Nützen Sie die Gelegenheit zum Austausch mit Experten

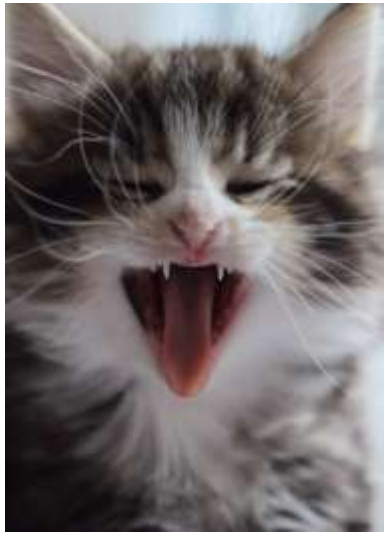
Die DOAG Datenbank wird **zweitägig**:

Am **30. und 31. Mai 2017** findet die nächste Veranstaltung für Oracle-Datenbankadministratoren statt.
Machen Sie sich auf ein vielfältiges Vortragsprogramm gefasst,
das tief unter die Haube der Oracle-Datenbank blickt
und zeitgleich Berufseinsteigern die Grundlagen der Administration vermittelt.

DOAG

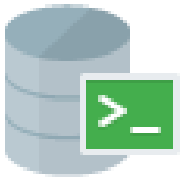
Deutsche ORACLE-Anwendergruppe e.V.

DOAG Konferenz 16.11.2017



SQLcl Quo vadis SQL*Plus?

Das neue SQL*Plus in der Praxis



Gunther Pippèrr - IT-Architekt - Berater

Background

Gunther Pippèrr arbeitet seit mehr als 17 Jahre intensiv mit den Produkten der Firma Oracle im Bereich Datenbanken/Applikationsserver und Dokumenten-Management.

Herr Pippèrr hat sich tiefes Wissen über den Aufbau komplexer IT Architektur aneignen können und hat dies in der Praxis erfolgreich umgesetzt.

Herr Pippèrr hat einen Abschluss als Dipl. Ing. Technische Informatik (FH) an der FH Weingarten.

Functional Expertise

- IT System Architekt
- Technische Projektleitung
- Design und Implementierung von Datenbank Anwendungen
- Entwurf und Umsetzung von IT Infrastrukturen zum Datenmanagement

Industry Expertise

- High-Tech
- Real Estate
- Utility
- Communications
- Pharm.

