Tipps und Tricks zur Erstellung von SQL*Plus Installationsscripten

Bereitstellung von Hilfsutilities für oft benötigte Mechanismen

Dr. Kurt Franke

Cellent Finance Solutions AG

Kurt.Franke@cellent-fs.de, Kurt-Franke@web.de

Agenda

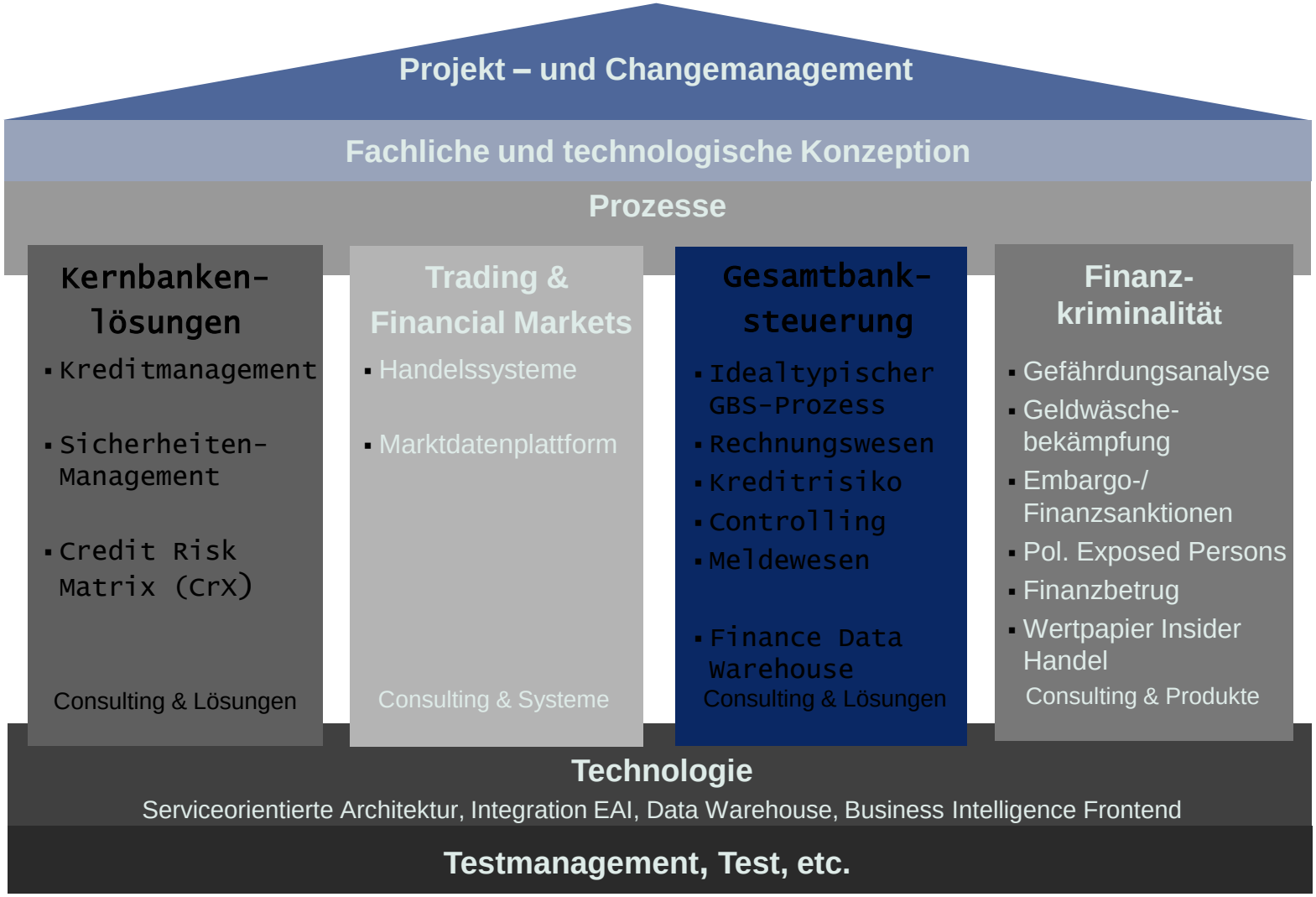
- cfs Kurzvorstellung
- SQL*Plus Grenzen der Ablaufsteuerung
- Anforderungen an Installationsscripte
- Einsetzbare SQL*Plus-Features
- Informationstransfer
  - aus SQL*Plus-Ebene heraus und zurück
- Voraussetzungen
- Beispiele für Hilfs-Utilities
- Zusammenfassung

Cellent Finance Solutions auf einen Blick

Wissen – Können - Machen

Firmensitz:	Stuttgart
Marktspräsenz:	seit 1985
Aktionärsstruktur:	100% Landesbank Baden-Württemberg (LBBW)
Aufsichtsrat:	Rudolf Zipf (Finanz-Vorstand der LBBW)
Anzahl Mitarbeiter:	200
Standorte:	Stuttgart und München
Umsatz:	€ 27 Mio.

