JavaScript direkt über die Tastatur in den Buffer eingegeben werden. Ein abschließendes „/" führt den Buffer dann aus *(siehe Abbildung 2)*.

Unser erstes Script in SQLcl könnte wie in *Listing 1* aussehen. Nach der Ausführung passiert … nichts. Der einzige Inhalt des Scripts ist eine Variablen-Deklaration. Variablen und Funktionen überleben in einer SQLcl-Session allerdings das Ende ihres Scripts. Im nächsten Script-Aufruf kann – ohne Deklaration – erneut darauf zugegriffen werden *(siehe Listing 2)*.

Ein nützliches neues Feature von SQLcl sind Aliase. Sie können in SQL, PL/SQL oder als Script angelegt werden. SQLcl speichert die Definition in einer eigenen Datei, sodass einmal angelegte Aliase auch beim nächsten Start von SQLcl wieder zur Verfügung stehen. Im Beispiel in *Listing 3* legen wir einen Alias „rememberMe" an, der den Inhalt der Variable „dontForget" ausgibt, wenn sie denn bereits deklariert wurde. Der Befehl „alias list" zeigt alle definierten Aliase an. Bei

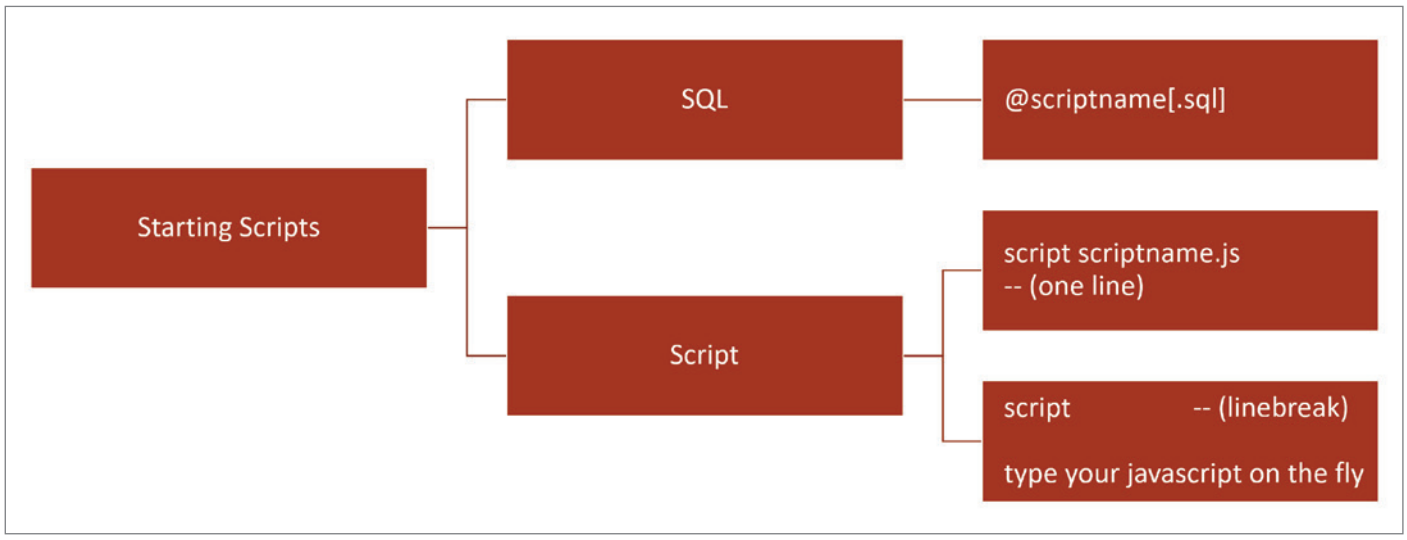

Abbildung 2: Starten von Scripts in SQLcl

der Installation bringt SQLcl bereits einige Beispiele mit.

Die Methode „ctx.write()“erzeugt die Ausgaben in der Konsole. Zeichenketten, die damit geschrieben werden sollen, müssen zwingend mit einem „\n” abgeschlossen sein, ansonsten erfolgt keine Ausgabe. Die Funktion „print()“ erfüllt den gleichen Zweck und ist sogar bequemer, weil hier das abschließende Zeichen nicht erforderlich ist. Beide benutzen unterschiedliche Output-Streams, wie man sehen kann, wenn die Aufrufe vermischt werden *(siehe Listing 4)*. Das führt dazu, dass die Reihenfolge der ausgegebenen Zeilen nicht wie erwartet ist.