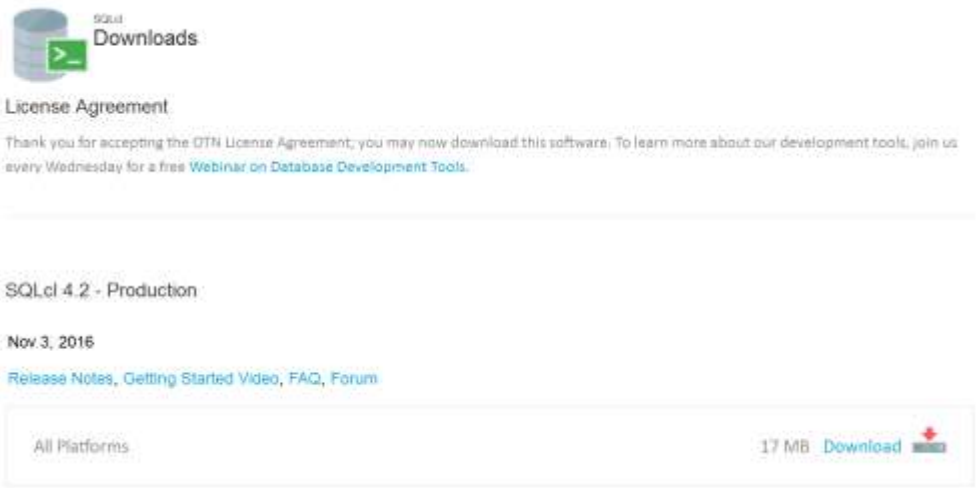
Selected Experience

- Datenbank Architekt für ein Projekt zur Massendatenverarbeitung in der Telekommunikation
- Architekt und technische Projektverantwortung für IT Infrastrukturprojekte, z.B.:
  - Unterstützung beim Betrieb der Datenbank Umgebung für das größte deutsche Kunden Bindungsprogramm
  - Zentrale Datenhaltung für eine Münchner Hotelgruppe mit über 25 Hotels weltweit
  - Messdaten Erfassung für russischen Kabelnetzbetreiber
  - Redundante Cluster Datenbank Infrastrukturen für diverse größere Web Anwendungen wie Fondplattform und Versicherungsportale
- Architekt und technische Projektverantwortung für ein Smart Metering Portal für das Erfassen von Energiezählerdaten und das Asset Management
- Architekt und Projektleitung , Datenbank Design und Umsetzung für die Auftragsverwaltung mit Steuerung von externen Mitarbeitern für den Sprachdienstleister von deutschen Technologiekonzern

2017 – Noch Freie Termine – Workshops / Coaching / Projektarbeit

Installation

http://www.oracle.com/technetwork/developer-tools/sqlcl/downloads/index.html



Download Zip File und auspacken z.B. nach $ORACLE_HOME/products

Problem
WARNING: Could not open/create prefs root node Software\JavaSoft\Prefs

```
PS C:\oracle\products\sqlcl\bin> .\sql.bat /nolog
Nov 04, 2016 12:46:16 PM java.util.prefs.WindowsPreferences <init>
WARNING: Could not open/create prefs root node Software\JavaSoft\Prefs at root 0x80000002. Windows RegCreateKeyEx(...) returned error code 5.

SQLcl: Release 4.2.0 Production on Fri Nov 04 12:46:16 2016

Copyright (c) 1982, 2016, Oracle.  All rights reserved.
```

Lösung: Key **HKEY_LOCAL_MACHINE\SOFTWARE\JavaSoft\Prefs** anlegen!

Lizenz??

- Kann frei heruntergeladen werden
- OTN Lizenz
- Lizenz an sich ist mir aber noch unklar!

Im Forum angefragt:

https://community.oracle.com/message/14100385

Übersicht

Demo

```
-- define the alias
alias ls=SELECT object_name FROM user_objects

--call
ls
```

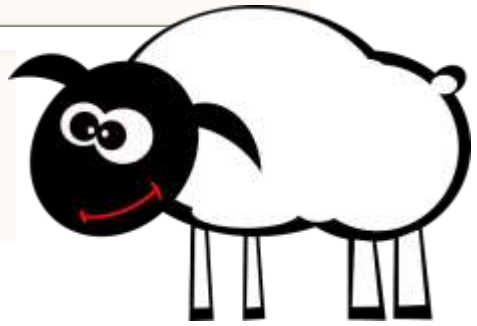
Mit “TAB” Kommando erweitern

```
SQL> ddl dept
  CREATE TABLE "SCOTT"."DEPT"
   (    "DEPTNO" NUMBER(2,0),
        "DNAME" VARCHAR2(14),
        "LOC" VARCHAR2(13),
   ....
```

Kommandozeilen Buffer unter Linux!

```
SELECT /*csv*/       * FROM employees;        -- Comma-separated values
SELECT /*delimited*/ * FROM employees;        -- (same as csv)
SELECT /*fixed*/     * FROM employees;        -- Fixed-width fields with trailing blanks
SELECT /*html*/      * FROM employees;        -- Marked-up HTML table
SELECT /*insert*/    * FROM employees;        -- SQL INSERT statements
SELECT /*loader*/    * FROM employees;        -- Pipe-delimited format suitable for SQL*Loader
SELECT /*text*/      * FROM employees;        -- Plain text
SELECT /*xml*/       * FROM employees;        -- Tagged XML
```

```
oerr ora 600
..
*Cause:    This IS the generic internal error NUMBER FOR Oracle program
...
```



Wie erkenne ich wo ich bin?

- SQL*Plus

```
SQL> define _SQLPLUS_RELEASE
DEFINE _SQLPLUS_RELEASE = "1201000200" (CHAR)
```

- SQLcl

```
SQL> define _SQLPLUS_RELEASE
DEFINE _SQLPLUS_RELEASE = "040200162601205" (CHAR)
```

```
select decode(substr('&&_SQLPLUS_RELEASE',0,1), '0','SQLCL','SQLPLUS') from dual;
```

Wo steht die History?

