

**Oracle Database 12c:
Develop PL/SQL Program Units**

Schulungsunterlagen • Band II

D80170DE10

Production 1.0

Oktober 2013

Bestellnummer: D83982

ORACLE®

Autor

Supriya Ananth

Technischer Inhalt und Überarbeitung

Wayne Abbott

Anjulapponni Azhagulekshmi

Christopher Burandt

Yanti Chang

Diganta Choudhury

Salome Clement

Laszlo Czinkoczki

Steve Friedberg

Nancy Greenberg

KimSeong Loh

Miyuki Osato

Manish Pawar

Brian Pottle

Swarnapriya Shridhar

Grafische Gestaltung

Seema Bopaiah

Redaktion

Raj Kumar

Richard Wallis

Herausgeber

Glenn Austin

Sumesh Koshy

Copyright © 2013, Oracle und/oder verbundene Unternehmen. All rights reserved. Alle Rechte vorbehalten.

Diese Software und zugehörige Dokumentation werden im Rahmen eines Lizenzvertrages zur Verfügung gestellt, der Einschränkungen hinsichtlich Nutzung und Offenlegung enthält und durch Gesetze zum Schutz geistigen Eigentums geschützt ist. Sofern nicht ausdrücklich in Ihrem Lizenzvertrag vereinbart oder gesetzlich geregelt, darf diese Software weder ganz noch teilweise in irgendeiner Form oder durch irgendein Mittel zu irgendeinem Zweck kopiert, reproduziert, übersetzt, gesendet, verändert, lizenziert, übertragen, verteilt, ausgestellt, ausgeführt, veröffentlicht oder angezeigt werden. Reverse Engineering, Disassemblierung oder Dekompliierung der Software ist verboten, es sei denn, dies ist erforderlich, um die gesetzlich vorgesehene Interoperabilität mit anderer Software zu ermöglichen.

Die hier angegebenen Informationen können jederzeit und ohne vorherige Ankündigung geändert werden. Wir übernehmen keine Gewähr für deren Richtigkeit. Sollten Sie Fehler oder Unstimmigkeiten finden, bitten wir Sie, uns diese schriftlich mitzuteilen.

Wird diese Software oder zugehörige Dokumentation an die Regierung der Vereinigten Staaten von Amerika bzw. einen Lizenznehmer im Auftrag der Regierung der Vereinigten Staaten von Amerika geliefert, gilt Folgendes:

U.S. GOVERNMENT END USERS:

Oracle programs, including any operating system, integrated software, any programs installed on the hardware, and/or documentation, delivered to U.S. Government end users are "commercial computer software" pursuant to the applicable Federal Acquisition Regulation and agency-specific supplemental regulations. As such, use, duplication, disclosure, modification, and adaptation of the programs, including any operating system, integrated software, any programs installed on the hardware, and/or documentation, shall be subject to license terms and license restrictions applicable to the programs. No other rights are granted to the U.S. Government.

Oracle und Java sind eingetragene Marken der Oracle Corporation und/oder ihrer verbundenen Unternehmen. Andere Namen und Bezeichnungen können Marken ihrer jeweiligen Inhaber sein.

Inhaltsverzeichnis

1 Einführung

- Ziele 1-2
- Lektionsagenda 1-3
- Lernziele 1-4
- Empfohlene Kursagenda 1-5
- Lektionsagenda 1-7
 - Das in diesem Kurs verwendete Schema "Human Resources (HR)" 1-8
 - Informationen zu den Kursaccounts 1-9
 - In diesem Kurs verwendete Anhänge 1-10
 - PL/SQL-Entwicklungsumgebungen 1-11
 - Was ist Oracle SQL Developer? 1-12
 - PL/SQL in SQL*Plus codieren 1-13
 - Ausgaben von PL/SQL-Blöcken ermöglichen 1-14
 - Lektionsagenda 1-15
 - Oracle Cloud 1-16
 - Oracle Cloud-Services 1-17
 - Cloudbereitstellungsmodelle 1-18
 - Lektionsagenda 1-19
 - SQL- und PL/SQL-Dokumentation von Oracle 1-20
 - Zusätzliche Ressourcen 1-21
 - Zusammenfassung 1-22
 - Übungen zu Lektion 1 – Überblick: Erste Schritte 1-23

2 Prozeduren erstellen

- Ziele 2-2
- Lektionsagenda 2-3
- Modularisierte Unterprogramme erstellen 2-4
- Unterprogramme mit Schichten erstellen 2-5
- Entwicklung mit PL/SQL-Blöcken modularisieren 2-6
- Anonyme Blöcke – Überblick 2-7
- PL/SQL-Laufzeitarchitektur 2-8
- Was sind PL/SQL-Unterprogramme? 2-9
- PL/SQL-Unterprogramme – Vorteile 2-10
- Anonyme Blöcke und Unterprogramme – Unterschiede 2-11
- Lektionsagenda 2-12

Was sind Prozeduren? 2-13
Prozeduren erstellen – Überblick 2-14
Prozeduren mit der SQL-Anweisung CREATE OR REPLACE erstellen 2-15
Prozeduren mit SQL Developer erstellen 2-16
In SQL Developer Prozeduren kompilieren und Kompilierungsfehler anzeigen 2-17
Kompilierungsfehler in SQL Developer korrigieren 2-18
Benennungskonventionen der in diesem Kurs verwendeten PL/SQL-Strukturen 2-19
Was sind Parameter und Parametermodi? 2-20
Formale und tatsächliche Parameter – Vergleich 2-21
Prozedurale Parametermodi 2-22
Parametermodi – Vergleich 2-23
Parametermodus IN – Beispiel 2-24
Parametermodus OUT – Beispiel 2-25
Parametermodus IN OUT – Beispiel 2-26
OUT-Parameter mit der untergeordneten Routine DBMS_OUTPUT.PUT_LINE anzeigen 2-27
OUT-Parameter mit SQL*Plus-Hostvariablen anzeigen 2-28
Verfügbare Notationen für die Übergabe von tatsächlichen Parametern 2-29
Tatsächliche Parameter übergeben – Prozedur add_dept erstellen 2-30
Tatsächliche Parameter übergeben – Beispiele 2-31
Option DEFAULT für Parameter 2-32
Prozeduren aufrufen 2-34
Prozeduren mit SQL Developer aufrufen 2-35
Lektionsagenda 2-36
Behandelte Exceptions 2-37
Behandelte Exceptions – Beispiel 2-38
Nicht behandelte Exceptions 2-39
Nicht behandelte Exceptions – Beispiel 2-40
Prozeduren mithilfe der SQL-Anweisung DROP oder mit SQL Developer entfernen 2-41
Prozedurinformationen mithilfe von Data Dictionary Views anzeigen 2-42
Prozedurinformationen mit SQL Developer anzeigen 2-43
Lektionsagenda 2-44
PL/SQL-Bind-Typen 2-45
Unterprogramme mit BOOLEAN-Parametern 2-46
Quiz 2-47
Zusammenfassung 2-48
Übungen zu Lektion 2 – Überblick: Prozeduren erstellen, kompilieren und aufrufen 2-49

3 Funktionen erstellen und Unterprogramme debuggen

Ziele 3-2

Lektionsagenda 3-3

Stored Functions – Überblick 3-4

Funktionen erstellen 3-5

Prozeduren und Funktionen – Unterschiede 3-6

Funktionen erstellen und ausführen – Überblick 3-7

Stored Functions mit CREATE FUNCTION-Anweisungen erstellen und aufrufen –

Beispiel 3-8

Funktionen mithilfe verschiedener Methoden ausführen 3-9

Funktionen mit SQL Developer erstellen und kompilieren 3-11

Funktionen mit SQL Developer ausführen 3-12

Benutzerdefinierte Funktionen in SQL-Anweisungen – Vorteile 3-13

Funktionen in SQL-Ausdrücken – Beispiel 3-14

Benutzerdefinierte Funktionen in SQL-Anweisungen aufrufen 3-15

Funktionen aus SQL-Ausdrücken aufrufen – Einschränkungen 3-16

Funktionen aus SQL-Ausdrücken aufrufen – Seiteneffekte ausschalten 3-17

Funktionen aus SQL aufrufen – Einschränkungen: Beispiel 3-18

Benannte und gemischte Notation aus SQL 3-19

Benannte und gemischte Notation aus SQL – Beispiel 3-20

Funktionen mit Data Dictionary Views anzeigen 3-21

Funktionsinformationen mit SQL Developer anzeigen 3-22

Funktionen mithilfe der SQL-Anweisung `DROP` oder mit SQL Developer entfernen 3-23

Quiz 3-24

Übung 1 zu Lektion 3 – Überblick 3-25

Lektionsagenda 3-26

PL/SQL-Unterprogramme mit SQL Developer-Debugger debuggen 3-27

Unterprogramme debuggen – Überblick 3-28

Registerkarte zur Bearbeitung von Prozedur- oder Funktionscode 3-29

Registerkarte für Prozeduren oder Funktionen – Symbolleiste 3-30

Registerkarte **Debugging – Log** – Symbolleiste 3-31

Weitere Registerkarten 3-33

Prozeduren debuggen – Beispiel: Neue Prozedur `emp_list` erstellen 3-34

Prozeduren debuggen – Beispiel: Neue Funktion `get_location` erstellen 3-35

Breakpoints festlegen und `emp_list` für Debugmodus kompilieren 3-36

Funktion `get_location` für Debugmodus kompilieren 3-37

`emp_list` debuggen und Werte für Parameter `PMAXROWS` eingeben 3-38

`emp_list` debuggen – In Code gehen (F7) 3-39

Daten anzeigen	3-41
Variablen beim Debugging des Codes ändern	3-42
emp_list debuggen – Code übergehen	3-43
emp_list debuggen – Code verlassen (UMSCHALT+F7)	3-44
emp_list debuggen – Code bis Cursor ausführen (F4)	3-45
emp_list debuggen – Zum Methodenende gehen	3-46
Unterprogramme debuggen – Überblick	3-47
Übung 2 zu Lektion 3 – Überblick: SQL Developer-Debugger – Einführung	3-48
Zusammenfassung	3-49

4 Packages erstellen

Ziele	4-2
Lektionsagenda	4-3
Was sind PL/SQL-Packages?	4-4
Packages – Vorteile	4-5
PL/SQL-Packages – Komponenten	4-7
Packagekomponenten – interne und externe Sichtbarkeit	4-8
PL/SQL-Packages entwickeln – Überblick	4-9
Lektionsagenda	4-10
Packagespezifikationen mit CREATE PACKAGE-Anweisungen erstellen	4-11
Packagespezifikationen mithilfe von SQL Developer erstellen	4-12
Packagebodys mithilfe von SQL Developer erstellen	4-13
Packagespezifikationen – Beispiel: comm_pkg	4-14
Packagebody erstellen	4-15
Packagebodys – Beispiel: comm_pkg	4-16
Packageunterprogramme aufrufen – Beispiele	4-17
Packageunterprogramme aufrufen – mit SQL Developer	4-18
Packages ohne Body erstellen und verwenden	4-19
Packages im Data Dictionary anzeigen	4-20
Packages in SQL Developer anzeigen	4-21
Packages mit SQL Developer oder der SQL-Anweisung DROP entfernen	4-22
Packages erstellen – Richtlinien	4-23
Quiz	4-24
Zusammenfassung	4-25
Übungen zu Lektion 4 – Überblick: Packages erstellen und verwenden	4-26

5 Mit Packages arbeiten

Ziele	5-2
Lektionsagenda	5-3
Unterprogramme in PL/SQL überladen	5-4

Prozeduren überladen – Beispiel: Packagespezifikationen erstellen	5-6
Prozeduren überladen – Beispiel: Packagebody erstellen	5-7
Überladung und Package STANDARD	5-8
Ungültige Prozedurreferenzen	5-9
Ungültige Prozedurreferenzen mithilfe von Vorwärtsdeklarationen auflösen	5-10
Packages initialisieren	5-11
Packagefunktionen in SQL	5-12
Seiteneffekte von PL/SQL-Unterprogrammen ausschalten	5-13
Packagefunktionen in SQL – Beispiel	5-14
Lektionsagenda	5-15
Persistente Packagezustände	5-16
Persistente Zustände von Packagevariablen – Beispiel	5-18
Persistente Zustände von Packagecursors – Beispiel	5-19
Package CURS_PKG ausführen	5-21
Assoziative Arrays in Packages	5-22
Quiz	5-23
Zusammenfassung	5-24
Übungen zu Lektion 5 – Überblick: Mit Packages arbeiten	5-25

6 Von Oracle bereitgestellte Packages zur Anwendungsentwicklung

Ziele	6-2
Lektionsagenda	6-3
Von Oracle bereitgestellte Packages	6-4
Von Oracle bereitgestellte Packages – Beispiele	6-5
Lektionsagenda	6-7
Funktionsweise des Packages DBMS_OUTPUT	6-8
Mit dem Package UTL_FILE mit Betriebssystemdateien interagieren	6-9
Einige Prozeduren und Funktionen von UTL_FILE	6-10
Dateibearbeitung mit dem Package UTL_FILE – Überblick	6-11
Verfügbare deklarierte Exceptions im Package UTL_FILE	6-12
Funktionen FOPEN und IS_OPEN – Beispiel	6-13
UTL_FILE – Beispiel	6-16
Was ist das Package UTL_MAIL?	6-18
UTL_MAIL einrichten und verwenden – Überblick	6-20
UTL_MAIL-Unterprogramme – Zusammenfassung	6-21
UTL_MAIL installieren und verwenden	6-22
Syntax der Prozedur SEND	6-23
Prozedur SEND_ATTACH_RA	6-24
E-Mails mit einem binären Anhang senden – Beispiel	6-25
Prozedur SEND_ATTACH_VARCHAR2	6-27

E-Mails mit einem Textanhang senden – Beispiel 6-28
Quiz 6-30
Zusammenfassung 6-31
Übungen zu Lektion 6 – Überblick: Von Oracle bereitgestellte Packages zur Anwendungsentwicklung 6-32