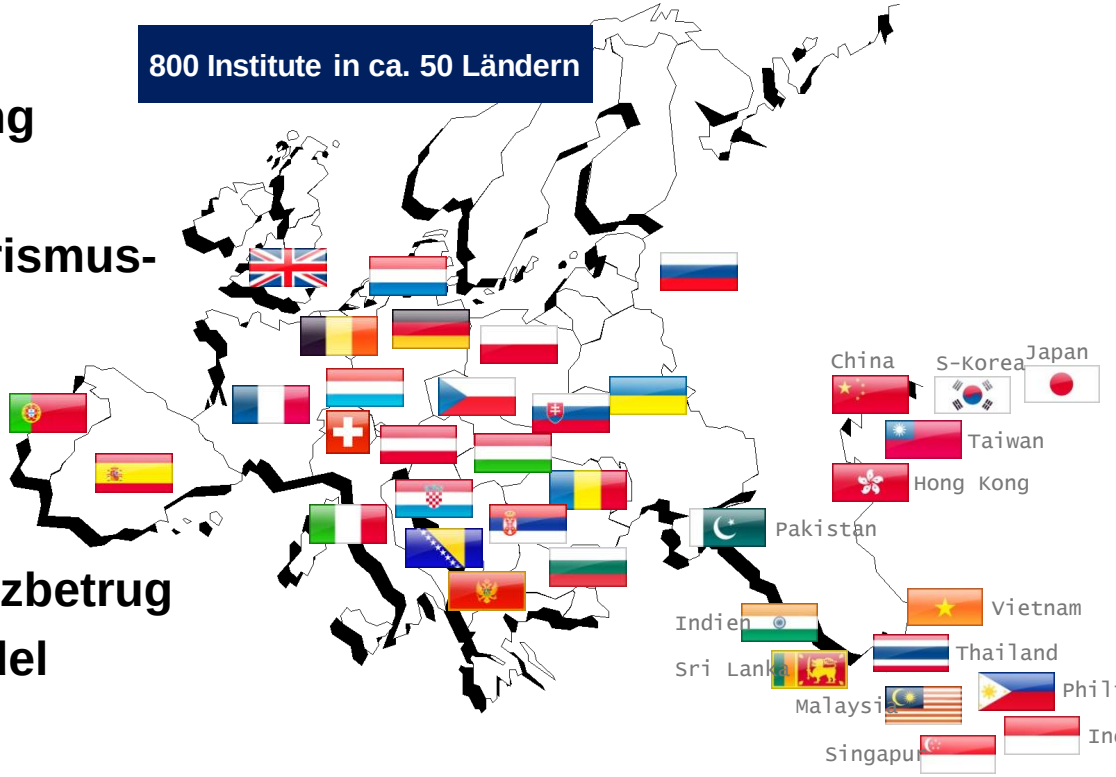
Das Angebot der Cellent Finance Solutions



Bekämpfung von Finanzkriminalität

Kundenstamm von mehr als 800 Instituten in 50 Ländern

- **Geldwäschebekämpfung**
- **Gefährdungsanalyse**
- **Bekämpfung der Terrorismus-finanzierung**
- **Embargoüberwachung**
- **Kundenscreening**
- **PEP-Identifikation**
- **Bekämpfung von Finanzbetrug**
- **Wertpapier-Insiderhandel**



Referenzen SMARAGD Anti-Finance Crime Suite (Auszug)

SQL*Plus Grenzen der Ablaufsteuerung

- keine vollwertige SQL*Plus Kommandosprache
  - wie z. B. in Unix Shell
  - mit Kontroll-Elementen (Bedingungen, Schleifen, …)
- Verwendung Syntax aus direct aufrufbaren Ebenen
  - SQL
  - PL / SQL
  - host (proprietär)
- Informationstransfer
  - SQL*Plus ⇨ Syntaxebene ⇨ SQL*Plus
    - Hintransfer
      - Kommando oder Statement
    - Rücktransfer
      - abgefragte Information
      - Rückmeldung OK oder Fehler