- Linux im User Home unter ~/.sqlcl
  - history.xml – Alle alten Befehle:

```
<historyItem>
   <original><![CDATA[create user gpi identified by tiger;]]></original>
   <timestamp><![CDATA[Nov 11, 2016]]></timestamp>
   <timesUsed><![CDATA[1]]></timesUsed>
   <timing><![CDATA[145]]></timing>
   <failure>false</failure>
</historyItem>
```

Und Sie lernen es nie ….

  - aliases.xml – alias Definitionen

- Windows unter C:\Users\<user>\AppData\Roaming\sqlcl

Dies Dateien sind mal wieder in keiner Weise geschützt!
Das SQL*Plus **login.sql** Problem wird aus der Vergangenheit mitübernommen!

Internet Zugriff auf Start

New version: 4.2.0.99.99 available to download

- Auf welche URL greift SQLcl zu:
  - oracle.sqldeveloper.sqlcl.jar siehe META-Inf/cfu.json

```
{
    "url": "http://www.oracle.com/webfolder/technetwork/sqldeveloper/sqlcl.json",
    "timeout": 5000
}
```

www.oracle.com/webfolder/technetwork/sqldeveloper/sqlcl.json

```
{ "sqlcl": {
        "cfu":
            [ "http://slc04qag.us.oracle.com/builds/sqlcl/sqlcl-cfu.json",
              "https://apex.oracle.com/pls/apex/dbtools/usage/sqlcl.json"
            ]
        }
}
```

Oracle Intern!

https://apex.oracle.com/pls/apex/dbtools/usage/sqlcl.json

{"version":"4.2.0.99.99","date_published":"21-MAY-2016","message":"Dont forget to attend Oracle OpenWorld September 18 to 22, 2016","message_date":"22-MAY-2016","message_end_date":"22-SEP-2016"}


```
{"version":"4.2.0.99.99"
,"date_published":"21-MAY-2016"
,"message":"Dont forget to attend Oracle OpenWorld September 18 to 22, 2016"
,"message_date":"22-MAY-2016",
"message_end_date":"22-SEP-2016"
}
```

Pro und Kontra

Pro

- Standalone lauffähig
- Praktische neue Funktionalitäten
  - Wie Aliase etc.
- Kommandozeilen Buffer unter Linux/Mac
- Integration von weiteren Skriptsprachen

Contra

- Java 8 notwendig, in gehosteten Umgebungen nicht so einfach verfügbar
- Massive UTF-8 / Umlaut Probleme unter Windows
- SQL*Plus Kommandos nicht zu 100% übernommen
- Immer noch recht viele kleine ärgerliche Bugs
- Security steht wohl nicht im Fokus
- Wenig echte Dokumentation

Umlaut /UTF8 Problem unter Windows (1)

- Leider liegt hier wohl ein generelles Problem mit Java und Standard Out in der DOS / Power Shell Konsole mit http://jline.sourceforge.net/ vor

Eingabe von Umlauten schwierig:

```
SQL> select * from emp where ename='□□□□';
```

Probleme mit dem Print Out von Umlauten aus Script Sprachen

```
SQL> script helloWorld.py
Hello World!
Viele Gr□?¼□?Ÿe aus M□?¼nchen
```

Sehr ärgerlich im 21' Jahrhundert der IT…

Ständig nachfragen und nörgeln auf:
https://community.oracle.com/community/database/developer-tools/sql_developer/sqlcl

Umlaut /UTF8 Problem unter Windows (2)

- Leichte Verbesserung mit Patch der sql.bat auf Zeichensatz 1252:

```
32 REM chcp 65001 >nul 2>&1
33 chcp 1252
34 set NLS_LANG=AMERICAN_AMERICA.WE8MSWIN1252
```

- Verwendung Format ANSICONSOLE

```
set SQLFORMAT ANSICONSOLE
```

```
SCOTT@GPI-saturn>select 'Grüße aus Nürnberg' as "ANSI ÖÄÜ" from dual;
ANSI ÖÄÜ
Grüße aus Nürnberg
```

Nachteil: Formatangaben mit COLUMN werden nicht mehr unterstützt!