Flusskontrolle

Jeder, der schon einmal ein SQL-Script für SQL*Plus geschrieben hat, weiß, dass diese nicht allzu dynamisch werden können: Über die Anweisung „whenever sqlerror” konnte die Ausführung abgebrochen oder der Fehler ignoriert werden. Schleifen oder Verzweigungen sind nicht möglich. Wir alle haben uns mit den üblichen Workarounds abgefunden: das „Zusammenspoolen” von Scripts oder das Ausweichen auf PL/SQL. In SQLcl steht für Batch Scripts endlich der ganze Umfang moderner Script-Sprachen zur Verfügung. Das Beispiel in *Listing 5* legt eine Tabelle nur dann an, wenn sie nicht bereits existiert. Selbstverständlich sind auch Schleifen möglich. Das nächste Beispiel berechnet eine Fibonacci-Folge und fügt die Werte mithilfe von Bind-Variablen in die Tabelle ein *(siehe Listing 6)*.

Blobs vom Dateisystem laden

Da die benutzten Script-Sprachen für die Java Virtual Machine geschrieben und dort ausgeführt werden, ist es einfach, aus der Script-Sprache heraus auf Java-Klassen und Objekte zuzugreifen. Das Beispiel in *Listing 7* macht sich dies für das Laden von Dateien aus dem Dateisystem zunutze: Der Dateiname wird beim Aufruf als Argument an das Script übergeben, die Datei gelesen und als Blob in der Datenbank gespeichert. Sollen anstelle von Blobs Clobs gespeichert werden, lauten die Zeilen 7 und 8 wie in *Listing 8*.

```
script
   var dontForget = 'Blumen kaufen';
/
```

Listing 1

```
script
   ctx.write(dontForget+'\n');
/
```

Listing 2

```
alias rememberMe=script
   if(typeof dontForget != 'undefined') print("Don‘t forget: "+ dontFor-
get);
```

Listing 3

```
script
   ctx.write('1 - ctx\n');
   print(    '2 - print');
   ctx.write('3 - ctx\n');
   print(    '4 - print');
   ctx.write('5 - ctx\n');
   print(    '6 - print');
/
2 - print
4 - print
6 - print
1 - ctx
3 - ctx
5 - ctx
```

Listing 4

```
script
// Check auf Vorhandene DB Objekte
/* Test ob die Zieltabelle vorhanden ist */
var tabName = „SQLCL_DEMO“;
var tabCnt = util.executeReturnOneCol(
      'select count(*) '+   // JS ist es egal, welche
      ' from tabs '     +   // Anführungszeichen verwandt
      " where table_name = ,"+tabName+"'" // werden
      );

if (tabCnt === 0){
   ctx.write("Tabelle "+tabName+" nicht vorhanden\n" +
       "Lege sie an.\n");

   sqlcl.setStmt( "set echo on\n"
       + "set feedback on \n"
       + "create table "+tabName+"   ( indx     number not null    \n"
       + "                           , fibonacci number             \n"
       + "                           );                             \n"
       // Alles was in SQLcl geht, geht auch hier:
       + "set serveroutput on size unl \n"
       + "alter table "+tabName+" add constraint pk_"+tabName+" primary
key (indx);");
   // Die einzelnen Befehle sind anschließend in der SQLcl Historie ab-
gelegt...
   // und können mit den Cursor-Tatsten wieder abgerufen werden
   sqlcl.run();
} else {
   ctx.write("Tabelle "+tabName+" schon vorhanden.\n");
}
/
```

Listing 5

Array Magic

Arrays sind mächtige Sprachkonstrukte. In JavaScript können Arrays sowohl einfache Werte als auch komplexe Objekte aufnehmen und beliebig tief geschachtelt sein. Im Beispiel in *Listing 9* definieren wir eine Benutzerliste als Array, bestehend aus Objekten mit je zwei Feldern „user" und „password". Wenn das Array einmal angelegt ist, bleibt es wie alle Variablen für den Rest der SQLcl-Session persistent. Das nächste Script durchläuft alle Elemente des Arrays in einer Schleife und gibt den Inhalt des Felds „user" aus *(siehe Listing 10)*.

In JavaScript muss keine Schleife bemüht werden, um ein Array zu durchsuchen: Arrays können mit Filterfunktionen erweitert werden. Die Funktion „getPasswd" aus dem nächsten Beispiel definiert einen solchen Filter, der die anonyme Funktion innerhalb der runden Klammern für alle Elemente des Arrays aufruft. Die Filterfunktion erzeugt ein neues Array mit allen Elementen, für die die anonyme Funktion „true" zurückliefert *(siehe Listing 11)*. In dem Beispiel wird davon ausgegangen, dass die User im Array eindeutig sind. Es ist also in Ordnung, nur das Kennwort des ersten Treffers zurückzuliefern.

Background Sessions

JavaScript kennt keine Threads, Java hingegen schon. Mit dem Thread-System von Java lassen sich aus einer SQLcl-Session beliebig viele weitere Verbindungen in die Datenbank eröffnen. Auf diese Weise können Aufgaben parallel abgearbeitet oder die Hauptsession überwacht werden.