Anforderungen an Installationsscripte I

- ein Scriptsatz für alle unterstützten Platformen
  - ggf. einige wenige OS-spezifische Scripte
- Bestimmung des lokalen Operatingsystems
  - zur Auswahl geeigneter OS-spezifischer Aktionen
  - Aufruf der richtigen OS-spezifischen Scripte
- Hereinholen von Environment-Variablen
  - insbesondere NLS_LANG
    - ➢ zur Prüfung der Zeichensatzeinstellung
      - – für korrektes Einfügen von Daten aus den Scripten in die DB
      - – Einstellung muss zu Codierung passen
- Prüfung von Directory-Schreibrechten
  - zum Anlegen von Log-Dateien
  - zur Erstellung temporärer SQL-Scripte

Anforderungen an Installationsscripte II

- Unterdrückung von Fehlermeldungen
  - bei aufgerufenen Scripten
    - irrelevante Fehler
      - z. B. ORA-00942 bei DROP TABLE
    - später gehandelte Fehler
      - z. B. Compilefehler wg. noch nicht auflösbarer Abhängigkeiten
  - Debugmode mit Fehlerausgabe für Entwickler
- Spooling in mehrere Dateien
  - z. B. spooling in dynamisches SQL-Script bei laufendem Logging
  - paralleles Spooling müsste in SQL*Plus implementiert werden
  - deshalb zumindest automatisches Switch-Back
    - in vorhergehende Spooldatei
    - nach Beenden des aktuellen Spoolvorgangs
    - ohne Spooldateien als Parameter an gesamte Script-Aufruf-Hierarchie

Anforderungen an Installationsscripte III

- bedingter Scriptaufruf
  - über Callscript mit Condition-Parameter
  - optional mit Alternativ-Script (ELSE-Zweig)
- Abbruchscript mit bedingtem Aufruf
  - muss auch ohne Schreibzugriff auf Dateien funktionieren
    - ➢ für Abbruch bei fehlendem Schreibzugriff
  - deshalb kein internes spooling und store möglich
  - Aktion darf nur im Fehlerfall ausgeführt werden
  - deshalb spezielles IF-Script zum Aufruf
    - ➢ mit den gleichen Einschränkungen: kein Schreibzugriff
  - mit Debugmode ohne Abbruch zur Variablensichtung

Einsetzbare SQL*Plus-Features I

- host Kommando
  - alles, was Operatingsystem-Shell bietet
  - platform-spezifisch
  - keine allgemeine Implementierung möglich
  - sinnvoller Einsatz nur für:
    - Durchführung Aktionen auf OS-Ebene
      - z. B. Löschen von Dateien
    - Informations-Abfrage von OS-Ebene
      - Bestimmung des OS-Typs
      - Abfrage von Environment-Variablen

Einsetzbare SQL*Plus-Features II

- spool Kommando
  - Mitschreiben Ausgabe in Datei
  - nur eine Datei auf einmal
  - Öffnen einer neuen Datei schließt automatisch die bisherige
  - Steuerung der Ausgabe über set-Variablen (echo, termout, …)
- store Kommando
  - Speichern der aktuellen Einstellungen (SET-Variablen)
    - ➢ als set-Kommandos
  - Wiederherstellen nach beliegigen Änderungen
    - ➢ durch Aufrufen des gespeicherten Datei

Einsetzbare SQL*Plus-Features III

- start Kommando
  - Aufruf von SQL-Scripten
    - gesucht wird in aktuellem Verzeichnis und SQLPATH
  - @ ist Synonym des start-Kommandos
    - mit oder ohne Blank-Trennung zum Script
  - @@ wie @, aber Scriptsuche nur im gleichen Verzeichnis
    - wie aufrufendes Script
  - maximaler Nesting-Level 20
    - unter Windows früher nur 5
  - Scriptname in define Variable
    - ermöglicht dynamische Bestimmung des auszuführenden Scripts
      - ohne Verwendung von spool-Dateien

Einsetzbare SQL*Plus-Features IV

- whenever Klauseln
  - whenever sqlerror
    - Definieren Aktion bei SQL-Fehler (nicht bei SQL*Plus-Fehler)
  - whenever oserror
    - Definieren Aktion bei OS-Fehler (z. B. beim Schreiben spool-Datei)
  - mögliche Aktionen
    - EXIT
      - Exit-Value: Konstante oder Zahl oder Bindvariable
      - Transactionsverhalten: COMMIT oder ROLLBACK
      - verläßt SQL*Plus komplett
        - keine weiteren Aktionen möglich
    - CONTINUE
      - Transactionsverhalten: COMMIT oder ROLLBACK
      - ermöglicht nachfolgendes Handling
        - Bestimmung COMMIT oder ROLLBACK durch SELECT
    - Aktion kann in define-Variable stehen