JAVA Lang Variable setzen

- JAVA_LANG bestimmt die Einstellungen
- JAVA_TOOLS_OPTIONS setzen

set JAVA_TOOL_OPTIONS='-Duser.language=en -Duser.region=US -Dfile.encoding=UTF-8'

```
Neu in aktueller Release:

help set encoding

SET ENCODING
  SET ENCODING { UTF8,GBK,...}


show encodings
```

Scripting mit SQLcl

- Klassisch – Aufruf mit “@”
  - SQL Skripte wie in SQL*Plus

- Integration JavaScript - Aufruf mit “script”
  - Aufruf von JavaScript als Skriptsprache
    - Auch innerhalb von SQL Skripten kombinierbar

- Integration von weiteren Skriptsprachen nach der Java JSR-223 Spezifikation - Aufruf mit “script <dateiame>.xx” , wie Jython , Ruby, Lua etc.
  - Script Interpreter Klasse muss im Klassenpfad stehen
  - Interpreter wird über die Datei Endung des Scripts ausgewält
    - *.js => integrierter JS Nashorn Interpreter von Oracle
    - *.py => Jython Interpreter aus der “jython-standalone-2.7.0.jar”

Welche Scriptsprache wählen?

- Default JS oder eigener Interpreter?

Was spricht für JS?

- Einfacher in SQL Script integrierbar
- Besser von Oracle getestet
- Mehr Support bei Problemen

Was spricht für Jython

- Was für Python Liebhaber

Scripting mit SQLcl mit JavaScript

- Default – JS Interpreter **Oracle Nashorn**
- Beispiel:

```
SQL> script
  2   ctx.write("Hello Word\n");
  3  /
Hello Word
```

Default: Java Script

- Die folgende Objekte bieten den Durchgriff auf die QLcl Session an:
  - **sqlcl** - SQLCL selbst
    - Befehl setzen mit sqlcl.setStmt(„select * from dual“) und mit sqlcl.run() auszuführen
  - **ctx** - Objekt vom Typ ScriptContext
    - Kann direkt angesprochen werden - wie „ctx.write(„String“)“
  - **util** – Hilfsklasse für komplexere Aufgaben
    - Wie „var user=util.executeReturnOneCol('select user from dual');“
  - **conn** – Connection Object
    - Wie “conn.getMetaData().getURL()”

Vorteil: kann über “script <code> /” direkt in der Konsole aufgerufen werden!

SQLcl Commands erweiteren/anpassen

- Über die CommandRegistry und den CommandListener läßt sich die Verarbeitung im SQLcl anpassen

JS Script addTimeListener.js

Laufzeit am Ende
eines SQL's mit ausgeben

```
// Refernz to the SQLcl Command Registry Class
var CommandRegistry = Java.type("oracle.dbtools.raptor.newscriptrunner.CommandRegistry");

// CommandListener for creating new any command Listener
var CommandListener =  Java.type("oracle.dbtools.raptor.newscriptrunner.CommandListener")

// Java Script object to hold the new methodes
var extCommand = {};

// Called to attempt to handle any command
extCommand.handleEvent = function (conn,ctx,sqlcl) {
    // return FALSE to indicate the command was not handled
    // now other commandListeners can handle this command
   return false;
}

// fired before ANY command
extCommand.beginEvent = function (conn,ctx,sqlcl) {
                    //  Get the actual command
    if ( sqlcl.getSql().indexOf("showTime") > 0 ){
                                    ctx.putProperty("gpi.showTiming",true);
                        ctx.write("\n -- Debug sqlcl: " + sqlcl.getSql()+ " Set timing on true \n");
                                    //remember the time
                                    ctx.putProperty("gpi.startTiming",new Date());
                    }
}

// fired after ANY Command
extCommand.endEvent = function (conn,ctx,sqlcl) {
   if ( ctx.getProperty("gpi.showTiming") ) {
                      var end = new Date().getTime();
                      var start = ctx.getProperty("gpi.startTiming");
      start = start.getTime();
                      // print out elapsed time of all commands
      ctx.write("Command elapsed time :: " + (end - start) + " ms \n");
                      //unset
                      ctx.putProperty("gpi.showTiming",false);
   }
}

// Actual Extend of the Java CommandListener

var ShowTimingCmd = Java.extend(CommandListener, {
                                    handleEvent: extCommand.handleEvent ,
        beginEvent:  extCommand.beginEvent  ,
        endEvent:    extCommand.endEvent
});

// Registering the new Command
CommandRegistry.addForAllStmtsListener(ShowTimingCmd.class);
```