7 Dynamisches SQL

Ziele 7-2
Lektionsagenda 7-3
Ausführungsablauf von SQL-Anweisungen 7-4
Mit dynamischem SQL arbeiten 7-5
Dynamisches SQL 7-6
Natives dynamisches SQL (NDS) 7-7
Anweisung EXECUTE IMMEDIATE 7-8
Verfügbare Methoden zur NDS-Verwendung 7-9
Dynamisches SQL mit DDL-Anweisungen – Beispiele 7-11
Dynamisches SQL mit DML-Anweisungen 7-12
Dynamisches SQL mit Single Row-Abfragen – Beispiel 7-13
Anonyme PL/SQL-Blöcke dynamisch ausführen 7-14
PL/SQL-Code mit nativem dynamischem SQL kompilieren 7-15
Lektionsagenda 7-16
Package DBMS_SQL 7-17
Unterprogramme des Packages DBMS_SQL 7-18
DBMS_SQL mit DML-Anweisungen – Zeilen löschen 7-20
DBMS_SQL mit parametrisierten DML-Anweisungen 7-22
Quiz 7-23
Zusammenfassung 7-24
Übungen zu Lektion 7 – Überblick: Natives dynamisches SQL 7-25

8 Überlegungen zum Design von PL/SQL-Code

Ziele 8-2
Lektionsagenda 8-3
Konstanten und Exceptions standardisieren 8-4
Exceptions standardisieren 8-5
Exception-Behandlung standardisieren 8-6
Konstanten standardisieren 8-7
Lokale Unterprogramme 8-8
Rechte des Eigentümers und Rechte des ausführenden Benutzers – Vergleich 8-9
Rechte des ausführenden Benutzers angeben – AUTHID auf CURRENT_USER einstellen 8-10

Autonome Transaktionen 8-11
Autonome Transaktionen – Features 8-12
Autonome Transaktionen – Beispiel 8-13
Lektionsagenda 8-15
PL/SQL-Packages und Standalone Stored Subprograms Rollen erteilen 8-16
Lektionsagenda 8-17
Hint NOCOPY 8-18
Auswirkungen des Hints NOCOPY 8-19
Wann ignoriert der PL/SQL-Compiler den Hint NOCOPY? 8-20
Hint PARALLEL_ENABLE 8-21
Sessionübergreifender Ergebniscache für PL/SQL-Funktionen 8-22
Ergebniscache für Funktionen aktivieren 8-23
Zwischengespeicherte Funktionen deklarieren und definieren – Beispiel 8-24
Klausel DETERMINISTIC mit Funktionen 8-26
Lektionsagenda 8-27
Klausel RETURNING 8-28
Bulk Binding 8-29
Bulk Binding – Syntax und Schlüsselwörter 8-30
Bulk Binding mit FORALL – Beispiel 8-32
BULK COLLECT INTO mit Abfragen 8-34
BULK COLLECT INTO mit Cursors 8-35
BULK COLLECT INTO mit einer RETURNING-Klausel 8-36
Bulk Binding in wenig gefüllten Collections 8-37
Bulk Binding mit Indexarray 8-40
Quiz 8-41
Zusammenfassung 8-42
Übungen zu Lektion 8 – Überblick: Überlegungen zum Design von
PL/SQL-Code 8-43

9 Trigger erstellen

Ziele 9-2
Was sind Trigger? 9-3
Trigger definieren 9-4
Triggerereignistypen 9-5
Anwendungs- und Datenbanktrigger 9-6
Trigger implementieren – Szenarios für Geschäftsanwendungen 9-7
Verfügbare Triggertypen 9-8
Triggerereignistypen und Triggerbody 9-9
DML-Trigger mit der Anweisung CREATE TRIGGER erstellen 9-10
Triggerauslösung angeben (Timing) 9-12

Trigger auf Anweisungs- bzw. Zeilenebene – Vergleich 9-13
DML-Trigger mit SQL Developer erstellen 9-14
Auslösereihenfolge der Trigger – Single Row-Bearbeitung 9-15
Auslösereihenfolge der Trigger – Multiple Row-Bearbeitung 9-16
DML-Statement Trigger erstellen – Beispiel: SECURE_EMP 9-17
SECURE_EMP-Trigger testen 9-18
Bedingungsprädikate 9-19
DML-Row Trigger erstellen 9-20
Qualifizierer OLD und NEW 9-21
Qualifizierer OLD und NEW – Beispiel 9-22
Row Trigger mit der Klausel WHEN bedingt auslösen 9-24
Triggerausführungsmodell – Zusammenfassung 9-25
Integritäts-Constraints mit After Triggern implementieren 9-26
INSTEAD OF-Trigger 9-27
Trigger INSTEAD OF erstellen – Beispiel 9-28
Trigger INSTEAD OF zum Ausführen von DML für komplexe Views erstellen 9-29
Triggerstatus 9-31
Deaktivierte Trigger erstellen 9-32
Trigger mit SQL-Anweisungen ALTER und DROP verwalten 9-33
Trigger mit SQL Developer verwalten 9-34
Trigger testen 9-35
Triggerinformationen anzeigen 9-36
USER_TRIGGERS 9-37
Quiz 9-38
Zusammenfassung 9-39
Übungen zu Lektion 9 – Überblick: Statement Trigger und Row Trigger erstellen 9-40

10 Komplexe, DDL- und Datenbankereignistrigger erstellen

Ziele 10-2
Was ist ein komplexer Trigger? 10-3
Mit komplexen Triggern arbeiten 10-4
Komplexe Trigger – Vorteile 10-5
Zeitpunktbereiche von komplexen Triggern für Tabellen 10-6
Komplexe Trigger für Tabellen – Struktur 10-7
Komplexe Trigger für Tabellen – Struktur 10-8
Komplexe Trigger – Einschränkungen 10-9
Triggereinschränkungen für sich verändernde Tabellen 10-10
Sich verändernde Tabellen – Beispiel 10-11
Fehler in sich verändernden Tabellen mit komplexen Triggern beheben 10-13
Trigger für DDL-Anweisungen erstellen 10-15

Datenbankereignistrigger erstellen 10-16
Trigger für Systemereignisse erstellen 10-17
Trigger LOGON und LOGOFF – Beispiel 10-18
CALL-Anweisungen in Triggern 10-19
Datenbankereignistrigger – Vorteile 10-20
Erforderliche Systemberechtigungen zur Triggerverwaltung 10-21
Richtlinien für das Entwerfen von Triggern 10-22
Quiz 10-23
Zusammenfassung 10-24
Übungen zu Lektion 10 – Überblick: Komplexe, DDL- und Datenbankereignistrigger erstellen 10-25

11 PL/SQL-Compiler

Ziele 11-2
Lektionsagenda 11-3
Initialisierungsparameter für PL/SQL-Kompilierung 11-4
Initialisierungsparameter für PL/SQL-Kompilierung – Verwendung 11-5
Compiler-Einstellungen 11-7
PL/SQL-Initialisierungsparameter anzeigen 11-8
PL/SQL-Initialisierungsparameter anzeigen 11-9
PL/SQL-Initialisierungsparameter ändern – Beispiel 11-10
Lektionsagenda 11-11
PL/SQL-Kompilierungszeitwarnungen für Unterprogramme – Überblick 11-12
Vorteile von Compiler-Warnungen 11-14
PL/SQL-Kompilierungswarnungen – Kategorien 11-15
Warnstufen einstellen 11-16
Compilerwarnstufen einstellen – mit `PLSQL_WARNINGS` 11-17
Compilerwarnstufen einstellen – mit `PLSQL_WARNINGS`: Beispiel 11-18
Compilerwarnstufen einstellen – mit `PLSQL_WARNINGS` in SQL Developer 11-19
Aktuelle Einstellung von `PLSQL_WARNINGS` anzeigen 11-20
Compilerwarnungen anzeigen: Mit SQL Developer, SQL*Plus oder Data Dictionary Views 11-22
SQL*Plus-Warnmeldungen – Beispiel 11-23
Richtlinien für die Verwendung von `PLSQL_WARNINGS` 11-24
Lektionsagenda 11-25
Compilerwarnstufen einstellen – mit dem Package `DBMS_WARNING` 11-26
Unterprogramme des Packages `DBMS_WARNING` 11-28
`DBMS_WARNING`-Prozeduren – Syntax, Parameter und zulässige Werte 11-29
`DBMS_WARNING`-Prozeduren – Beispiel 11-30

DBMS_WARNING-Funktionen – Syntax, Parameter und zulässige Werte	11-31
DBMS_WARNING-Funktionen – Beispiel	11-32
DBMS_WARNING – Beispiel	11-33
Warnmeldung PLW 06009	11-35
Warnmeldung PLW 06009 – Beispiel	11-36
Quiz	11-37
Zusammenfassung	11-38
Übungen zu Lektion 11 – Überblick: PL/SQL-Compiler	11-39

12 Abhängigkeiten verwalten

Ziele	12-2
Abhängigkeiten von Schemaobjekten – Überblick	12-3
Abhängigkeiten	12-4
Direkte lokale Abhängigkeiten	12-5
Direkte Objektabhängigkeiten abfragen – View USER_DEPENDENCIES	12-6
Objektstatus abfragen	12-7
Invalidierung abhängiger Objekte	12-8
Schemaobjektänderungen, durch die einige Abhängigkeiten ungültig werden –	
Beispiel	12-9
Direkte und indirekte Abhängigkeiten anzeigen	12-11
Abhängigkeiten anzeigen – View DEPTREE	12-12
Präzisere Abhängigkeitsmetadaten mit Oracle Database 11g	12-13
Fein granulierte Abhängigkeitsverwaltung	12-14
Fein granulierte Abhängigkeitsverwaltung – Beispiel 1	12-15
Fein granulierte Abhängigkeitsverwaltung – Beispiel 2	12-17
Änderungen bei Synonymabhängigkeiten	12-18
Gültige PL/SQL-Programmeinheiten und -Views beibehalten	12-19
Lokale Abhängigkeiten – Weiteres Szenario	12-20
Richtlinien zur Reduzierung für ungültig erklärter Objekte	12-21
Objekte neu validieren	12-22
Remote-Abhängigkeiten	12-23
Remote-Abhängigkeiten – Konzepte	12-24
Parameter REMOTE_DEPENDENCIES_MODE einstellen	12-25
Remote-Prozedur B wird um 8 Uhr kompiliert	12-26
Lokale Prozedur A wird um 9 Uhr kompiliert	12-27
Prozedur A ausführen	12-28
Remote-Prozedur B wird um 11 Uhr rekompiliert	12-29
Prozedur A ausführen	12-30
Signaturmodus	12-31
PL/SQL-Programmeinheiten rekompilieren	12-32

Nicht erfolgreiche Rekomplilierung 12-33
Erfolgreiche Rekomplilierung 12-34
Prozeduren rekompilieren 12-35
Packages und Abhangigkeiten – Unterprogramm referenziert das Package 12-36
Packages und Abhangigkeiten – Packageunterprogramm referenziert
 Prozedur 12-37
Quiz 12-38
Zusammenfassung 12-39
Übungen zu Lektion 12 – berblick: Abhangigkeiten im Schema verwalten 12-4

A Tabellenbeschreibungen

B SQL Developer

Ziele B-2
Was ist Oracle SQL Developer? B-3
SQL Developer – Spezifikationen B-4
SQL Developer 3.2 – Benutzeroberflache B-5
Datenbankverbindungen erstellen B-7
Datenbankobjekte durchsuchen B-10
Tabellenstrukturen anzeigen B-11
Dateien durchsuchen B-12
Schemaobjekte erstellen B-13
Neue Tabellen erstellen – Beispiel B-14
SQL Worksheet B-15
SQL-Anweisungen ausfuhren B-19
SQL-Skripte speichern B-20
Gespeicherte Skriptdateien ausfuhren – Methode 1 B-21
Gespeicherte Skriptdateien ausfuhren – Methode 2 B-22
SQL-Code formatieren B-23
Snippets B-24
Snippets – Beispiel B-25
Papierkorb B-26
Prozeduren und Funktionen debuggen B-27
Datenbankberichte B-28
Benutzerdefinierte Berichte erstellen B-29
Suchmaschinen und externe Tools B-30
Voreinstellungen festlegen B-31
SQL Developer-Layout zurucksetzen B-33
Data Modeler in SQL Developer B-34
Zusammenfassung B-35

C SQL*Plus

- Ziele C-2
- SQL und SQL*Plus – Interaktion C-3
- SQL-Anweisungen und SQL*Plus-Befehle – Vergleich C-4
- SQL*Plus – Überblick C-5
- Bei SQL*Plus anmelden C-6
- Tabellenstrukturen anzeigen C-7
- SQL*Plus – Bearbeitungsbefehle C-9
- LIST, n und APPEND C-11
- Befehl CHANGE C-12
- SQL*Plus – Dateibefehle C-13
- Befehle SAVE, START C-14
- Befehl SERVEROUTPUT C-15
- SQL*Plus-Befehl SPOOL C-16
- Befehl AUTOTRACE C-17
- Zusammenfassung C-18

D REF-Cursor

- Cursorvariablen D-2
- Cursorvariablen – Verwendung D-3
- REF CURSOR-Typen definieren D-4
- OPEN-FOR-, FETCH- und CLOSE-Anweisungen D-7
- Fetch-Vorgänge – Beispiel D-10

E Häufig verwendete SQL-Befehle

- Ziele E-2
- Einfache SELECT-Anweisungen E-3
- SELECT-Anweisungen E-4
- WHERE-Klauseln E-5
- ORDER BY-Klauseln E-6
- GROUP BY-Klauseln E-7
- Data Definition Language E-8
- CREATE TABLE-Anweisungen E-9
- ALTER TABLE-Anweisungen E-10
- DROP TABLE-Anweisungen E-11
- GRANT-Anweisungen E-12
- Typen von Berechtigungen E-13
- REVOKE-Anweisungen E-14
- TRUNCATE TABLE-Anweisungen E-15

Data Manipulation Language	E-16
INSERT-Anweisungen	E-17
UPDATE-Anweisungen – Syntax	E-18
SELECT-Anweisungen	E-19
Anweisungen zur Transaktionskontrolle	E-20
COMMIT-Anweisungen	E-21
ROLLBACK-Anweisungen	E-22
SAVEPOINT-Anweisungen	E-23
Joins	E-24
Typen von Joins	E-25
Mehrdeutige Spaltennamen eindeutig kennzeichnen	E-26
Natural Joins	E-27
Equi Join	E-28
Datensätze mithilfe von Equi Joins abrufen	E-29
Suchbedingungen mit den Operatoren AND und WHERE hinzufügen	E-30
Datensätze mithilfe von Non-Equi Joins abrufen	E-31
Datensätze mithilfe von USING-Klauseln abrufen	E-32
Datensätze mithilfe von ON-Klauseln abrufen	E-33
Left Outer Join	E-34
Right Outer Join	E-35
Full Outer Join	E-36
Self-Joins – Beispiel	E-37
Cross Join	E-38
Zusammenfassung	E-39