Einsetzbare SQL*Plus-Features V

- define Variablen
  - ähnlich Shell-Variablen
  - fast überall in Scripten einsetzbar
    - außer für Werte-Rückgabe aus PL/SQL
    - Variablenreferenz wird durch Wert ersetzt
  - auch Parameter werden define-Variablen
  - Verwendung nicht definierter Variablen vermeiden
    - sonst erfolgt interactive Abfrage
    - einmalig mit automatischem define bei &&varname
    - wiederholt ohne define bei &varname
  - define ohne Zuweisung gibt Variable aus
    - als define Zuweisung
    - keine interactive Rückfrage bei undefinierter Variablen
    - ermöglicht Default-Settings für undefinierte Variablen
    - spool -> define -> spool off > define defaultwert -> @spoolfile

Einsetzbare SQL*Plus-Features VI

- define Variablen (Fortsetzung)
  - define Zuweisungsziel kann in anderer define-Variablen stehen
    - ➔ Name für Rückgabe-Variable als Parameter möglich
  - Wert kann via SELECT gesetzt werden
    - mit column Klausel
      - column mycolumn noprint new_value myvar
        - SELECT mycolumn befüllt myvar
        - keine Ausgabe wegen noprint Klausel
    - mehrere Columns / Variablen in einem SELECT
    - bei mehreren zuückgegebenen Rows
      - Werte aus zuletzt gelieferter Row stehen in Variablen
  - maximal 2048 define Variablen
    - bei Überschreitung Fehlerhandling kaum möglich (SP2-0138)
    - ➔ Erreichen Maximal-Anzahl vermeiden
      - Vorsicht bei Pseudo-Arrays (myvar_01, myvar_02 , …)
    - undefine nicht mehr benötigter Variablen

Einsetzbare SQL*Plus-Features VII

- Bind-Variablen
  - nur in SQL , PL / SQL und print Kommando verwendbar
    - nicht allgemein an beliebigen Scriptstellen
    - print für normale Bind-Variable (nicht REF CURSOR)
      - SELECT :var AS var FROM dual;
  - werden benötigt zur Rückgabe von Werten aus PL/SQL
  - danach Wert als 2. Schritt via SELECT in define Variable
    - wenn allgemeine Verwendung erforderlich
  - Werte-Übergabe zwischen SQL-Statements
    - nicht beschränkt auf Stringlänge 240 wie bei define Variablen
    - auch CLOB möglich
  - wiederholte Werte-Übergabe an SQL-Statements
    - mit verändertem Inhalt
      - besser zu handeln als define Variable