Abgeleitet aus:
https://github.com/oracle/oracle-db-tools/blob/master/sqlcl/examples/customCommand.js

Scripting mit SQLcl mit Python in Jython

- Download von jython-standalone-2.7.0.jar von http://www.jython.org

- Einbinden in den Klassenpfad von SQLcl
  - Unter Windows über die Start Datei sql.bat

```
53
54 set CPFILE=%CPFILE%;%SQL_HOME%\lib\jython-standalone-2.7.0.jar
55
```

  - Linux

```
export CLASSPATH=$CLASSPATH:./jython-standalone-2.7.0.jar
```

Eingebunden über die die Java JSR-223 Spezifikation

Jython Project => http://www.jython.org

Der erste Fehlschlag .....

Jython Script:

```
1
2 print "Hello World!"
```

Aufruf mit @

```
SQL> @helloWorld.py
```

Nichts tut sich .....

Aufruf mit “script”

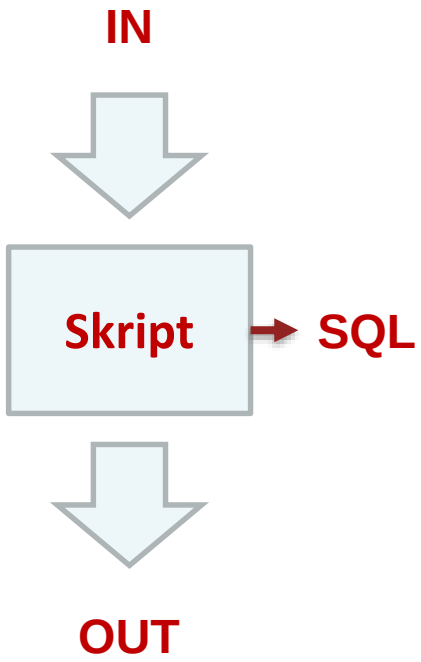
```
SQL> script helloWorld.py
Hello World!
```

Funktioniert!

@ ruft nur SQL Scripte auf!

Welche Scripting Pattern brauchen wir?

- Laufzeit-Umgebung erkennen
- Argumente/Parameter an ein Skript übergeben
  - Beim Aufruf
  - SQLcl Variablen im Script verwenden
- Auf die SQLcl Runtime Umgebung wie die DB Session zugreifen
- Rückgabe Werte wieder in SQLcl weiter verarbeiten
- SQL Kommandos im Skript ausführen
- Reine SQL Skripte nach bestimmten Regeln aufrufen
- SQLcl Commandos erweitern



Laufzeit Umgebung

- Mit Jython Standard Methode verwenden

```
import os
import platform
import time

print "Actual Path ::" , os.getcwd()
print "Home Drive  ::" , os.getenv('HOMEDRIVE')
print "Home Path   ::" , os.getenv('HOMEPATH')
print "Plattform   ::" , platform.platform()

print "Zeit        ::" , time.strftime('%Y%m%d%H%M%S')
```

getActDirectory.py

```
SQL> script getActDirectory.py
Actual Path :: C:\oracle\products\sqlcl\bin
Home Drive  :: C:
Home Path   :: \Users\gpipperr
Plattform   :: Java-1.8.0_111-Java_HotSpot-TM-_64-Bit_Server_VM,_25.111-b14,_Oracle_Corporation-on-Windows_10-10.0-amd64
Zeit        :: 20161107112340
```

Direkt auf die Methoden von SQLcl zugreifen

- Auch innerhalb des Scopes des Scripts sind die “äußern” Instanzen der SQLcl Klassen erreichbar und können direkt verwendet werden

- **ctx** - Objekt vom Typ ScriptContext
  - kann direkt angesprochen werden - wie ctx.write(„String“)
- **sqlcl** - SQLCL selbst
  - sqlcl.setStmt(„select * from dual“) – Befehl setzen
  - sqlcl.run() – Befehl auszuführen
- **util** – Hilfsklasse für komplexere Aufgaben
  - wie var user=util.executeReturnOneCol('select user from dual'); um eine Zeile von der Datenbank zu holen

Auch innerhalb des Jython Scripts gültig!