F PL/SQL-Code verwalten

Ziele	F-2
Agenda	F-3
Conditional Compilation	F-4
Conditional Compilation – Funktionsweise	F-5
Auswahlanweisungen	F-6
Vordefinierte und benutzerdefinierte Abfrageanweisungen	F-7
Parameter PLSQL_CCFLAGS und Abfrageanweisungen	F-8
Einstellung für den Initialisierungsparameter PLSQL_CCFLAGS anzeigen	F-9
Parameter PLSQL_CCFLAGS und Abfrageanweisungen – Beispiel	F-10
Benutzerdefinierte Fehler mithilfe von Conditional Compilation-Fehleranweisungen auslösen	F-11
Statische Ausdrücke und Conditional Compilation	F-12
Package DBMS_DB_VERSION – boolesche Konstanten	F-13
Package DBMS_DB_VERSION – Konstanten	F-14

Conditional Compilation und Datenbankversionen – Beispiel	F-15
Quelltext mit DBMS_PREPROCESSOR-Prozeduren ausgeben oder abrufen	F-17
Agenda	F-18
Obfuscation	F-19
Obfuscation – Vorteile	F-20
Dynamische Obfuscation – Neuerungen gegenüber Oracle 10g	F-21
Nicht verschlüsselter PL/SQL-Code – Beispiel	F-22
Verschlüsselter PL/SQL-Code – Beispiel	F-23
Dynamische Obfuscation – Beispiel	F-24
PL/SQL Wrapper	F-25
PL/SQL Wrapper ausführen	F-26
Wrapping-Ergebnisse	F-27
Wrapping-Richtlinien	F-28
Package DBMS_DDL und Utility wrap – Vergleich	F-29
Zusammenfassung	F-30

G PL/SQL – Wiederholung

Ziele	G-2
Blockstruktur für anonyme PL/SQL-Blöcke	G-3
PL/SQL-Variablen deklarieren	G-4
Variablen mit dem Attribut %TYPE deklarieren – Beispiele	G-5
PL/SQL-Records erstellen	G-6
Attribut %ROWTYPE – Beispiele	G-7
PL/SQL-Tabellen erstellen	G-8
SELECT-Anweisungen in PL/SQL – Beispiel	G-9
Daten einfügen – Beispiel	G-10
Daten aktualisieren – Beispiel	G-11
Daten löschen – Beispiel	G-12
Transaktionen mit den Anweisungen COMMIT und ROLLBACK steuern	G-13
IF-, THEN- und ELSIF-Anweisungen – Beispiel	G-14
Basisschleifen – Beispiel	G-16
FOR-Schleifen – Beispiel	G-17
WHILE-Schleifen – Beispiel	G-18
Implizite SQL-Cursorattribute	G-19
Explizite Cursor steuern	G-20
Explizite Cursor steuern – Cursor deklarieren	G-21
Explizite Cursor steuern – Cursor öffnen	G-22
Explizite Cursor steuern – Daten aus dem Cursor abrufen	G-23
Explizite Cursor steuern – Cursor schließen	G-24
Attribute von expliziten Cursors	G-25
Cursor FOR-Schleifen – Beispiel	G-26

Klausel FOR UPDATE – Beispiel G-27
 Klausel WHERE CURRENT OF – Beispiel G-28
 Vordefinierte Oracle-Serverfehler abfangen G-29
 Vordefinierte Oracle-Serverfehler abfangen – Beispiel G-30
 Nicht vordefinierte Fehler G-31
 Benutzerdefinierte Exceptions – Beispiel G-32
 RAISE_APPLICATION_ERROR-Prozeduren G-33
 Zusammenfassung G-35

H Trigger implementieren – Untersuchung

Ziele H-2
 Sicherheit im Server steuern H-3
 Sicherheit mit Datenbanktriggern steuern H-4
 Datenintegrität im Server durchsetzen H-5
 Datenintegrität mit Triggern schützen H-6
 Referenzielle Integrität im Server durchsetzen H-7
 Datenintegrität mit Triggern schützen H-8
 Tabellen im Server replizieren H-9
 Tabellen mit Triggern replizieren H-10
 Abgeleitete Daten im Server berechnen H-11
 Abgeleitete Werte mit Triggern berechnen H-12
 Ereignisse mit Triggern protokollieren H-13
 Zusammenfassung H-15

I DBMS_SCHEDULER- und HTP-Packages

Ziele I-2
 Webseiten mit dem HTP-Package generieren I-3
 HTP-Packageprozeduren I-4
 HTML-Dateien mit SQL*Plus erstellen I-5
 Package DBMS_SCHEDULER I-6
 Jobs erstellen I-8
 Jobs mit Inlineparametern erstellen I-9
 Jobs mit Programmen erstellen I-10
 Jobs für Programme mit Argumenten erstellen I-11
 Jobs mit Ausführungsplänen erstellen I-12
 Wiederholungsintervalle für Jobs einstellen I-13
 Jobs mit benannten Programmen und Ausführungsplänen erstellen I-14
 Jobs verwalten I-15
 Data Dictionary Views I-16
 Zusammenfassung I-17

11

PL/SQL-Compiler

ORACLE

Copyright © 2013, Oracle und/oder verbundene Unternehmen. All rights reserved. Alle Rechte vorbehalten.

Ziele

Nach Ablauf dieser Lektion haben Sie folgende Ziele erreicht:

- **Initialisierungsparameter des PL/SQL-Compilers verwenden**
- **PL/SQL-Kompilierungszeitwarnungen verwenden**

Der Inhalt dieses Kapitels bezieht sich ausschließlich auf 11g und spätere Releases. Unter Umständen funktionieren manche der aufgeführten Beispiele in einer 10g-Umgebung nicht.



Copyright © 2013, Oracle und/oder verbundene Unternehmen. All rights reserved. Alle Rechte vorbehalten.

Lektionsagenda

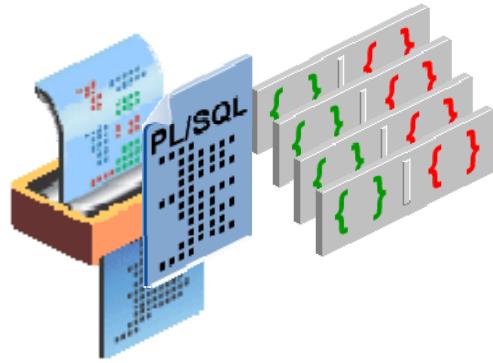
- Initialisierungsparameter `PLSQL_CODE_TYPE` und `PLSQL_OPTIMIZE_LEVEL` für die PL/SQL-Kompilierung verwenden
- PL/SQL-Kompilierungszeitwarnungen verwenden
 - Mit dem Initialisierungsparameter `PLSQL_WARNING`
 - Mit Packageunterprogrammen vom Typ `DBMS_WARNING`

ORACLE®

Copyright © 2013, Oracle und/oder verbundene Unternehmen. All rights reserved. Alle Rechte vorbehalten.

Initialisierungsparameter für die PL/SQL-Kompilierung

- **PLSQL_CODE_TYPE**
- **PLSQL_OPTIMIZE_LEVEL**
- **PLSQL_CCFLAGS**
- **PLSQL_WARNINGS**



ORACLE®

Copyright © 2013, Oracle und/oder verbundene Unternehmen. All rights reserved. Alle Rechte vorbehalten.

In den Releases vor Oracle Database 10g übersetzte der PL/SQL-Compiler den Code in Maschinencode, ohne viele performancebezogene Änderungen anzuwenden. Seit Oracle Database 11g verwendet PL/SQL einen Optimierungscompiler, der Code neu anordnen kann, um die Performance zu verbessern. Sie müssen nichts unternehmen, um die Vorteile des neuen Optimizers nutzen zu können. Er ist standardmäßig aktiviert.

Hinweis

- Der Initialisierungsparameter **PLSQL_CCFLAGS** wird in der Lektion "PL/SQL-Code verwalten" behandelt.
- Der Initialisierungsparameter **PLSQL_WARNINGS** wird im weiteren Verlauf dieser Lektion erläutert.

Initialisierungsparameter für die PL/SQL-Kompilierung

- **PLSQL_CODE_TYPE:** Gibt den Kompilierungsmodus für PL/SQL-Library-Einheiten an

```
PLSQL_CODE_TYPE = { INTERPRETED | NATIVE }
```

- **PLSQL_OPTIMIZE_LEVEL:** Gibt die Optimierungsstufe für das Kompilieren von PL/SQL-Library-Einheiten an

```
PLSQL_OPTIMIZE_LEVEL = { 0 | 1 | 2 | 3 }
```

ORACLE®

Copyright © 2013, Oracle und/oder verbundene Unternehmen. All rights reserved. Alle Rechte vorbehalten.

Parameter **PLSQL_CODE_TYPE**

Dieser Parameter gibt den Kompilierungsmodus für PL/SQL-Library-Einheiten an. Wenn Sie INTERPRETED wählen, werden die PL/SQL-Library-Einheiten in das PL/SQL-Bytecodeformat kompiliert. Module dieser Art werden von der PL/SQL-Interpreter-Engine ausgeführt. Wenn Sie NATIVE wählen, werden die PL/SQL-Library-Einheiten (anonyme PL/SQL-Blöcke der oberen Ebene möglicherweise ausgenommen) in nativen Code (Maschinencode) kompiliert. Module dieser Art werden nativ ohne Interpreteroverhead ausgeführt. Eine Änderung des Parameters wirkt sich nicht auf bereits kompilierte PL/SQL-Library-Einheiten aus. Der Wert dieses Parameters wird dauerhaft mit den einzelnen Library-Einheiten gespeichert. Wenn eine PL/SQL-Library-Einheit nativ kompiliert wird, verwenden auch alle folgenden automatischen Neukompilierungen dieser Library-Einheit die native Kompilierung. Seit Oracle Database 11g ist die native Kompilierung einfacher und besser integriert. Außerdem müssen weniger Initialisierungsparameter eingestellt werden.

Wenn die Kompilierung sehr großer Anwendungen durch den Optimizer-Overhead in seltenen Fällen einmal sehr lange dauert, können Sie die Optimierungsstufe herabsetzen, indem Sie den Initialisierungsparameter **PLSQL_OPTIMIZE_LEVEL** von seinem Standardwert 2 auf 1 einstellen. Noch seltener kann es zu einer Änderung des Exception-Verhaltens kommen, die sich darin äußert, dass eine Exception überhaupt nicht oder früher als erwartet ausgelöst wird. Wenn Sie **PLSQL_OPTIMIZE_LEVEL** auf 0 einstellen, wird jede Neuanordnung des Codes unterbunden.

Parameter PLSQL_OPTIMIZE_LEVEL

Dieser Parameter gibt die Optimierungsstufe für das Kompilieren von PL/SQL-Library-Einheiten an. Je höher die Einstellung für diesen Parameter, desto größeren Aufwand betreibt der Compiler bei der Optimierung von PL/SQL-Library-Einheiten. Folgende Werte sind verfügbar (die Werte 0, 1 und 2 sind seit Oracle 10g verfügbar):

0: Die Auswertungsreihenfolge und damit das Muster von Seiteneffekten, Exceptions und Packageinitialisierungen von Oracle9i und früheren Releases wird übernommen. Darüber hinaus wird die neue semantische Identität von BINARY_INTEGER und PLS_INTEGER entfernt und die früheren Regeln für die Auswertung ganzzahliger Ausdrücke werden wiederhergestellt. Der Code wird zwar etwas schneller als in Oracle9i ausgeführt, doch die Stufe 0 braucht den Großteil der seit Oracle Database 10g erzielten Performancesteigerungen von PL/SQL wieder auf.

1: PL/SQL-Programme werden umfassend optimiert. So werden beispielsweise unnötige Berechnungen und Exceptions beseitigt. Die ursprüngliche Reihenfolge des Quellcodes wird jedoch in der Regel nicht verändert.

2: Umfangreiche Palette moderner Optimierungstechniken, die über die Optimierungen von Einstellung 1 hinaus gehen. Unter anderem kann der Quellcode an eine von seiner ursprünglichen Position relativ weit entfernte Position verschoben werden.

3: Dieser Wert wurde in Oracle Database 11g eingeführt. Er wendet eine breite Palette moderner Optimierungstechniken an, die über den Umfang von Einstellung 2 hinausgehen, automatisch an. Hierzu zählen auch Techniken, die nicht speziell angefordert werden. Dadurch wird das Inlining von Prozeduren möglich. Bei diesem Optimierungsprozess werden Prozeduraufrufe durch eine Kopie des aufzurufenden Prozedurbodys ersetzt. Die kopierte Prozedur wird fast immer schneller ausgeführt als der ursprüngliche Aufruf. Um das Inlining von Unterprogrammen zuzulassen, nehmen Sie entweder den Standardwert 2 des Initialisierungsparameters

PLSQL_OPTIMIZE_LEVEL an oder stellen den Wert 3 ein. Mit PLSQL_OPTIMIZE_LEVEL = 2 müssen Sie jedes Unterprogramm angeben, für das Inlining ausgeführt werden soll. Wenn Sie PLSQL_OPTIMIZE_LEVEL auf 3 einstellen, sucht der PL/SQL-Compiler neben den angegebenen Unterprogrammen nach weiteren Unterprogrammen, für die Inlining ausgeführt wird.

Hinweis: Weitere Informationen zum Inlining finden Sie im Dokument *Oracle Database PL/SQL Language Reference* und in dem von einem Dozenten gehaltenen Kurs *Oracle Database Advanced PL/SQL*.

In der Regel ergibt sich durch die Einstellung 2 für diesen Parameter eine bessere Ausführungsperformance. Wenn der Compiler jedoch bei einem bestimmten Quellmodul sehr langsam läuft oder die Optimierung aus einem bestimmten Grund nicht sinnvoll ist (zum Beispiel bei einer sehr kurzfristig fertigzustellenden Entwicklung), erhalten Sie mit der Einstellung 1 eine fast ebenso gute Kompilierung bei weniger Zeitaufwand. Der Wert dieses Parameters wird dauerhaft mit der Library-Einheit gespeichert.