Einsetzbare SQL*Plus-Features VIII

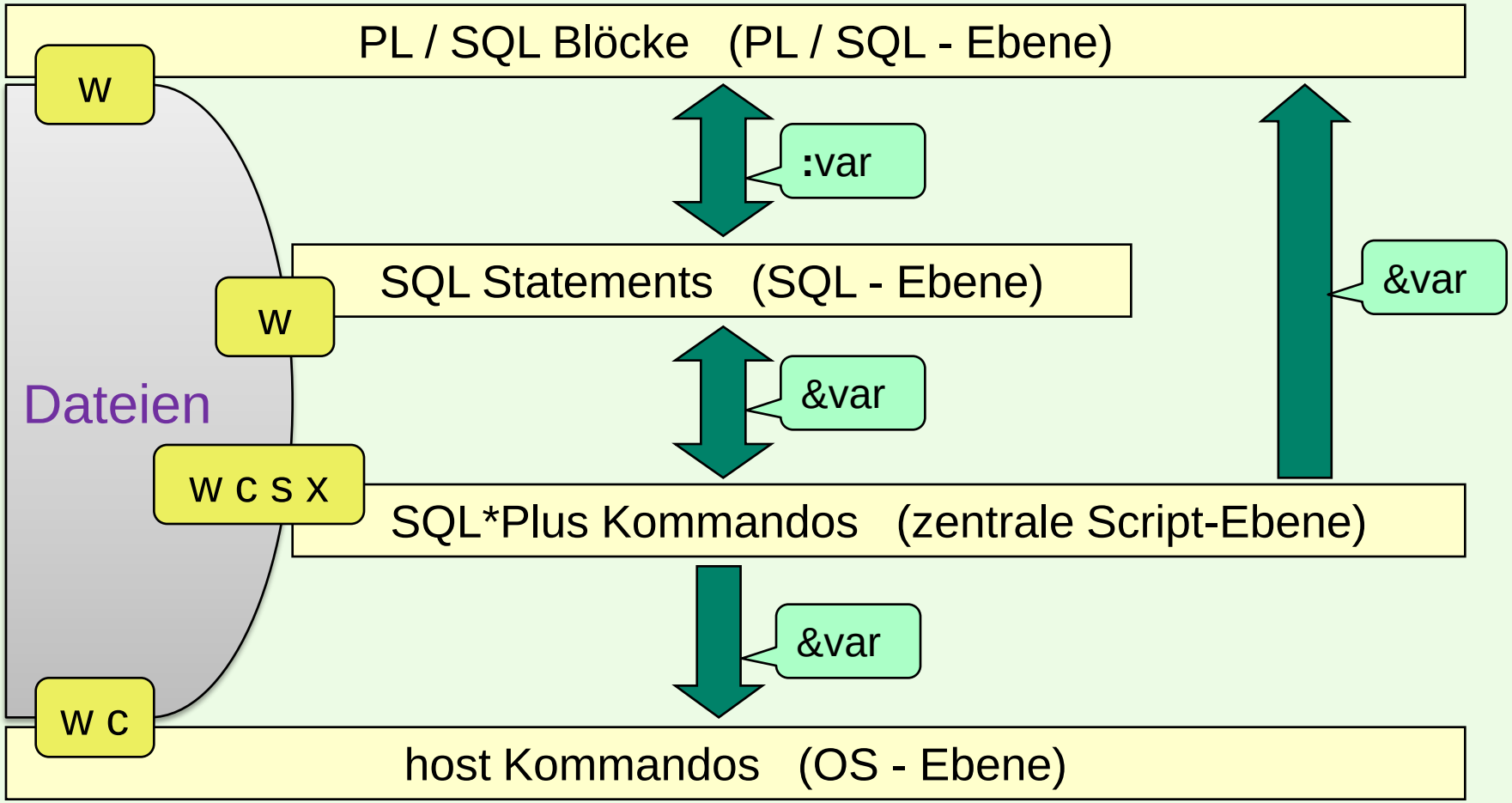
- SQL-Buffer
  - kann ein beliebiges SQL-Statement aufnehmen und ausführen
    - aber keine SQL*Plus Kommandos
  - Statement kann zur Laufzeit dynamisch erzeugt werden
  - ➔ dynamisch erstellte SELECT‘s mit Ausgabe möglich
    - ohne Erzeugung einer spool-Datei
  - define Variablen können dabei eingesetzt werden
    - beim Schreiben in den SQL-Buffer keine Ersetzung
    - erst beim nachfolgenden Ausführen Ersetzung mit aktuellem Wert
    - auch als erstes Wort einer Zeile (sonst nicht möglich)
    - ermöglicht Abfolge ähnlicher Statements
  - Befüllen mit
    - SQL*Plus Kommandos input und append
  - Leeren mit
    - SQL*Plus Kommando „del 1 last“ oder gezielt einzelne Zeilen

Einsetzbare SQL*Plus-Features IX

- SQL-Buffer (Fortsetzung)
  - Speichern mit Save Kommando
    - unabhängig von spool-Datei
    - es wird immer eine Zeile mit „/“ angehängt
    - define Variablen werden auch beim Speichern nicht substituiert
      - ➔ Fehler beim Aufruf wenn Variable als erstes Wort
    - Für reines Speichern und Ausführen als Script
      - auch SQL*Plus-Kommandos möglich
      - oder mehrere SQL-Statements
      - ggf. Handling vorsehen für letzte Zeile mit „/“
      - z. B. als letztes Kommando „del 1 last“ einfügen
        - ➔ SP2-0103: SQL-Puffer enthält keine auszuführenden Befehle
        - kein echter Fehler, kann ignoriert werden
    - ermöglicht Erzeugung Script-Datei ohne spool
      - als Ausweg, wenn spool nicht unterbrochen werden darf

Informationstransfer zwischen Arbeitsebenen

- Arbeitsebenen in SQL-Scripten



Informationstransfer zwischen Arbeitsebenen

- zwischen allen Ebenen in beide Richtungen
  - mittels SQL-Script-Dateien
  - erzeugt durch Ausgabe-spooling , save , OS-Kommandos
  - Einschränkungen:
    - ➢ SQL-Script-Aufruf immer aus SQL*Plus-Ebene
      - – inclusive Ausführung SQL und PL/SQL
- von SQL*Plus – Ebene in alle Ebenen
  - durch Verwendung von define Variablen
- von OS – Ebene in alle Ebenen
  - keine weitere Möglichkeit außer Dateien
  - ➔ nicht möglich, wenn keine Schreibrechte