Java importieren - im Jython Script verwenden

- Importieren mit

```
from java.lang import System as javaSystem



# get the Java Enviroment Setting of this session
# and use the Python str function inside the Java Part !
#

javaSystem.out.println( "user.lang    ::" +
       str(javaSystem.getProperty("user.lang"))
);

javaSystem.out.println( "user.country ::" +
      str(javaSystem.getProperty("user.country"))
);
```

useJava.py

SQLcl – Password abfragen

- Referenz auf die Eingabe von SQLcl holen

```
from java.lang import Character

# show this sign a replacement for the userimport
param2=Character('*')

reader = ctx.getProperty("script.runner.jline");

passwd = reader.readLine(None, param2);

print " \n Get the Password:" + passwd
```

setPassword.py

```
sql>script showPassword.py
******
 Get the Password:123456
```

ctx - ScriptContext für Ausgaben verwenden

- Mit dem Context Objekt eine Ausgabe auf Standard Out erzeugen:

```
# use Context Class
ctx.write("Hello World")
```

- Die Session abfragen:

```
print "Connection URL       :: " +
str(ctx.getProperty("cli.conn.url"))
print "The actual connection:: " +
str(ctx.getProperty("cli.conn.props"))
```

showSQLclEnv.py

```
Connection URL       :: jdbc:oracle:oci8:@(DESCRIPTION =    (ADDRESS = (PROTOCOL = TCP)(HOST = 10.10.10.1)(PORT = 1521)
    (CONNECT_DATA =      (SERVER = DEDICATED)      (SERVICE_NAME = GPI)    )  )/n
The actual connection:: {user: scott, password: tiger, v$session.program: SQLcl}/n
```

Der nächste Fehlschlag .....

- Script:

```
print "1 Hello World!"

ctx.write("2 Hello World mit der Referenz auf ctx")

print "3 Viele Grüße aus Nürnberg"

ctx.write("4 Viele Grüße aus Nürnberg mit der Referenz auf ctx")
```

helloWorld.py

- Ergebnis:

```
SQL> script helloWorld.py
1 Hello World!
3 Viele Gr□?¼□?Ÿe aus N□?¼rnberg
```

Lösung ctx.write mit “\n” verwenden

- Lösung: “**\n**” immer bei Verwendung von ctx.write am Ende verwenden!
- ABER => unterschiedliche Output Streams!

```
print     " 1 Hello World!"
ctx.write(" 2 Hello World mit der Referenz auf ctx\n")
print     " 3 Viele Grüße aus Nürnberg"
ctx.write(" 4 Viele Grüße aus Nürnberg Referenz auf ctx\n")
```

helloWorld.py

```
SQL> script helloWorld.py
1 Hello World!
3 Viele Gr□?¼□?Ÿe aus N□?¼rnberg
2 Hello World mit der referenz auf ctx
4 Viele GrÃ□Â¼Ã□Å¸e aus NÃ□Â¼rnberg mit der referenz auf ctx
```

Argumente-/Parameterübergabe

- JS - **`args`**

```
SQLcl>script inParameter.js  a b c d
```

```
for(var i=0;i<args.length;i++){
        ctx.write(args[i] + "\n\n");
}
```

- Python - **`sys.argv`**

```
SQLcl>script inParameter.py a b c d
```

```
import sys
print "Number of arguments:", len(sys.argv), "arguments"
print " Argument List: ", str(sys.argv)

for arg in sys.argv:
        print arg
```

Funktioniert noch nicht!
BUG?

Mit diskutieren unter
https://community.oracle.com/thread/3988096

Wieder etwas seltsames- __name__ Verhalten

- Problem: main function wird nicht gerufen

```
print "Outside main"

def main():
            print "Inside main"

if __name__ == '__main__':
            main()
```

callMainV1.py

```
sql>script callMainV1.py
Outside main
sql>h
```

Wieder etwas seltsames - __name__ Verhalten

- Lösung:

```
mainTest.py

print "Outside main 1."
print __name__

def main():
        print "Inside main"

if (__name__ == '__main__') or (__name__ == '__builtin__'):
        main()

print "Outside main 2."
```

callMainV2.py

```
sql>script calMainv2.py
Outside main
__builtin__
Inside main
```

SQLcl Bind- und Substitutionsvariablen

- SQLcl
  - Echte Bind Variablen mit ":"