Hinweis

Der Parameter PLSQL_CODE_TYPE in Oracle Database 10g ersetzt die folgenden veralteten Parameter:

- PLSQL_NATIVE_C_COMPILER
- PLSQL_NATIVE_MAKE_FILE_NAME
- PLSQL_NATIVE_MAKE.Utility
- PLSQL_NATIVE_LINKER

Der Parameter PLSQL_DEBUG wurde in Oracle Database 11g verworfen, da die Generierung von Debugginginformationen durch den PL/SQL-Compiler nicht mehr über diesen Parameter gesteuert werden muss. Debugginginformationen werden immer generiert. Es werden keine speziellen Parameter benötigt.

Compilereinstellungen

Compileroption	Beschreibung
<code>PLSQL_CODE_TYPE</code>	Gibt den Kompilierungsmodus für PL/SQL-Library-Einheiten an
<code>PLSQL_OPTIMIZE_LEVEL</code>	Gibt die Optimierungsstufe für das Kompilieren von PL/SQL-Library-Einheiten an
<code>PLSQL_WARNINGS</code>	Aktiviert oder deaktiviert die Ausgabe von Warnmeldungen durch den PL/SQL-Compiler
<code>PLSQL_CCFLAGS</code>	Steuert Conditional Compilation für jede PL/SQL-Library-Einheit gesondert

Die beste Performance wird im Allgemeinen mit folgenden Einstellungen erzielt:

```
PLSQL_CODE_TYPE = NATIVE
PLSQL_OPTIMIZE_LEVEL = 2
```

ORACLE

Copyright © 2013, Oracle und/oder verbundene Unternehmen. All rights reserved. Alle Rechte vorbehalten.

Der neue Compiler verbessert die Performance von PL/SQL-Code und ermöglicht eine fast doppelt so schnelle Ausführung wie bei einer Oracle8i-Datenbank und eine 1,5- bis 1,75-mal schnellere Ausführung als bei Oracle9i Database Release 2.

Die beste Performance erzielen Sie mit folgenden Compilereinstellungen:

```
PLSQL_CODE_TYPE = NATIVE
PLSQL_OPTIMIZE_LEVEL = 2
```

PL/SQL-Initialisierungsparameter anzeigen

Einstellungen für PL/SQL-Objekte mit den Data Dictionary Views `USER | ALL | DBA_PLSQL_OBJECT_SETTINGS` anzeigen:

```
DESCRIBE USER_PLSQL_OBJECT_SETTINGS
```

DESCRIBE USER_PLSQL_OBJECT_SETTINGS		
Name	Null	Type
NAME	NOT NULL	VARCHAR2(128)
TYPE		VARCHAR2(12)
PLSQL_OPTIMIZE_LEVEL		NUMBER
PLSQL_CODE_TYPE		VARCHAR2(4000)
PLSQL_DEBUG		VARCHAR2(4000)
PLSQL_WARNINGS		VARCHAR2(4000)
NLS_LENGTH_SEMANTICS		VARCHAR2(4000)
PLSQL_CCFLAGS		VARCHAR2(4000)
PLSCOPE_SETTINGS		VARCHAR2(4000)

Copyright © 2013, Oracle und/oder verbundene Unternehmen. All rights reserved. Alle Rechte vorbehalten.

Die Data Dictionary View `USER_PLSQL_OBJECTS_SETTINGS` enthält folgende Spalten:

Owner: Eigentümer des Objekts. Diese Spalte wird in der View `USER_PLSQL_OBJECTS_SETTINGS` nicht angezeigt.

Name: Name des Objekts

Type: Verfügbare Auswahlmöglichkeiten: PROCEDURE, FUNCTION, PACKAGE, PACKAGE BODY, TRIGGER, TYPE oder TYPE BODY

PLSQL_OPTIMIZE_LEVEL: Optimierungsstufe, die beim Kompilieren des Objekts verwendet wurde

PLSQL_CODE_TYPE: Kompilierungsmodus für das Objekt

PLSQL_DEBUG: Gibt an, ob das Objekt zum Debugging kompiliert wurde oder nicht

PLSQL_WARNINGS: Beim Kompilieren des Objekts verwendete Warnungseinstellungen des Compilers

NLS_LENGTH_SEMANTICS: Beim Kompilieren des Objekts verwendete NLS-Längensemantik

PLSQL_CCFLAGS: Beim Kompilieren des Objekts verwendetes Conditional Compilation Flag

PLSCOPE_SETTINGS: Steuert Collections zur Kompilierungszeit, Querverweise und Speicherung der Bezeichner im PL/SQL-Quellcode (neu in Oracle Database 11g)

PL/SQL-Initialisierungsparameter anzeigen und einstellen

```
SELECT name, type, plsql_code_type AS CODE_TYPE,
       plsql_optimize_level AS OPT_LVL
  FROM user_plsql_object_settings;
```

NAME	TYPE	CODE_TYPE	OPT_LVL
1 ADD_COL	PROCEDURE	INTERPRETED	0
2 ADD_DEPARTMENT	PROCEDURE	INTERPRETED	0
3 ADD_DEPARTMENT_NOEX	PROCEDURE	INTERPRETED	0
4 ADD_DEPT	PROCEDURE	INTERPRETED	0
5 ADD_EMPLOYEE	PROCEDURE	INTERPRETED	0
6 ADD_JOB_HISTORY	PROCEDURE	INTERPRETED	1

...

- Wert des Compilerinitialisierungsparameters mit der Anweisung **ALTER SYSTEM** oder **ALTER SESSION** einstellen
- Der Zugriff auf die Werte der Parameter erfolgt bei Ausführung der Anweisung **CREATE OR REPLACE**.

ORACLE

Copyright © 2013, Oracle und/oder verbundene Unternehmen. All rights reserved. Alle Rechte vorbehalten.

Hinweis

- Weitere Informationen zu den Anweisungen **ALTER SYSTEM** und **ALTER SESSION** finden Sie im Dokument *Oracle Database SQL Reference*.
- Die Data Dictionary View-Familie **DBA_STORED_SETTINGS** wurde verworfen und in Oracle Database 10g durch die Data Dictionary View-Familie **DBA_PLSQL_OBJECT_SETTINGS** ersetzt.

PL/SQL-Initialisierungsparameter ändern – Beispiel

```
ALTER SESSION SET PLSQL_OPTIMIZE_LEVEL = 1;
ALTER SESSION SET PLSQL_CODE_TYPE = 'NATIVE';
```

session SET altered.
session SET altered.

```
-- code displayed in the notes page
CREATE OR REPLACE PROCEDURE add_job_history
...
```

@code_11_09_sa.sql

NAME	TYPE	CODE_TYPE	OPT_LVL
1 ADD_COL	PROCEDURE	INTERPRETED	0
2 ADD_DEPARTMENT	PROCEDURE	INTERPRETED	0
3 ADD_DEPARTMENT_NOEX	PROCEDURE	INTERPRETED	0
4 ADD_DEPT	PROCEDURE	INTERPRETED	0
5 ADD_EMPLOYEE	PROCEDURE	INTERPRETED	0
6 ADD_JOB_HISTORY	PROCEDURE	NATIVE	1

ORACLE

Copyright © 2013, Oracle und/oder verbundene Unternehmen. All rights reserved. Alle Rechte vorbehalten.

Sie können den Kompilierungsmodus kompilierter PL/SQL-Objekte von interpretiert auf nativ ändern. Stellen Sie hierfür zunächst den Parameter `PLSQL_CODE_TYPE` auf `NATIVE` und optional auch die anderen Parameter ein, und kompilieren Sie das Programm neu. Um die native Kompilierung des gesamten PL/SQL-Codes durchzusetzen, müssen Sie alle Codeblöcke nacheinander neu kompilieren. Mithilfe spezieller Skripte (im Verzeichnis `rdmbs/admin`) ist es möglich, eine vollständig native Kompilierung (`dbmsupgnv.sql`) oder eine vollständig interpretierte Kompilierung (`dbmsupgini.sql`) zu erreichen. Die Prozedur `add_job_history` wird wie folgt erstellt:

```
CREATE OR REPLACE PROCEDURE add_job_history
( p_emp_id          job_history.employee_id%type
, p_start_date      job_history.start_date%type
, p_end_date        job_history.end_date%type
, p_job_id          job_history.job_id%type
, p_department_id   job_history.department_id%type )
IS
BEGIN
  INSERT INTO job_history (employee_id, start_date,
                           end_date, job_id, department_id)
    VALUES(p_emp_id, p_start_date, p_end_date,
           p_job_id, p_department_id);
END add_job_history;
/
```

Lektionsagenda

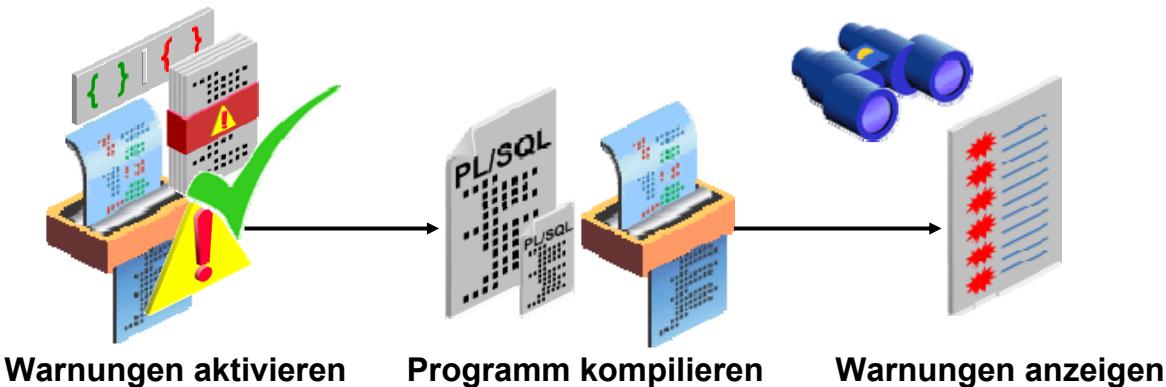
- Initialisierungsparameter `PLSQL_CODE_TYPE` und `PLSQL_OPTIMIZE_LEVEL` für die PL/SQL-Kompilierung verwenden
- **PL/SQL-Kompilierungszeitwarnungen verwenden:**
 - Mit dem Initialisierungsparameter `PLSQL_WARNING`
 - Mit Packageunterprogrammen vom Typ `DBMS_WARNING`

ORACLE

Copyright © 2013, Oracle und/oder verbundene Unternehmen. All rights reserved. Alle Rechte vorbehalten.

PL/SQL-Kompilierungszeitwarnungen für Unterprogramme – Überblick

In Oracle Database 10g wurde der PL/SQL-Compiler durch eine Warnfunktion für Unterprogramme erweitert.



ORACLE

Copyright © 2013, Oracle und/oder verbundene Unternehmen. All rights reserved. Alle Rechte vorbehalten.

Um Programme robuster zu machen und Probleme zur Laufzeit zu vermeiden, können Sie nun prüfen, ob bestimmte Bedingungen vorliegen, die eine Warnung auslösen können. Diese Bedingungen sind nicht schwerwiegend genug, um einen Fehler zu generieren und die Kompilierung eines Unterprogramms zu verhindern. Sie können jedoch darauf hindeuten, dass im Unterprogramm ein undefiniertes Ergebnis generiert oder ein Performanceproblem verursacht wird.

In Releases vor Oracle Database 10g hatte die Kompilierung von PL/SQL-Programmen zwei Ergebnisse:

- Erfolg - die Kompilierung generierte eine gültige Einheit
- Fehler - Kompilierungsfehler weisen auf Syntax- oder Semantikfehler im Programm hin

Doch auch erfolgreich kompilierte Programme können gegen Best Practices verstößen oder Potenzial für eine Effizienzsteigerung bieten. In Oracle Database 10g wurde ein neues benutzerfreundliches Feature eingeführt, das in solchen Fällen Warnmeldungen ausgeben kann. Mithilfe der Compilerwarnungen können Entwickler häufige Fehler bei der Codierung vermeiden und damit die Produktivität verbessern.

PL/SQL unterstützt die Übergabe der Parameter `IN OUT` und `OUT` per Wert oder Referenz über den Compiler-Hint `NOCOPY`. Die Parameterübergabe per Wert ist schon an sich weniger effizient, weil dabei mehrere Kopien der Daten erstellt werden. Ab Oracle Database 11g erkennt und empfiehlt der Compiler automatisch die Verwendung des Hints `NOCOPY` mit den Parametertypen `LOB`, `RECORD` oder `COLLECTION`.

Mit dem Warnungsfeature des PL/SQL-Compilers kann die Kompilierung eines PL/SQL-Programms zu weiteren Ergebnissen führen:

- Erfolg mit Kompilierungswarnungen
- Fehler mit Kompilierungsfehlern und -warnungen

Der Compiler kann auch bei einer erfolgreichen Kompilierung Warnmeldungen ausgeben. Ein Kompilierungsfehler muss korrigiert werden, damit die Stored Procedure verwendet werden kann. Eine Warnung dagegen nur Informationscharakter.

Beispiele für Warnmeldungen

SP2-0804: Mit Kompilierungswarnungen erstellte Prozedur

PLW-07203: Der Parameter 'IO_TBL' kann vom Compiler-Hint NOCOPY profitieren

Compilerwarnungen – Vorteile

- **Programme robuster gestalten und Probleme zur Laufzeit vermeiden**
- **Potenzielle Performanceprobleme identifizieren**
- **Faktoren identifizieren, die zu nicht definierten Ergebnissen führen**



ORACLE

Copyright © 2013, Oracle und/oder verbundene Unternehmen. All rights reserved. Alle Rechte vorbehalten.

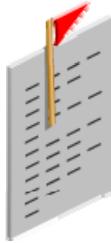
Mit Compilerwarnungen können Sie:

- Ihre Programme stabiler machen und Probleme zur Laufzeit vermeiden
- potenzielle Performanceprobleme identifizieren
- Faktoren identifizieren, die zu nicht definierten Ergebnissen führen

Hinweis

- Sie können die Prüfung auf bestimmte Warnungsbedingungen aktivieren, die nicht schwerwiegend genug sind, um Fehler zu generieren und die Kompilierung von Unterprogrammen zu verhindern.
- Warnmeldungen können während der Kompilierung von PL/SQL-Unterprogrammen ausgegeben werden. Anonyme Blöcke generieren keine Warnungen.
- Alle PL/SQL-Warnungen verwenden das Präfix PLW.