Informationstransfer zwischen Arbeitsebenen

- von SQL – Ebene in SQL*Plus – Ebene
  - durch spezielle column Definition direct in define Variable
    - column "My Col" noprint new_value my_var
      - noprint nur für Ausgabe-Unterdrückung
- von PL / SQL – Ebene in SQL*Plus-Ebene
  - außer spool-Datei keine directe Möglichkeit
  - indirect via Bind-Variable und SQL-Ebene in define Variable
- von PL / SQL – Ebene in SQL – Ebene
  - über Zuweisung von PL/SQL-Variable an Bind-Variable
  - Bind-Variable muss in SQL*Plus-Ebene definiert worden sein
- SQL – Ebene in PL / SQL – Ebene
  - über Zuweisung Bind-Variable an PL/SQL-Variable

Voraussetzungen

- Verbindung zur Datenbank
  - wg. Einsatz von SELECT-Statements
- SQLPLUS_PRODUCT_PROFILE
  - keine Einschränkungen für aufrufenden User
- mindestens Oracle Version 10
  - spool filename.ext append – vorher nicht verfügbar
    - ➢ kann unter Unix als Script mit Shell-Mitteln nachgebildet werden
      - – spool schreibt in Named Pipe
      - – Shell-Prozeß (Hintergrund) kopiert in Zieldatei (Append)
- Schreibrechte im aktuellen Arbeitsverzeichnis
  - ggf. für spezielle Log- und SQL-Script-Generierungs-Verzeichnisse

Utilities: `import-OS-info.sql`

- Bestimmen des lokalen Betriebssystems
  - OS-spezifisch (unterstützt UNIX und WINDOWS Platformen)
  - Betriebssystem-Typ wird detectiert (UNIX oder WIN)
    - durch Bestimmung des actuellen Arbeitsverzeichnisses (cwd)
      - via host Kommando
      - erzeugen SQL-Script mit define Variablen Zuweisung
      - ausführen mit SQL*Plus
  - Auswertung mit SQL (decode()) nach folgender Regel:
      - IF cwd LIKE '/%', THEN 'UNIX',
      - ELSEIF cwd LIKE '_:%', THEN 'WIN',
      - ELSE 'UNKOWN',
  - Actuelles Arbeitsverzeichnis als Mehrwert
  - Rückgabe in define Variablen
    - LOCAL_OS_TYPE und CURRENT_WORKING_DIR
    - optional andere Variablen-Namen über Parameter angebbar

Utilities: `import-NLS_LANG-setting.sql`

- Import von Environment Variablen
  - OS-spezifisch (unterstützt UNIX und WINDOWS Platformen)
  - holt Information aus NLS_LANG ins SQL*Plus
    - via host Kommando
    - erzeugen SQL-Script mit define Variablen Zuweisung
    - ausführen mit SQL*Plus
  - Auswertung mit SQL
    - separate Variable für Characterset-Anteil
  - Rückgabe in define Variablen
    - NLS_LANG NLS_CHARACTER_SET
    - optional andere Variablen-Namen über Parameter angebbar

Utilities: `dir-write-check.sql`

- Directory-Schreibrechte prüfen
  - Script muss ohne Schreibrechte lauffähig sein
    - ➔ keine eigenen Dateien
    - ➔ keine Verwendung des host-Kommandos
  - 2 feste Parameter: Verzeichnisname und Rückgabe-Variable
  - Erzeugen commited Eintrag in PLAN_TABLE
    - GLOBAL TEMPORARY , PUBLIC write
    - dann uncommited UPDATE darauf
  - whenever oserror continue rollback
  - Spool-Versuch im angegebenen Verzeichnis
    - mit eindeutigem Dateinamen
      - zur Vermeidung Prüfung Dateischreibrechte
    - rollback bei Fehler
  - Auswertung Eintrag in PLAN_TABLE
  - Rückgabe in define Variable: schreibbar : 1, sonst: 0

Utilities: `delete-file.sql`

- Löschen von Dateien
  - nicht mit SQL*Plus-Mitteln möglich
  - OS-spezifisch (unterstützt UNIX und WINDOWS Platformen)
  - via host Kommando
    - zuerst Windows Syntax mit Error auf nul-Device
    - danach Unix Syntax mit Error auf null-Device
      - inclusive Löschen nul-Datei aus Windos Syntax
      - bei Ausführung unter Unix
  - Ermöglicht Standard-Aufruf auf allen unterstützten Platformen