Gleiche Logik wie in SQL*Plus

```
-- definieren
Sqlcl> variable bindVariable varchar2(20)
-- Wert setzen
Sqlcl> begin :bindVariable:='HUGO; end;
/
-- ausgeben
Sqlcl>print bindVariable
BINDVARIABLE
------------
HUGO
```

  - Reine Textersetzung mit "&"

```
SQL> define subVariable=HUGO

SQL> prompt &&subVariable
HUGO
```

Ergebnisse wieder in SQLcl zur Verfügung stellen

Alternativ über spool files

- sqlcl.setStmt

```
Sql> variable bindVariable varchar2(20)

Sql>script setVariables.py

sqlcl.setStmt("define subVariable=HUGO");
sqlcl.run();

sqlcl.setStmt("begin :bindVariable :='HUGO'; end;");
sqlcl.run();


Sql>define subVariable
DEFINE subVariable = "HUGO" (CHAR)

Sql>print bindVariable

RETURNRESULT
----------------------
HUGO
```

setVariables.sql

Statement mit SQLcl Klassen ausführen

- Ein einfaches Statement in SQLcl ausführen und an Standard out ausgeben:

Setzen

Ausführen

```
#Execute a statement
sqlcl.setStmt("select * from dual");


sqlcl.run();
```

```
SQL> script
  2  sqlcl.setStmt("select * from dual");
  3  sqlcl.run();
  4  /

D
-
X
```

Util Objekt - SQL Befehl absetzen

- Einen Wert von der DB holen
- executeReturnOneCol(<string>,binds)
  - executes und returns die erste Zeile und die ersten Spalte

```
# Use Util class
dbUser = util.executeReturnOneCol('select user from dual');

print "Actual DB Use is: ",dbUser
```

Util Objekt - executeReturnListofList

- executeReturnListofList(<string>,binds)
  - executes and returns an array(rows) of arrays(row).

- Resultset aus SQL ermitteln

```
sql='select EMPNO,ENAME,SAL from emp where sal < 1000';

retVal = util.executeReturnListofList(sql,None);

for e in retVal:
        print e
```

Util Objekt - SQL mit Bind Variablen ausführen

- In Jython Java Objekt erzeugen und für die Bind Variablen Übergabe verwenden

Importieren

Anlegen

Verwenden

```
from java.util import HashMap

# create object for the bind variable
bind=HashMap()
bind['EMPNO'] = '7369'

# SQL statement with bind usage
sql='select EMPNO,ENAME,SAL from emp where EMPNO=:EMPNO'

# execute the statment and iterate over the result
retVal = util.executeReturnListofList(sql,bind)

# interate over the result
for e in retVal:
        print e
```

useBindVariable.py

SQL*Plus – Script je nach Bedarf aufrufen

- Je nachdem ein Script in SQL*Plus aufrufen

```
variable OS varchar2(20):='LINUX'

--- define  this variable to avoid error with nolog logins
define SCRIPTPART_CALL='set_no_titel.sql'

col SCRIPTPART_COL new_val SCRIPTPART_CALL

select decode (:OS, 'LINUX,
                  , 'set_linux_title.sql',
'set_windows_title.sql')
as SCRIPTPART_COL from dual
/

-- call the os script for the title setting
--
@@&&SCRIPTPART_CALL "&z" "&y„

undefine OS
```

SQL*Plus Syntax

SQLcl - Skript je nach Bedarf aufrufen

- Mit **sqlcl.setStmt** lässt sich alles aufrufen, was auch über die Kommandozeile möglich ist
  - **Beispiel:** SQL Script je nach Bedarf aufrufen

```
OS='LINUX'

if OS == 'LINUX':
        sqlcl.setStmt('@set_linux_title.sql')
else:
        sqlcl.setStmt('@set_windows_title.sql')

sqlcl.run()
```

Jython Syntax

SQLcl in Jython verwenden

- Jar files im Klassenpfad einbinden
- SQLcl Klassen importieren
- Connection auf die DB anlegen
- Referenz auf den ScriptRunner holen

```
#import various things
from java.sql import DriverManager
from oracle.dbtools.db import DBUtil
from oracle.dbtools.raptor.newscriptrunner import *
#plain ol jdbc connection
conn  =
DriverManager.getConnection('jdbc;oracle;thin;@//10.10.10.1;1521/GPI','scott','tig
er');
#get a DBUtil for later if needed
util  = DBUtil.getInstance(conn);
#create sqlcl
sqlcl = ScriptExecutor(conn);
#setup the context
ctx = ScriptRunnerContext()
#set the context
sqlcl.setScriptRunnerContext(ctx)
ctx.setBaseConnection(conn);
#change the format
sqlcl.setStmt('set sqlformat json');
sqlcl.run();
#run the sql
sqlcl.setStmt('select * from emp');
sqlcl.run();
```

callSQLclFromJython.py

Siehe https://github.com/oracle/oracle-db-tools/blob/master/sqlcl/examples/jython_example.sh

Welches Werkzeug past wo am besten?