PL/SQL-Kompilierungswarnungen – Kategorien



SEVERE



PERFORMANCE



INFORMATIONAL



ALL

ORACLE

Copyright © 2013, Oracle und/oder verbundene Unternehmen. All rights reserved. Alle Rechte vorbehalten.

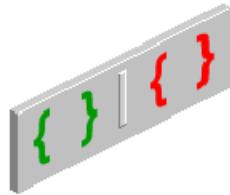
PL/SQL-Warnmeldungen sind in Kategorien unterteilt, sodass Sie ähnliche Warnungen bei der Kompilierung gruppenweise unterdrücken oder anzeigen können:

- SEVERE: Meldungen zu Bedingungen, die zu unerwartetem Verhalten oder falschen Ergebnissen führen können, beispielsweise Probleme bei der Aliaserstellung für Parameter
- PERFORMANCE: Meldungen zu Bedingungen, die zu Performanceproblemen führen können. Beispiel: Übergabe eines VARCHAR2-Wertes an die Spalte NUMBER in der Anweisung INSERT
- INFORMATIONAL: Meldungen zu Bedingungen, die sich nicht auf Performance oder Korrektheit auswirken, aber geändert werden können, um die Übersichtlichkeit und Verwaltbarkeit des Codes zu erhöhen. Beispiel: Unerreichbarer Code, der niemals ausgeführt werden kann

Warnstufen einstellen

Sie können Warnstufen wie folgt einstellen:

- Deklarativ:
 - Mit dem Initialisierungsparameter **PLSQL_WARNINGS**
- Programmgesteuert:
 - Mit dem Package **DBMS_WARNING**



Initialisierungsparameter
PLSQL_WARNINGS



Package
DBMS_WARNING

ORACLE

Copyright © 2013, Oracle und/oder verbundene Unternehmen. All rights reserved. Alle Rechte vorbehalten.

Sie können die Warnstufen des Compilers wie folgt einstellen:

Mit dem Initialisierungsparameter **PLSQL_WARNINGS**

Mit der Einstellung für **PLSQL_WARNINGS** wird die Ausgabe von Warnmeldungen durch den PL/SQL-Compiler aktiviert oder deaktiviert. Außerdem wird festgelegt, welche Warnmeldungen als Fehler angezeigt werden. Die Einstellungen für den Parameter **PLSQL_WARNINGS** werden zusammen mit den einzelnen kompilierten Unterprogrammen gespeichert. Mit dem Initialisierungsparameter **PLSQL_WARNINGS** können Sie:

- die Erfassung aller Warnungen, der Warnungen einer gewählten Kategorie oder bestimmter Warnmeldungen aktivieren oder deaktivieren
- alle Warnungen, gewählte Kategorien von Warnungen oder bestimmte Warnmeldungen als Fehler behandeln
- die vorgenannten Optionen kombinieren

Mit dem Schlüsselwort **All** können Sie zusammenfassend alle Warnmeldungen angeben:

SEVERE, **PERFORMANCE** und **INFORMATIONAL**

Mit dem Package **DBMS_WARNING**

Das Package **DBMS_WARNING** bietet eine Möglichkeit, das Verhalten von PL/SQL-Warnmeldungen zu beeinflussen. Im Einzelnen kann die Einstellung des Initialisierungsparameters **PLSQL_WARNINGS** gelesen und geändert werden, der steuert, welche Warnungen unterdrückt, angezeigt oder als Fehler behandelt werden. Dieses Package stellt die Schnittstelle zum Abfragen, Ändern und Löschen aktueller System- oder Sessioneinstellungen dar. Es wird im weiteren Verlauf dieser Lektion erläutert.

Compilerwarnstufen einstellen – mit PLSQL_WARNINGS

```
ALTER [SESSION|SYSTEM]
PLSQL_WARNINGS = 'value_clause1'[ , 'value_clause2']...
```

```
value_clause = Qualifier Value : Modifier Value
```

```
Qualifier Value = { ENABLE | DISABLE | ERROR }

Modifier Value =
{ ALL | SEVERE | INFORMATIONAL | PERFORMANCE |
{ integer | (integer [, integer ] ...) } }
```

ORACLE

Copyright © 2013, Oracle und/oder verbundene Unternehmen. All rights reserved. Alle Rechte vorbehalten.

Einstellungen von Compilerwarnungen ändern

Der Parameterwert besteht aus einer durch Kommas getrennten Liste von Qualifier- und Modifier-Schlüsselwörtern in Anführungszeichen. Die Schlüsselwörter sind durch Doppelpunkte getrennt. Verfügbare Qualifier-Werte: ENABLE, DISABLE und ERROR. Der Modifier-Wert ALL gilt für alle Warnmeldungen. SEVERE, INFORMATIONAL und PERFORMANCE gelten für Meldungen in der jeweiligen Kategorie und eine Liste mit Ganzzahlen für bestimmte Warnmeldungen.

Mögliche Werte für ENABLE, DISABLE und ERROR:

- ALL
- SEVERE
- INFORMATIONAL
- PERFORMANCE
- numeric_value

Werte für numeric_value:

- Zwischen 5000-5999 für SEVERE
- Zwischen 6000-6249 für INFORMATIONAL
- Zwischen 7000-7249 für PERFORMANCE

Compilerwarnstufen einstellen – mit PLSQL_WARNINGS: Beispiele

```
ALTER SESSION
SET plsql_warnings = 'enable:severe',
  'enable:performance',
  'disable:informational';
```

session SET altered.

```
ALTER SESSION
SET plsql_warnings = 'enable:severe';
```

session SET altered.

```
ALTER SESSION SET PLSQL_WARNINGS='ENABLE:SEVERE',
  'DISABLE:PERFORMANCE' , 'ERROR:05003';
```

session SET altered.

ORACLE

Copyright © 2013, Oracle und/oder verbundene Unternehmen. All rights reserved. Alle Rechte vorbehalten.

Mit dem Befehl **ALTER SESSION** oder **ALTER SYSTEM** können Sie den Initialisierungsparameter **PLSQL_WARNINGS** ändern. Auf der Folie werden verschiedene Beispiele für das Aktivieren und Deaktivieren der Compilerwarnungen gezeigt.

1. Beispiel

Mit diesem Beispiel aktivieren Sie Warnungen vom Typ **SEVERE** und **PERFORMANCE** und deaktivieren Warnungen vom Typ **INFORMATIONAL**.

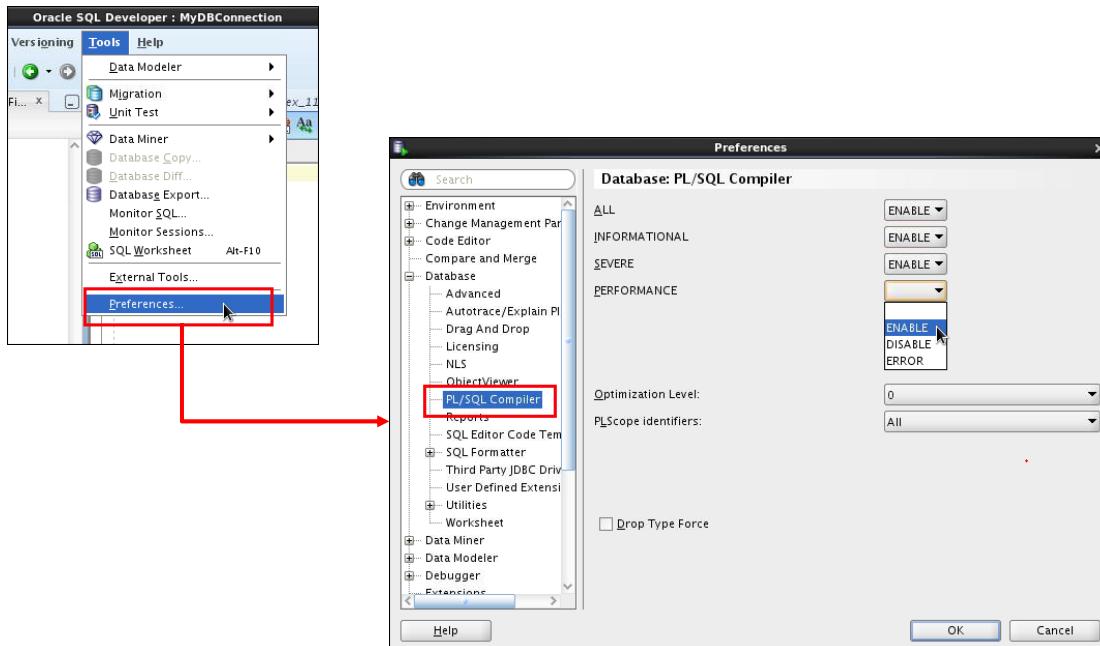
2. Beispiel

Mit dem zweiten Beispiel aktivieren Sie nur Warnungen vom Typ **SEVERE**.

3. Beispiel

Bestimmte Meldungen können statt als Fehler als Warnungen behandelt werden. In diesem Beispiel wissen Sie, dass die Warnmeldung **PLW-05003** auf ein schwerwiegendes Problem im Code hindeutet. Sie nehmen daher **ERROR : 05003** in die Einstellung **PLSQL_WARNINGS** auf, damit diese Bedingung statt einer Warnmeldung eine Fehlermeldung (**PLS_05003**) auslöst. Eine Fehlermeldung führt zu einem Kompilierungsfehler. In diesem Beispiel deaktivieren Sie außerdem Warnungen vom Typ **PERFORMANCE**.

Compilerwarnstufen einstellen – mit PLSQL_WARNINGS in SQL Developer



ORACLE

Copyright © 2013, Oracle und/oder verbundene Unternehmen. All rights reserved. Alle Rechte vorbehalten.

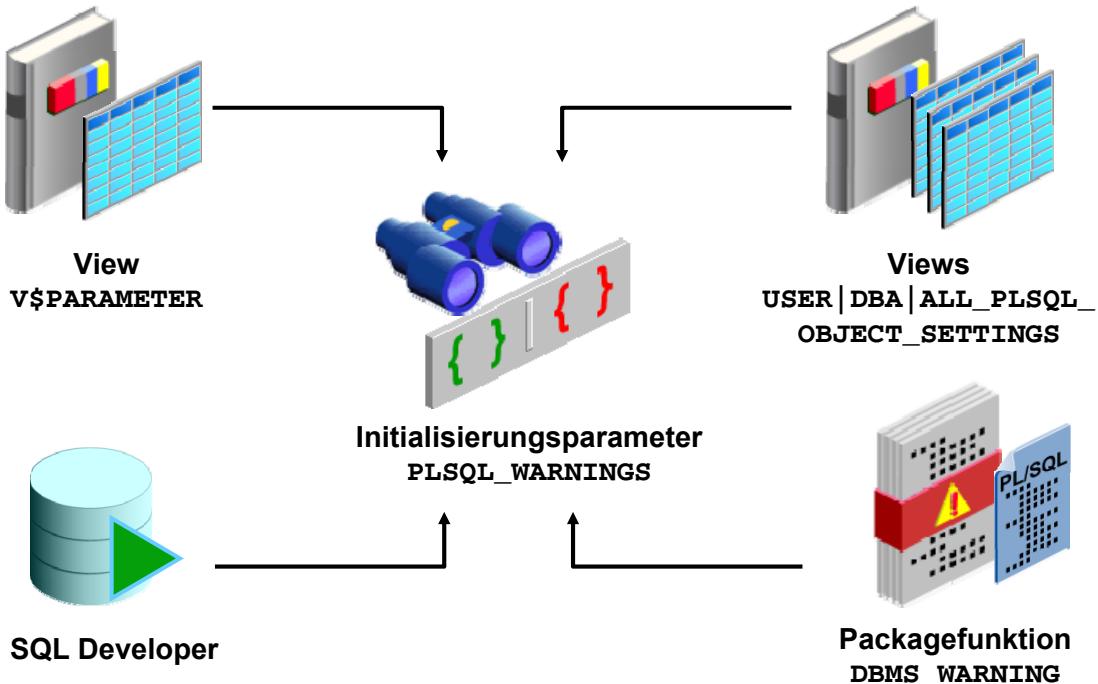
Der PL/SQL-Compiler enthält Optionen zur Kompilierung von PL/SQL-Unterprogrammen. Nur wenn das Kontrollkästchen **Generate PL/SQL Debug Information** aktiviert ist, werden PL/SQL-Debugginginformationen in den kompilierten Code aufgenommen. Nur wenn der Code mit Debugginginformationen generiert wurde, können Sie an einzelnen Codezeilen stoppen und mit dem Debugger auf Variablen zugreifen.

Kategorien für PL/SQL-Kompilierungswarnungen in SQL Developer einstellen und anzeigen

Sie können die Anzeige von Meldungen vom Typ **Informational**, **Severe** und **Performance** steuern. Der Typ **ALL** überschreibt alle Einzelspezifikationen für die anderen Meldungstypen. Folgende Einstellungen sind für die einzelnen Meldungstypen möglich:

- **Keine Eingabe (leer)**: Der für **ALL** angegebene Wert wird verwendet. Ist kein Wert angegeben, wird der Oracle-Standardwert verwendet
- **Enable**: Alle Meldungen dieser Kategorie werden angezeigt.
- **Disable**: Die Anzeige aller Meldungen dieser Kategorie wird deaktiviert.
- **Error**: Nur die Fehlermeldungen dieser Kategorie werden angezeigt.

Aktuelle Einstellung von PLSQL_WARNINGS anzeigen



Copyright © 2013, Oracle und/oder verbundene Unternehmen. All rights reserved. Alle Rechte vorbehalten.