Im offiziellem Oracle-DB-Tools-GitHub-Repository gibt es zwei Beispiele, die den Einsatz verdeutlichen, für diesen Artikel jedoch zu lang sind: „bg.js" verdeutlicht das Prinzip und „longops.js" ist eine Anwendung, die aus einer zweiten Session „v$session_longops" überwacht wird und bei langen Operationen eine Fortschrittsanzeige ausgibt.

Andere Script-Sprachen

Die Nashorn-Bibliothek des aktuellen Java 1.8 implementiert die bereits etwas in die Jahre gekommene JavaScript-Version „ECMA Script 5". Der aktuelle Stan-

```
script
// Schleife
var binds = {};
var a = 0, b = 1, f = 1, n=10, ret;
for(var i = 2; i <= n; i++) {
   f = a + b;
   a = b; b = f;
   binds.i = i;
   binds.f = f;
   util.execute( "insert into "+tabName
                 + "(indx,fibonacci) values(:i, :f)"
                 , binds);
}
sqlcl.setStmt( "commit; \n"
             + "select * from "+tabName+"; \n"
             );
sqlcl.run();
/
```

Listing 6

```
var HashMap = Java.type("java.util.HashMap");
var bindmap = new HashMap();

print("\nreading file: "+ args[1]);
var filePath=args[1];

var blob=conn.createBlob();
var bstream=blob.setBinaryStream(1);

java.nio.file.Files.copy( java.nio.file.FileSystems.getDefault().getPath(filePath)
                        , bstream );
bstream.flush();

bindmap.put("inhalt", blob);
bindmap.put("pfad", filePath);


if(!util.execute( "insert into dokument (datei_inhalt,datei_pfad, datum)
values(:inhalt, :pfad, sysdate)"
                 , bindmap)
   ){ print("insert failed, exiting");
      exit;
}
sqlcl.setStmt( "commit; \n"
             + "set sqlformat ansiconsole \n"
             + ' select datei_pfad "Dateipfad",dbms_lob.getlength(datei_in-
halt) "Größe", to_char(datum,\'DD.MM.YYYY HH24:MI:SS\') "Zeit" ,
             + "from dokument;");
sqlcl.run();
```

Listing 7

```
var blob=conn.createClob();
var bstream=blob.setAsciiStream(1);
```

Listing 8

```
script
users=[
   {"user" : "DATA", "password" : "E_DXxXxXe2r" }
,  {"user" : "DOOTZ","password" : "E_OXxXxX_uas" }
,  {"user" : "BARRY","password" : "Barry's Secret" }
];
/
```

Listing 9

```
script
for (var i=0; i < users.length; i++ ){
   ctx.write ("Testing Schema "+ users[i].user+"\n");
/
```

Listing 10

```
script
// Searching Password for user
function getPasswd(username){
   return users.filter(function ( obj ) {
                                        return obj.user === username;
                                        }
                      )[0].password;
}

// Using in Scripts:
var uname = "BARRY";
ctx.write( "The password of Schema " + uname + " is "
   + getPasswd(uname) + "\n");

/
```

Listing 11

```
script
/* ES6 Demo */
const user = util.executeReturnOneCol('select user from dual');

const a="Hello ECMA Script 6";
let b="const and let are new to ES6";

print(`ES 6 allows multiline strings
   enclosed in backticks and variables includes with "$"
   and curly braces ${a} ${b} User: ${user}`);
/
```

Listing 12

dard „ES6“ bringt eine Fülle von Verbesserungen, unter anderem die beiden neuen Schlüsselwörter „const“ und „let“ für die Variablendeklaration sowie Mehrzeilenzeichenketten *(siehe Listing 12)*.

Die Java-Version 9, die zurzeit als Preview verfügbar ist, bietet eine ES6-fähige Nashorn-Implementierung. Wenn man SQLcl mit Java 1.9 startet, wird das obige Beispiel ohne Fehler ausgeführt. SQLcl erkennt anhand der Dateiendung, dass ein Script in einer anderen Sprache als JavaScript geschrieben wurde. Wenn ein passendes JAR-File im Classpath zu finden ist, wird dieses anstelle von Nashorn als Interpreter verwendet. Erfolgreich ausprobiert hat der Autor das mit „lua“ (JVM Implementierung „luaj“) und „Python“ (Jython-standalone).

Fazit

Die Scripting-Funktion im neuen SQLcl hat großes Potenzial, die Arbeit von DBAs und Entwicklern zu erleichtern. Die Einstiegshürde ist gering. Es gibt noch viel zu entdecken und auszuprobieren. Die kompletten Beispiele dieses Artikels sind unter *„https://github.com/its-people/sqlclscripts“* zu finden.

Robert Marz
robert.marz@its-people.de