Utilities: `lspool.sql` und `lspool-off.sql`

- Automatisches Schachteln beim Spooling
  - Name spool-Datei in spezielle define-Variable für jeden Level
    - muss nicht in jedem Installations-Unterscript bekannt sein
      - durch Parameterübergabe
    - reservierte define-Variablen:
      - ___LSPOOL_LEVEL
        - enthält aktuellen Level
        - wird mit 0 erzeugt, wenn nicht vorhanden
      - ___LSPOOL_FILE_LEVEL_#
        - # steht für Level-Wert (Integer)
        - enthalt Name der spool-Datei für Level #
  - keine explicite Limitierung der Schachtelungstiefe
    - nur durch Anzahl define-Variablen

Utilities: `lspool.sql` und `lspool-off.sql`

- Automatisches Schachteln beim Spooling (Fortsetzung)
  - lspool.sql - Aufruf wie spool Kommando
    - Dateiname als Parameter
      - Verwendung von „off“ vermeiden, stoppt spooling
        - aber kein Zurückswitchen auf vorherige spool-Datei
    - optionaler 2. Parameter als Mode: wird durchgereicht an spool
      - CREATE, REPLACE oder APPEND
      - default-Wert ist REPLACE
  - Ablauf
    - spool off
    - Inkrementierung ___LSPOOL_LEVEL
    - define ___LSPOOL_FILE_LEVEL_# mit spool-Datei als Inhalt
    - define ___LSPOOL_FILE_LEVEL_&___LSPOOL_LEVEL = spool-Datei
    - spool <spool-Datei> <Mode>

Utilities: `lspool.sql` und `lspool-off.sql`

- Automatisches Schachteln beim Spooling (Fortsetzung II)
  - lspool-off.sql
    - keine Parameter
    - entspricht dem Kommando „spool off“
  - Ablauf
    - spool off
    - undefine ___LSPOOL_FILE_LEVEL_&___LSPOOL_LEVEL
      - mit gerade geschlossener spool-Datei
    - Dekrementierung ___LSPOOL_LEVEL
    - wenn neuer ___LSPOOL_LEVEL > 0
      - spool-Datei ist:
        - &(___LSPOOL_FILE_LEVEL_&___LSPOOL_LEVEL)
        - diese Syntax ist nur über Zwischen-spool möglich
      - spool <spool-Datei> APPEND
        - nun wird in vorheriger spool-Datei weiter-geschrieben

Utilities: `lspool.sql` und `lspool-off.sql`

- Automatisches Schachteln beim Spooling (Fortsetzung III)
  - lspool-flush.sql
    - als spin off
    - wie lspool-off.sql
    - aber ohne die Steps
      - undefine ___LSPOOL_FILE_LEVEL_&___LSPOOL_LEVEL
      - Dekrementierung ___LSPOOL_LEVEL
    - damit wird gleiche spool-Datei erneut geöffnet
      - ohne dass der Name angegeben werden muss
    - schließen der Datei bewirkt flushing
    - danach erneut im APPEND-Mode öffnen
    - wirkt genauso wie ein flush bei geöffneter Datei

Utilities: `ifelsestart.sql` und `ifstart.sql`

- Bedingte Script-Aufrufe
  - einfacher directer Aufruf
    - ohne explicites SELECT und spool
  - ifelsestart.sql
    - 1. Parameter ist Bedingung
      - alles, was als SELECT-WHERE-Condition möglich ist
      - mit Quotes wenn Leerzeichen enthalten sind
    - 2. Parameter ist Script für TRUE-Fall
    - 3. Parameter ist Script für FALSE-Fall
    - weitere 8 optionale Parameter werden durchgereicht
      - an das auszuführende Script
      - TRUE-Script und FALSE-Script erhalten gleiche Parameter
    - Aufruf über define-Variable
      - @&ifelsestart_command_line
      - mit Befüllung über SELECT und column
        - ohne spool

Utilities: `ifelsestart.sql` und `ifstart.sql`

- Bedingte Script-Aufrufe (Fortsetzung)
  - ifstart.sql
    - vereinfachte Version von ifelsestart.sql
      - ohne spezielles Script für den FALSE-Fall
      - der Parameter dafür entfällt
    - festes Script null.sql für den FALSE-Fall
      - wird im gleichen Verzeichnis erwartet
      - ist ein leeres Script (bis auf Kommentare)
      - sorgt für korrekte Syntax
    - null.sql kann auch als Parameter für ifelsestart.sql verwendet werden
      - oder für beliebige andere Einsatzmöglichkeiten