ORACLE

Aktuellen Wert des Parameters `PLSQL_WARNINGS` anzeigen

Sie können die aktuelle Einstellung des Parameters `PLSQL_WARNINGS` ermitteln, indem Sie eine `SELECT`-Anweisung für die View `V$PARAMETER` absetzen. Beispiel:

```
ALTER SESSION SET plsql_warnings = 'enable:severe',
    'enable:performance', 'enable:informational';
```

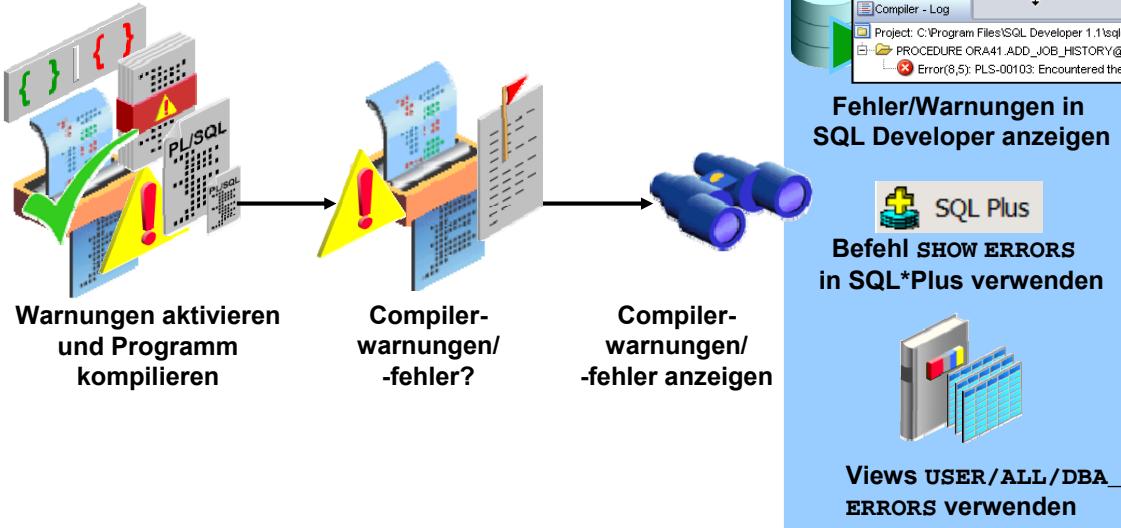
```
Session altered.
```

```
SELECT value FROM v$parameter WHERE name='plsql_warnings';
VALUE
-----
ENABLE:ALL
```

Alternativ können Sie mit dem Package DBMS_WARNING.GET_WARNING_SETTING_STRING und der gleichnamigen Prozedur die aktuellen Einstellungen für den Parameter PLSQL_WARNINGS abrufen:

```
DECLARE s VARCHAR2(1000);
BEGIN
    s := dbms_warning.get_warning_setting_string();
    dbms_output.put_line (s);
END;
/
anonymous block completed
ENABLE:ALL
```

Compilerwarnungen anzeigen – mit SQL Developer, SQL*Plus oder Data Dictionary Views



ORACLE®

Copyright © 2013, Oracle und/oder verbundene Unternehmen. All rights reserved. Alle Rechte vorbehalten.

Compilerwarnungen anzeigen

Mit SQL*Plus können Sie Warnungen anzeigen, die als Ergebnis der Kompilierung eines PL/SQL-Blockes ausgegeben werden. SQL*Plus zeigt an, dass eine Kompilierungswarnung ausgegeben wurde. Die Meldung **SP2-08xx: <object> created with compilation warnings** wird für Objekte angezeigt, die mit dem Modifier PERFORMANCE, INFORMATIONAL oder SEVERE kompiliert wurden. Zwischen den drei Modifiern wird kein Unterschied gemacht. Sie müssen die Compilerwarnungen aktivieren, bevor Sie das Programm kompilieren. Die Compilerwarnmeldungen können Sie mit einer der folgenden Methoden anzeigen:

Mit dem SQL*Plus-Befehl SHOW ERRORS

Mit diesem Befehl werden alle Compilerfehler einschließlich der neuen Compilerwarnungen und Informationsmeldungen angezeigt. Dieser Befehl wird unmittelbar nach der Verwendung des Befehls CREATE [PROCEDURE | FUNCTION | PACKAGE] aufgerufen. Mit dem Befehl SHOW ERRORS werden Warnungen und Compilerfehler angezeigt. Bei Aufruf von SHOW ERRORS werden neue Compilerwarnungen und -informationsmeldungen mit Compilerfehlern kombiniert.

Mit den Data Dictionary Views

PL/SQL-Compilerwarnungen können in den Data Dictionary Views USER_ | ALL_ | DBA_ERRORS angezeigt werden. Die Spalte ATTRIBUTES dieser Views enthält das neue Attribut WARNING. Die Warnmeldung wird in der Spalte TEXT angezeigt.

SQL*Plus-Warnmeldungen – Beispiel

```
CREATE OR REPLACE PROCEDURE bad_proc(p_out ...) IS
BEGIN
  . . .;
END;
/
SP2-0804: Procedure created with compilation warnings.
```

```
SHOW ERRORS;
Errors for PROCEDURE BAD_PROC:

LINE/COL      ERROR
-----
6/24          PLW-07203: parameter 'p_out' may benefit
                  from use of the NOCOPY compiler hint
```

ORACLE

Copyright © 2013, Oracle und/oder verbundene Unternehmen. All rights reserved. Alle Rechte vorbehalten.

Mit dem Befehl `SHOW ERRORS` in SQL*Plus zeigen Sie die Kompilierungsfehler einer Stored Procedure an. Wenn Sie diese Option ohne Argumente angeben, zeigt SQL*Plus die Kompilierungsfehler für die zuletzt erstellte oder geänderte Stored Procedure an. Wenn SQL*Plus eine Kompilierungswarnmeldung anzeigt, nachdem Sie eine Stored Procedure erstellt oder geändert haben, können Sie mit `SHOW ERRORS`-Befehlen weitere Informationen erhalten.

Mit der Unterstützung von PL/SQL-Warnungen wird die Palette der Rückmeldungen um eine dritte Meldung erweitert:

`SP2-08xx: <object> created with compilation warnings.`

Damit können Sie zwischen Kompilierungswarnungen und Kompilierungsfehler unterscheiden. Fehler müssen korrigiert werden, wenn Sie die Stored Procedure verwenden möchten, Warnungen hingegen dienen nur zu Informationszwecken.

Anhand des Präfix `SP2` können Sie die entsprechende Meldungsnummer im Handbuch *SQL*Plus User's Guide and Reference* nachschlagen und nach Ursachen und geeigneten Maßnahmen für die jeweilige Meldung suchen.

Hinweis: Die Compilerfehler und -warnungen können auch in den Data Dictionary Views `USER_ | ALL_ | DBA_ERRORS` angezeigt werden.

Richtlinien für die Verwendung von **PLSQL_WARNINGS**

- **Die Einstellungen für den Parameter PLSQL_WARNINGS werden zusammen mit den einzelnen kompilierten Unterprogrammen gespeichert.**
- **Wenn Sie das Unterprogramm mit einer der folgenden Anweisungen neu kompilieren, werden die für die Session gültigen aktuellen Einstellungen verwendet:**
 - CREATE OR REPLACE
 - ALTER . . . COMPILE
- **Wenn Sie das Unterprogramm mit der Anweisung ALTER . . . COMPILE mit der Klausel REUSE SETTINGS rekompilieren, wird die mit dem Programm gespeicherte ursprüngliche Einstellung verwendet.**



Copyright © 2013, Oracle und/oder verbundene Unternehmen. All rights reserved. Alle Rechte vorbehalten.

Wie bereits erwähnt, kann der Parameter PLSQL_WARNINGS auf Session- oder Systemebene eingestellt werden.

Die Einstellungen für den Parameter PLSQL_WARNINGS werden zusammen mit den einzelnen kompilierten Unterprogrammen gespeichert. Wenn Sie das Unterprogramm mit der Anweisung CREATE OR REPLACE neu kompilieren, werden die aktuellen Einstellungen für diese Session verwendet. Bei einer Neukompilierung des Unterprogramms mit einer Anweisung vom Typ ALTER . . . COMPILE wird die aktuelle Sessioneinstellung verwendet, sofern Sie nicht die Klausel REUSE SETTINGS in der Anweisung angeben, die die mit dem Unterprogramm gespeicherte ursprüngliche Einstellung verwendet.

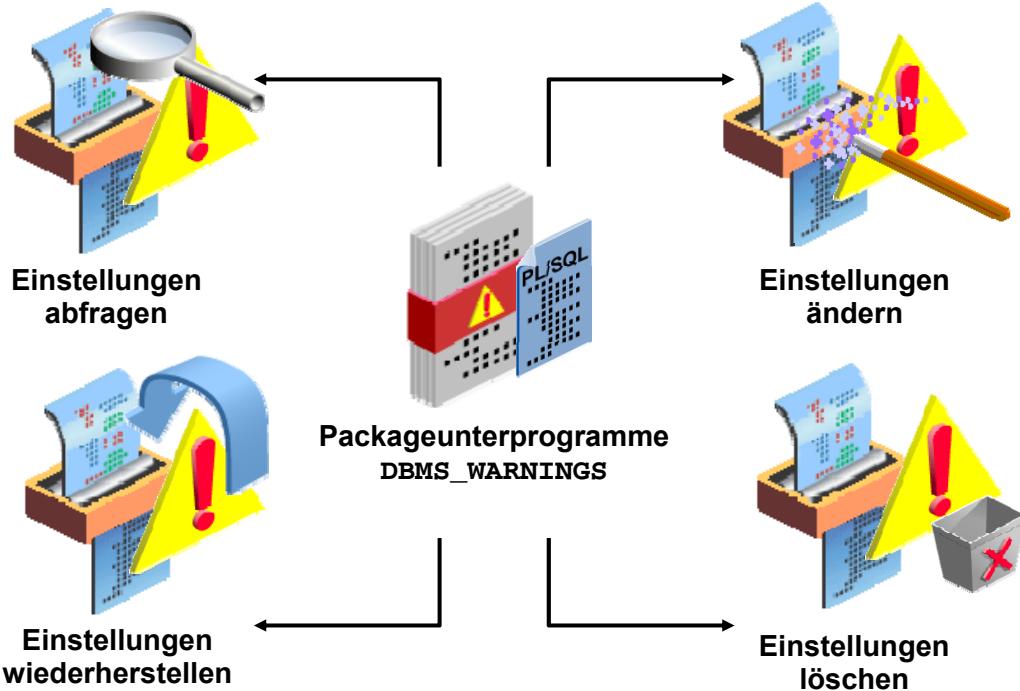
Lektionsagenda

- Initialisierungsparameter `PLSQL_CODE_TYPE` und `PLSQL_OPTIMIZE_LEVEL` für die PL/SQL-Kompilierung verwenden
- **PL/SQL-Kompilierungswarnungen verwenden**
 - Mit dem Initialisierungsparameter `PLSQL_WARNING`
 - Mit Packageunterprogrammen vom Typ `DBMS_WARNING`

ORACLE®

Copyright © 2013, Oracle und/oder verbundene Unternehmen. All rights reserved. Alle Rechte vorbehalten.

Compilerwarnstufen einstellen – mit dem Package DBMS_WARNING



Copyright © 2013, Oracle und/oder verbundene Unternehmen. All rights reserved. Alle Rechte vorbehalten.

Verwenden Sie das Package DBMS_WARNING, um die auf System- oder Sessionebene aktuellen PL/SQL-Warnungseinstellungen programmgesteuert zu ändern. Das Package DBMS_WARNING bietet eine Möglichkeit, das Verhalten von PL/SQL-Warnmeldungen zu beeinflussen. Im Einzelnen kann die Einstellung des Initialisierungsparameters `PLSQL_WARNINGS` gelesen und geändert werden, der steuert, welche Warnungen unterdrückt, angezeigt oder als Fehler behandelt werden. Dieses Package ist die Schnittstelle, über die aktuelle System- oder Sessioneinstellungen abgefragt, bearbeitet oder gelöscht werden können.

Das Package DBMS_WARNING ist hilfreich, wenn Sie eine Entwicklungsumgebung erstellen, die PL/SQL-Unterprogramme kompiliert. Mithilfe der Schnittstellenroutinen des Packages können Sie PL/SQL-Warnmeldungen programmgesteuert entsprechend Ihren Anforderungen steuern.

PL/SQL-Komplizierungswarnungen für Unterprogramme – Überblick: Beispiel

Sie erstellen Code zur Komplizierung von PL/SQL-Code. Sie wissen, dass der Compiler Performancewarnungen ausgibt, wenn Collection-Variablen als Parameter OUT oder IN OUT ohne Angabe des Hints NOCOPY übergeben werden. Die allgemeine Umgebung, die Ihr Komplizierung utility aufruft, verfügt möglicherweise über entsprechende Warnstufeneinstellungen. Unabhängig davon geben Ihre Geschäftsregeln vor, dass die eingestellte aufrufende Umgebung beibehalten werden muss und die Warnungen vom Komplizierungsprozess zu unterdrücken sind. Indem Sie im Package DBMS_WARNING Unterprogramme aufrufen, können Sie die aktuellen Warnungseinstellungen anzeigen, die Einstellungen entsprechend Ihren Geschäftsanforderungen ändern und die ursprünglichen Einstellungen wiederherstellen, nachdem die Verarbeitung abgeschlossen wurde.

Wenn Sie den Befehl ALTER SESSION oder ALTER SYSTEM zur Einstellung des Parameters PLSQL_WARNINGS verwenden, ersetzt der neu angegebene Wert den vorherigen Wert vollständig. Das neue Package DBMS_WARNING ist seit Oracle Database 10g verfügbar. Es enthält Schnittstellen für die Abfrage und schrittweise Änderung der Einstellung für den Parameter PLSQL_WARNINGS sowie die genauere Anpassung an Ihre Anforderungen.

Mit dem Package DBMS . WARNING können Sie den Parameter PLSQL_WARNINGS schrittweise ändern. So können Sie genau die gewünschten Warnungen einstellen, ohne überlegen zu müssen, wie Sie die Werte von Warnungen aufbewahren, die für Sie uninteressant sind. Beispiel: Beispiel: Der DBA aktiviert in der Initialisierungsparameterdatei nur SEVERE-Warnungen für die gesamte Datenbank. Der Entwickler, der neuen Code testet, möchte nur bestimmte Performance- und Informationsmeldungen anzeigen. In diesem Fall kann der Entwickler das Package DBMS_WARNING verwenden, um schrittweise die Warnungen hinzuzufügen, die er anzeigen möchte. So sieht er nur noch die für ihn interessanten Meldungen, ohne die Einstellungen des DBA ersetzen zu müssen.