- Pure SQL Scripts

- Nachteil
  - Logik in den Skripten oft nur sehr umständlich integrierbar

- Vorteil
  - Geringer Einarbeitungsaufwand

- Pure Python Script

- Nachteil:
  - cx_Oracle Lib nicht so einfach im RZ installier bar da ein echter Oracle Client notwendig ist!

- Vorteil
  - Noch komplexere Skripte möglich

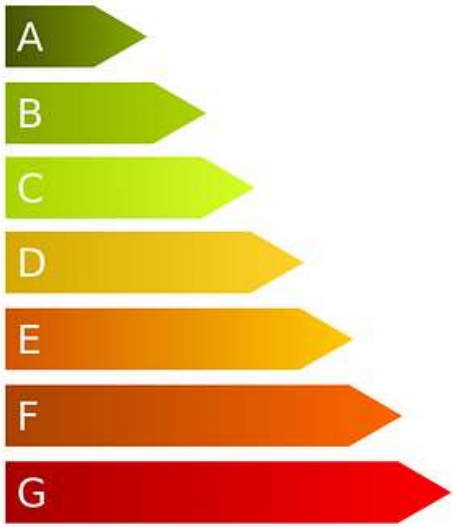
Summary

- SQLcl hat viele neue und interessante Features
- Lässt sich gut mit der eigenen Software ausrollen, da keine Client Installation notwendig
  - Lizenz noch mit Oracle Vertrieb prüfen!!!
- Umlaut/UTF-8 Problem unter Windows sehr ärgerlich
  - Über das Forum nerven, damit das Problem auch ernst genommen wird!
- Java 8 auf gehosteten Umgebungen nicht so einfach verfügbar, da noch nicht inkl. mit der Cluster oder DB 12c Installation als Standard dabei!
  - Sehr schade, ein eigener Installer muss dann immer mit einer JRE ausgeliefert werden!

Ihre Meinung

- Werden Sie von SQL*Plus umsteigen?

- Ihre Meinung?

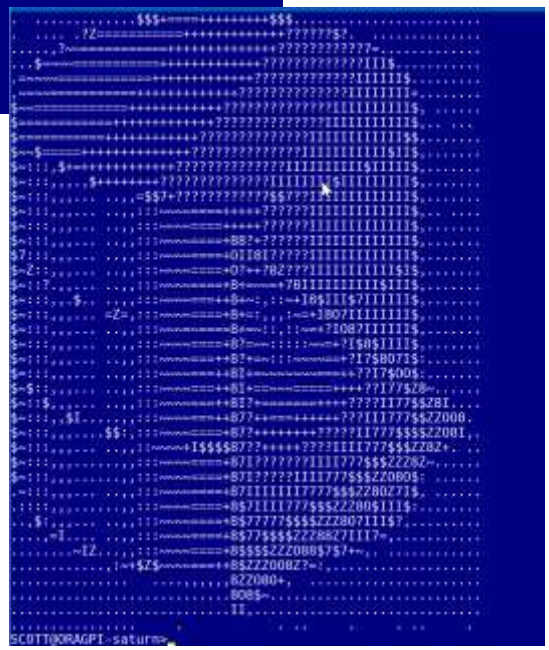


Mein Fazit:
Durchaus Wert sich damit intensiver zu beschäftigen!

Show sqldev

- show sqldev2

- show sqldev



Quellen

- Oracle Forum =>
- https://www.pipperr.de/dokuwiki/doku.php

- Wieder mal eine andere Script Library
  - https://orapowershell.codeplex.com/

- Bildmaterial : https://pixabay.com und https://openclipart.org/

2017 – Noch Freie Termine – Workshops / Coaching / Projektarbeit