Utilities: `ifstart-fatal-fin.sql` und `fatal-fin.sql`

- Abbrechen von SQL*Plus bei Fehlern
  - Scripte müssen ohne Schreibrechte lauffähig sein
    - für Abbruch bei fehlenden Schreibrechten
    - ➔ keine eigenen Dateien
  - ifstart-fatal-fin.sql für bedingten Aufruf von fatal-fin.sql
    - abgeleitet von ifstart.sql
    - ohne eigene Dateien
      - kein Parameter-Default-Handling
      - kein Sichern und Wiederherstellen von Einstellungen
        - fast keine Einstellungen ändern
    - Verwendung SQL-Buffer für Zusammenbau SELECT
      - zur Befüllung Aufruf-Kommandozeile mit fatal-fin.sql
    - nach Bedingung nur 2 feste Parameter
      - Exit Value und Fehlermeldung, Durchreichung an fatal-fin.sql
    - Aufruf von fatal-fin.sql, wenn Bedingung TRUE ist
      - sonst Aufruf von null.sql

Utilities: `ifstart-fatal-fin.sql` und `fatal-fin.sql`

- Abbrechen von SQL*Plus bei Fehlern (Fortsetzung)
  - fatal-fin.sql
    - beendet SQL*Plus
      - mit übergebenem Exit Value
      - nach Ausgabe der übergebenen Fehlermeldung
        - und Meldung Session-Ende und ggf. SQL*Plus Exit
    - Debugging Mode für Entwickler
      - über spezielle Einstellung des _EDITOR Settings
        - existiert immer ➔ kein Defaulthandling notwendig
        - wird sonst in Scripten nicht benötigt
      - nur die Session wird beendet, aber nicht SQL*Plus
      - zum Sichten der Variablen
      - Scripte können nicht mehr auf Datenbank zugreifen
    - dynamische whenever Klausel mit define-Variable
      - exit im Normalfall, continue im Debugging-Mode
    - danach RAISE Exception ORA-00028 (kill session)
    - ggf. spezielle Meldung, wenn SQL*Plus nicht verlassen wurde

Utilities: `DB-install.sql` `(eval-verbose_output_level.sql)`

- Aufrufe von Objectscripten mit Ausgabe-Steuerung
  - DB-install.sql
    - ruft übergebenes Object-Installationsscript auf
    - verwendet die Variablen verbose_#_output, mit # in (1,2,3)
      - mit den möglichen Werten ON un OFF
      - zur Steuerung des Outputs
    - Code:

```
set termout &verbose_1_output echo off
prompt installing &&1
set termout &verbose_2_output echo &verbose_3_output
@&1
set termout off echo off
```

    - ermöglicht zusätzlichen Output
      - durch Setzen der nächsthöheren Variable auf ON

Utilities: `DB-install.sql` `(eval-verbose_output_level.sql)`

- Aufrufe von Objectscripten mit Ausgabe-Steuerung
  - eval-verbose_output_level.sql
    - Hilfsscript zum Setzen der Steuer-Variablen verbose_#_output
    - Input: Variable verbose_output__level
      - Integer-Wert
      - Steuer-Variable wird ON, wenn # <= verbose_output__level
      - Steuer-Variable wird OFF, wenn # > verbose_output__level
    - optionaler Parameter definiert Defaultwert
      - von verbose_output__level, wenn nicht definiert

Zusammenfassung

- Erhebliche Codeanteile für:
  - Informationstransfer zwischen den Arbeitsebenen
  - Sichern, Ändern, Wiederherstellen von SQL*Plus-Einstellungen
  - Beseitigung von define Variablen und column Definitonen
    - die im Script erzeugt wurden
- Dies sind besonders fehlerträchtige Codeanteile
  - Fehler fallen meist erst in nachfolgenden Scripten auf
    - ➔ Debugging schwieriger und aufwändiger
- ➔ Einsatz solcher Scripte um so wichtiger
  - wegen verbesserter Qualität und Wartbarkeit
    - durch gekapselte und getestete Zusatz-Funktionalität
  - Entwickler können sich aufs wesentliche konzentrieren

Fragen & Antworten

Vielen Dank für Ihre Aufmerksamkeit

Kontakt: **Dr. Kurt Franke**
Cellent Finance Solutions AG, Calwer Str. 33, D-70173 Stuttgart
Email: Kurt.Franke@cellent-fs.de , Kurt-Franke@web.de
Phone: +49-(0)711-222992676 , Mobil: +49-(0)171-7963089