Package DBMS_WARNING – Unterprogramme

Szenario	Zu verwendende Unterprogramme
Warnungen einstellen	ADD_WARNING_SETTING_CAT (Prozedur) ADD_WARNING_SETTING_NUM (Prozedur)
Warnungen abfragen	GET_WARNING_SETTING_CAT (Funktion) GET_WARNING_SETTING_NUM (Funktion) GET_WARNING_SETTING_STRING (Funktion)
Warnungen ersetzen	ADD_WARNING_SETTING_STRING (Prozedur)
Namen der Warnungskategorien abrufen	GET_CATEGORY (Funktion)

Copyright © 2013, Oracle und/oder verbundene Unternehmen. All rights reserved. Alle Rechte vorbehalten.

Package DBMS_WARNING – Unterprogramme

Folgende Unterprogramme sind im Package DBMS_WARNING verfügbar:

- ADD_WARNING_SETTING_CAT: Ändert die Warnungseinstellungen der aktuellen Session oder des Systems für die zuvor angegebene Warnungskategorie (`warning_category`)
- ADD_WARNING_SETTING_NUM: Ändert die Warnungseinstellungen der aktuellen Session oder des Systems für die zuvor angegebene Warnungsnummer (`warning_number`)
- GET_CATEGORY: Gibt den Kategorienamen für eine Meldungsnummer zurück
- GET_WARNING_SETTING_CAT: Gibt die spezifische Warnungskategorie in der Session zurück
- GET_WARNING_SETTING_NUM: Gibt die spezifische Warnungsnummer in der Session zurück
- GET_WARNING_SETTING_STRING: Gibt die gesamte Warnungszeichenfolge für die aktuelle Session zurück
- SET_WARNING_SETTING_STRING: Ersetzt vorherige Einstellungen durch den neuen Wert

Hinweis: Weitere Informationen zum Package und zu den Unterprogrammen finden Sie in den Handbüchern *Oracle Database PL/SQL Packages and Types Reference*/*Oracle Database PL/SQL Packages and Types Reference* (je nach verwendeter Version).

DBMS_WARNING-Prozeduren – Syntax, Parameter und zulässige Werte

```
EXECUTE DBMS_WARNING.ADD_WARNING_SETTING_CAT (-
    warning_category      IN      VARCHAR2,
    warning_value        IN      VARCHAR2,
    scope                IN      VARCAHR2);
```

```
EXECUTE DBMS_WARNING.ADD_WARNING_SETTING_NUM (-
    warning_number        IN      NUMBER,
    warning_value         IN      VARCHAR2,
    scope                IN      VARCAHR2);
```

```
EXECUTE DBMS_WARNING.SET_WARNING_SETTING_STRING (-
    warning_value        IN      VARCHAR2,
    scope                IN      VARCHAR2);
```

ORACLE

Copyright © 2013, Oracle und/oder verbundene Unternehmen. All rights reserved. Alle Rechte vorbehalten.

warning_category ist der Name der Kategorie. Zulässige Werte:

- ALL
- INFORMATIONAL
- SEVERE
- PERFORMANCE

warning_value ist der Wert für die Kategorie. Zulässige Werte:

- ENABLE
- DISABLE
- ERROR

warning_number ist die Nummer der Warnmeldung. Zulässige Werte sind alle gültigen Warnungsnummern.

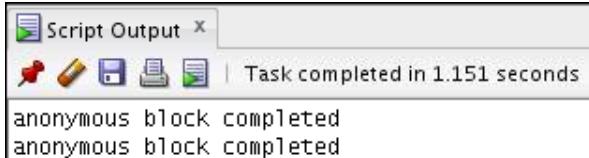
scope gibt an, ob die Änderungen im Session- oder Systemkontext durchgeführt werden. Zulässige Werte sind SESSION und SYSTEM. Zur Verwendung von SYSTEM ist die Berechtigung ALTER SYSTEM erforderlich.

DBMS_WARNING-Prozeduren – Beispiel

```
-- Establish the following warning setting string in the
-- current session:
-- ENABLE:INFORMATIONAL,
-- DISABLE:PERFORMANCE,
-- ENABLE:SEVERE

EXECUTE DBMS_WARNING.SET_WARNING_SETTING_STRING(
    'ENABLE:ALL', 'SESSION');

EXECUTE DBMS_WARNING.ADD_WARNING_SETTING_CAT(
    'PERFORMANCE', 'DISABLE', 'SESSION');
```



ORACLE

Copyright © 2013, Oracle und/oder verbundene Unternehmen. All rights reserved. Alle Rechte vorbehalten.

DBMS_WARNING-Prozeduren – Beispiel

Mit der Prozedur SET_WARNING_SETTING_STRING können Sie genau eine Warnungseinstellung festlegen. Für mehrere Warnungseinstellungen gehen Sie wie folgt vor:

1. Rufen Sie SET_WARNING_SETTING_STRING auf, um die erste Zeichenfolge für die Warnungseinstellung festzulegen.
2. Rufen Sie wiederholt ADD_WARNING_SETTING_CAT (oder ADD_WARNING_SETTING_NUM) auf, um der ersten Zeichenfolge weitere Einstellungen hinzuzufügen.

Im Beispiel auf der Folie wird die folgende Zeichenfolge für die Warnungseinstellung in der aktuellen Session festgelegt:

ENABLE:INFORMATIONAL,DISABLE:PERFORMANCE,ENABLE:SEVERE

DBMS_WARNING-Funktionen – Syntax, Parameter und zulässige Werte

```
DBMS_WARNING.GET_WARNING_SETTING_CAT (
    warning_category IN VARCHAR2) RETURN warning_value;
```

```
DBMS_WARNING.GET_WARNING_SETTING_NUM (
    warning_number IN NUMBER) RETURN warning_value;
```

```
DBMS_WARNING.GET_WARNING_SETTING_STRING
RETURN pls_integer;
```

```
DBMS_WARNING.GET_CATEGORY (
    warning_number IN pls_integer) RETURN VARCHAR2;
```

ORACLE

Copyright © 2013, Oracle und/oder verbundene Unternehmen. All rights reserved. Alle Rechte vorbehalten.

`warning_category` ist der Name der Kategorie. Zulässige Werte:

- ALL
- INFORMATIONAL
- SEVERE
- PERFORMANCE

`warning_number` ist die Nummer der Warnmeldung. Zulässige Werte sind alle gültigen Warnungsnummern.

`scope` gibt an, ob die Änderungen im Session- oder Systemkontext durchgeführt werden. Zulässige Werte sind SESSION und SYSTEM. Zur Verwendung von SYSTEM ist die Berechtigung ALTER SYSTEM erforderlich.

Hinweis: Verwenden Sie die Funktion GET_WARNING_SETTING_STRING, wenn Sie nicht die Berechtigung SELECT für die feste Tabelle v\$parameter oder v\$parameter2 besitzen oder die Warnungszeichenfolge selbst parsen und anschließend den neuen Wert mit SET_WARNING_SETTING_STRING ändern und einstellen möchten.

DBMS_WARNING-Funktionen – Beispiel

```
-- Determine the current session warning settings
SET SERVEROUTPUT ON
EXECUTE DBMS_OUTPUT.PUT_LINE( -
DBMS_WARNING.GET_WARNING_SETTING_STRING);
```

```
anonymous block completed
ENABLE:INFORMATIONAL,DISABLE:PERFORMANCE,ENABLE:SEVERE
```

```
-- Determine the category for warning message number
-- PLW-07203
SET SERVEROUTPUT ON
EXECUTE DBMS_OUTPUT.PUT_LINE( -
DBMS_WARNING.GET_CATEGORY(7203));
```

```
anonymous block completed
PERFORMANCE
```

ORACLE®

Copyright © 2013, Oracle und/oder verbundene Unternehmen. All rights reserved. Alle Rechte vorbehalten.

Hinweis

Die Meldungsnummern müssen als positive Ganzzahlen angegeben werden, da der Parameter GET_CATEGORY den Datentyp PLS_INTEGER aufweist (d. h. positive Ganzahlwerte zulässt).

DBMS_WARNING – Beispiel

```
CREATE OR REPLACE PROCEDURE compile_code(p_pkg_name VARCHAR2) IS
    v_warn_value    VARCHAR2(200);
    v_compile_stmt VARCHAR2(200) := 
        'ALTER PACKAGE '|| p_pkg_name ||' COMPILE';

BEGIN
    v_warn_value := DBMS_WARNING.GET_WARNING_SETTING_STRING;
    DBMS_OUTPUT.PUT_LINE('Current warning settings: ' ||
        v_warn_value);
    DBMS_WARNING.ADD_WARNING_SETTING_CAT(
        'PERFORMANCE', 'DISABLE', 'SESSION');
    DBMS_OUTPUT.PUT_LINE('Modified warning settings: ' ||
        DBMS_WARNING.GET_WARNING_SETTING_STRING);
    EXECUTE IMMEDIATE v_compile_stmt;
    DBMS_WARNING.SET_WARNING_SETTING_STRING(v_warn_value,
        'SESSION');
    DBMS_OUTPUT.PUT_LINE('Restored warning settings: ' ||
        DBMS_WARNING.GET_WARNING_SETTING_STRING);
END;
/
```

ORACLE®

Copyright © 2013, Oracle und/oder verbundene Unternehmen. All rights reserved. Alle Rechte vorbehalten.

Die Prozedur `compile_code` im Beispiel auf der Folie dient zum Kompilieren eines benannten PL/SQL-Packages. Der Code unterdrückt die Warnungen der Kategorie PERFORMANCE. Die Warnungseinstellungen der aufrufenden Umgebung müssen nach der Kompilierung wiederhergestellt werden. Der Code kennt die Warnungseinstellungen der aufrufenden Umgebung nicht. Er verwendet die Funktion `GET_WARNING_SETTING_STRING`, um die aktuelle Einstellung zu speichern. Mithilfe dieses Wertes in der letzten Zeile des Beispielcodes wird die Einstellung für die aufrufende Umgebung mit der Prozedur `DBMS_WARNING.SET_WARNING_SETTING_STRING` wiederhergestellt. Vor der Kompilierung des Packages mit natürlichem dynamischem SQL ändert die Prozedur `compile_code` die aktuelle Sessionwarnstufe, indem sie Warnungen für die Kategorie PERFORMANCE deaktiviert. Außerdem gibt der Code die ursprünglichen, geänderten und wiederhergestellten Warnungseinstellungen aus.

DBMS_WARNING – Beispiel

```
EXECUTE DBMS_WARNING.SET_WARNING_SETTING_STRING(-  
  'ENABLE:ALL', 'SESSION');
```

```
anonymous block completed
```

```
@/home/oracle/labs/plpu/code_ex/code_ex_scripts/code_11_33_s.sql
```

```
PROCEDURE compile_code compiled
```

```
EXECUTE compile_code('EMP_PKG');
```

```
anonymous block completed  
Current warning settings: ENABLE:ALL  
Modified warning settings: ENABLE:INFORMATIONAL,DISABLE:PERFORMANCE,ENABLE:SEVERE  
Restored warning settings: ENABLE:ALL
```

ORACLE®

Copyright © 2013, Oracle und/oder verbundene Unternehmen. All rights reserved. Alle Rechte vorbehalten.

Im Beispiel auf dieser Folie wird das Beispiel von der vorhergehenden Folie getestet. Zuerst aktivieren Sie alle Compilerwarnungen. Als Nächstes führen Sie das Skript von der vorherigen Seite aus. Zuletzt rufen Sie die Prozedur `compile_code` auf und übergeben ihr den Namen des vorhandenen Packages `EMP_PKG` als Parameter.

Warnmeldung PLW 06009

- **Die PLW-Warnung zeigt an, dass der Handler OTHERS Ihrer untergeordneten PL/SQL-Routine beendet werden kann, ohne eine der folgenden Aktionen auszuführen:**
 - Eine Form von RAISE
 - Einen Aufruf der Standardprozedur RAISE_APPLICATION_ERROR
- **Für Programmierungszwecke wird empfohlen, Exceptions von OTHERS-Handlern immer nach oben zu übergeben.**



ORACLE®

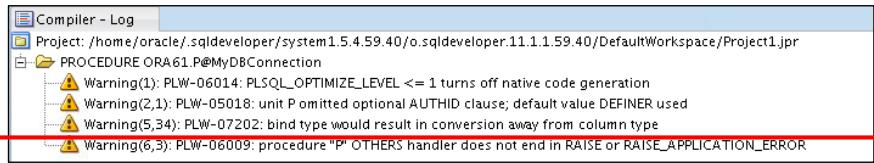
Copyright © 2013, Oracle und/oder verbundene Unternehmen. All rights reserved. Alle Rechte vorbehalten.

Für Programmierungszwecke wird empfohlen, die Exception über den Exception Handler OTHERS nach oben an die aufrufende untergeordnete Routine zu übergeben. Andernfalls riskieren Sie, dass Exceptions unbemerkt bleiben. Um diese Schwachstelle im Code zu umgehen, können Sie Warnungen für Ihre Session aktivieren und den zu prüfenden Code erneut kompilieren. Wird die Exception vom Handler OTHERS nicht behandelt, wird die Warnung PLW 06009 ausgegeben. Diese Warnung gilt ausschließlich für Oracle Database 11g und nachfolgende Releases.

Hinweis: PLW 06009 ist nicht die einzige neue Warnmeldung in Oracle Database 11g. Eine vollständige Liste aller PLW-Warnungen in Oracle Database 11g finden Sie im Dokument *Oracle Database Error Messages 11g Release 2 (11.2)*.

Warnung PLW 06009 – Beispiel

```
CREATE OR REPLACE PROCEDURE p(i IN VARCHAR2)
IS
BEGIN
    INSERT INTO t(col_a) VALUES (i);
EXCEPTION
    WHEN OTHERS THEN null;
END p;
/
ALTER PROCEDURE P COMPILE
PLSQL_warnings = 'enable:all' REUSE SETTINGS;
```



```
SELECT *
FROM user_errors
WHERE name = 'P'
```

	NAME	TYPE	SEQUENCE	LINE	POSITION	TEXT	ATTRIBUTE	MESSAGE_NUMBER
1	P	PROCEDURE	1	4	34	PLW-07202: bind type would result in conversion away from column type	WARNING	7202
2	P	PROCEDURE	2	1	1	PLW-05018: unit P omitted optional AUTHID clause; default value DEFINER used	WARNING	5018
3	P	PROCEDURE	3	0	0	PLW-06015: parameter PLSQL_DEBUG is deprecated; use PLSQL_OPTIMIZE_LEVEL = 1	WARNING	6015
4	P	PROCEDURE	4	0	0	PLW-06014: PLSQL_OPTIMIZE_LEVEL <= 1 turns off native code generation	WARNING	6014
5	P	PROCEDURE	5	5	3	PLW-06009: procedure "P" OTHERS handler does not end in RAISE or RAISE_APPLICATION_ERROR	WARNING	6009

ORACLE

Copyright © 2013, Oracle und/oder verbundene Unternehmen. All rights reserved. Alle Rechte vorbehalten.

Nachdem Sie das erste Codebeispiel auf der Folie ausgeführt und die Prozedur mit dem Object Navigator-Baum kompiliert haben, wird in der Registerkarte **Compiler – Log** die PLW-06009-Warnung angezeigt.

Sie können den Fehler auch in der Data Dictionary View `user_error` anzeigen.
Die Definition der im Beispiel auf der Folie verwendeten Tabelle `t` lautet:

```
CREATE TABLE t (col_a NUMBER);
```

Quiz

Die Kategorien für PL/SQL-Kompilierungszeit-Warnmeldungen lauten:

- a. SEVERE**
- b. PERFORMANCE**
- c. INFORMATIONAL**
- d. ALL**
- e. CRITICAL**



Copyright © 2013, Oracle und/oder verbundene Unternehmen. All rights reserved. Alle Rechte vorbehalten.

Richtige Antworten: a, b, c, d

PL/SQL-Warnmeldungen werden in Kategorien unterteilt, sodass Sie Gruppen ähnlicher Warnungen während der Kompilierung unterdrücken oder anzeigen können. Die Kategorien lauten:

- SEVERE: Meldungen über Bedingungen, die ein unerwartetes Verhalten oder falsche Ergebnisse zur Folge haben können. Beispiel: Probleme mit Parametern bei der Aliaserstellung
- PERFORMANCE: Meldungen über Bedingungen, die zu Performanceproblemen führen können. Beispiel: Übergabe eines Wertes vom Typ VARCHAR2 an die Spalte NUMBER in der Anweisung INSERT
- INFORMATIONAL: Meldungen über Bedingungen, die sich nicht auf die Performance oder Richtigkeit auswirken, die aber geändert werden können, um den Code pflegeleichter zu gestalten. Beispiel: Nicht erreichbarer Code, der nie ausgeführt werden kann
- ALL: Zeigt alle Kategorien an

Zusammenfassung

In dieser Lektion haben Sie Folgendes gelernt:

- Initialisierungsparameter des PL/SQL-Compilers verwenden
- PL/SQL-Kompilierungszeitwarnungen verwenden



Copyright © 2013, Oracle und/oder verbundene Unternehmen. All rights reserved. Alle Rechte vorbehalten.

Übungen zu Lektion 11 – Überblick: PL/SQL-Compiler

Diese Übungen behandeln folgende Themen:

- Compilerinitialisierungsparameter anzeigen
- Native Kompilierung für die Session aktivieren und Prozedur kompilieren
- Compilerwarnungen deaktivieren und dann die ursprünglichen Warnungseinstellungen der Session wiederherstellen
- Kategorien für einige Compiler-Warnmeldungsnummern identifizieren



Copyright © 2013, Oracle und/oder verbundene Unternehmen. All rights reserved. Alle Rechte vorbehalten.

In diesen Übungen zeigen Sie die Compilerinitialisierungsparameter an. Anschließend aktivieren Sie die native Kompilierung für die Session und kompilieren eine Prozedur. Danach unterdrücken Sie alle Compilerwarnungskategorien und stellen die ursprünglichen Warnungseinstellungen der Session wieder her. Zuletzt identifizieren Sie die Kategorien einiger Compiler-Warnmeldungsnummern.

Abhangigkeiten verwalten

12

ORACLE

Copyright © 2013, Oracle und/oder verbundene Unternehmen. All rights reserved. Alle Rechte vorbehalten.

Ziele

Nach Ablauf dieser Lektion haben Sie folgende Ziele erreicht:

- **Prozedurale Abhangigkeiten verfolgen**
- **Auswirkungen von Andlungen an Datenbankobjekten auf Prozeduren und Funktionen abschatzen**
- **Prozedurale Abhangigkeiten verwalten**

ORACLE

Copyright © 2013, Oracle und/oder verbundene Unternehmen. All rights reserved. Alle Rechte vorbehalten.

In dieser Lektion werden Abhangigkeiten zwischen Objekten sowie die implizite und explizite Rekompilierung ungultiger Objekte behandelt.

Abhängigkeiten von Schemaobjekten – Überblick

Objekttyp	Kann abhängig sein oder referenziert werden
Packagebody	Nur abhängig
Packagespezifikation	Beides
Sequence	Nur referenziert
Unterprogramm	Beides
Synonym	Beides
Tabelle	Beides
Trigger	Beides
Benutzerdefiniertes Objekt	Beides
Benutzerdefinierte Collection	Beides
View	Beides

ORACLE

Copyright © 2013, Oracle und/oder verbundene Unternehmen. All rights reserved. Alle Rechte vorbehalten.

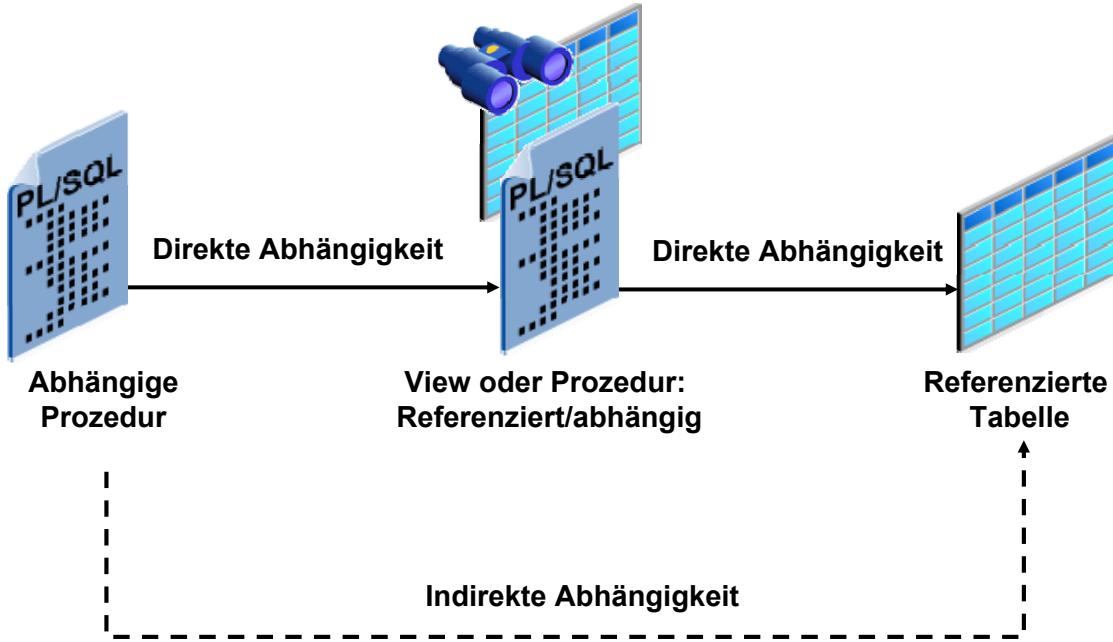
Abhängige und referenzierte Objekte

Bestimmte Typen von Schemaobjekten können in ihrer Definition andere Objekte referenzieren. Beispiel: Eine View ist definiert durch eine Abfrage, die Tabellen oder andere Views referenziert, und der Body eines Unterprogramms kann SQL-Anweisungen enthalten, die andere Objekte referenzieren. Verweist die Definition von Objekt A auf Objekt B, ist A (bezogen auf B) ein abhängiges Objekt und B (bezogen auf A) ein referenziertes Objekt.

Auswirkungen von Abhängigkeiten

- Wenn Sie die Definition eines referenzierten Objekts ändern, kann sich dies auf die Funktionsfähigkeit abhängiger Objekte auswirken. Beispiel: Bei Änderung der Tabellendefinition wird die Prozedur eventuell nicht mehr fehlerfrei ausgeführt.
- Der Oracle-Server zeichnet Abhängigkeiten zwischen Objekten automatisch auf. Für die Verwaltung von Abhängigkeiten erhalten alle Schemaobjekte einen Status (VALID bzw. INVALID), der im Data Dictionary gespeichert ist. Sie können den Status über die Data Dictionary View `USER_OBJECTS` anzeigen.
- Schemaobjekte mit dem Status VALID wurden bereits kompiliert und sind, wenn sie referenziert werden, sofort verfügbar.
- Schemaobjekte mit dem Status INVALID müssen dagegen kompiliert werden, bevor sie verwendet werden können.

Abhängigkeiten



ORACLE

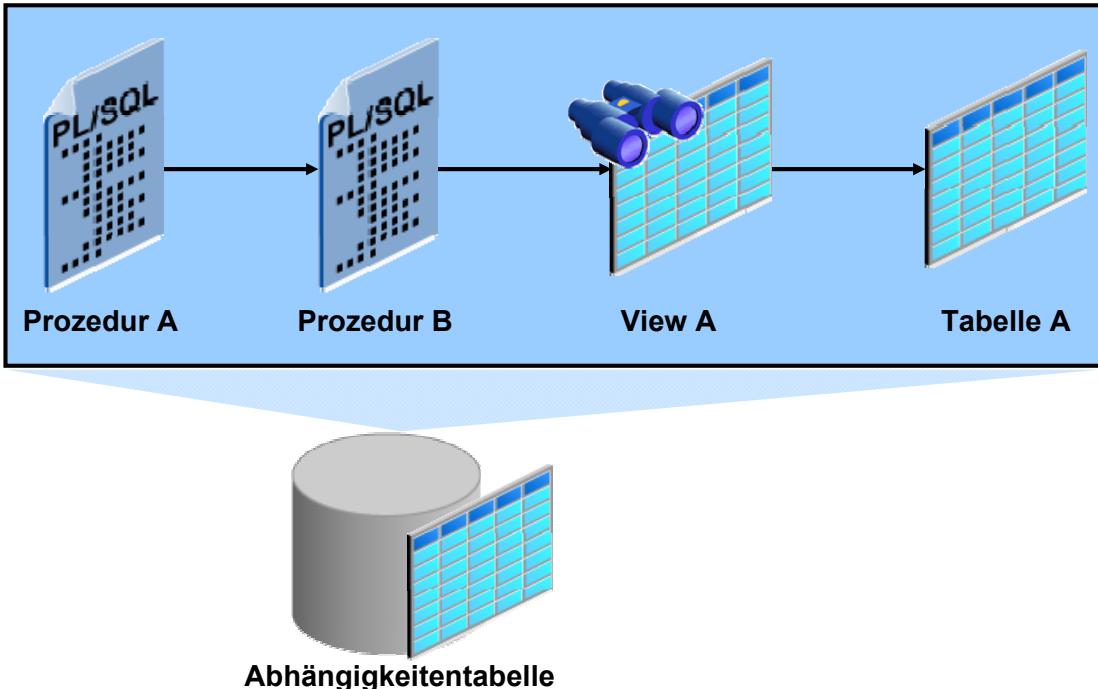
Copyright © 2013, Oracle und/oder verbundene Unternehmen. All rights reserved. Alle Rechte vorbehalten.

Abhängige und referenzierte Objekte

Prozeduren oder Funktionen können folgende Objekte direkt oder indirekt (über eine zwischengeschaltete View, Prozedur oder Funktion bzw. über eine in einem Package integrierte Prozedur oder Funktion) referenzieren:

- Tabellen
- Views
- Sequences
- Prozeduren
- Funktionen
- In einem Package integrierte Prozeduren oder Funktionen

Direkte lokale Abhangigkeiten



Copyright © 2013, Oracle und/oder verbundene Unternehmen. All rights reserved. Alle Rechte vorbehalten.

ORACLE

Lokale Abhangigkeiten verwalten

Bei lokalen Abhangigkeiten befinden sich die Objekte auf demselben Knoten in derselben Datenbank. Der Oracle-Server verwaltet alle lokalen Abhangigkeiten automatisch mithilfe der internen Abhangigkeitentabelle der Datenbank. Wird ein referenziertes Objekt geandert, werden die abhangigen Objekte gelegentlich invalidiert. Beim nachsten Aufruf eines invalidierten Objekts rekompiliert der Oracle-Server das Objekt automatisch.

Wenn Sie die Definition eines referenzierten Objekts ndern, kann sich dies – je nach Typ der nderung – auf die Funktionsfahigkeit abhangiger Objekte auswirken. Beispiel: Nachdem Sie eine Tabelle gelscht haben, sind alle Views, die auf dieser Tabelle basierten, unbrauchbar.

In Oracle Database 10g wurde der Befehl CREATE OR REPLACE SYNONYM erweitert, um die Invalidierung abhangiger PL/SQL-Programmeinheiten und referenzierender Views so gering wie mglich zu halten. Im weiteren Verlauf dieser Lektion wird darauf ausfhrlicher eingegangen.

Ab Oracle Database 11g werden Abhangigkeiten auf Elementebene innerhalb der Programm-einheit berwacht. Dies wird als fein granulierte Abhangigkeit bezeichnet und im weiteren Verlauf dieser Lektion genauer erlutert.

Direkte Objektabhängigkeiten abfragen – View USER_DEPENDENCIES

```
desc user_dependencies
Name          Null    Type
-----        ----   -----
NAME           NOT NULL VARCHAR2(128)
TYPE            VARCHAR2(18)
REFERENCED_OWNER  VARCHAR2(128)
REFERENCED_NAME   VARCHAR2(128)
REFERENCED_TYPE    VARCHAR2(18)
REFERENCED_LINK_NAME VARCHAR2(128)
SCHEMADID      NUMBER
DEPENDENCY_TYPE  VARCHAR2(4)
```

```
SELECT name, type, referenced_name, referenced_type
FROM   user_dependencies
WHERE  referenced_name IN ('EMPLOYEES', 'EMP_VW');
```

NAME	TYPE	REFERENCED_NAME	REFERENCED_TYPE
1 EMP_PKG	PACKAGE	EMPLOYEES	TABLE
2 EMP_PKG	PACKAGE BODY	EMPLOYEES	TABLE
3 EMPLOYEE_INITJOBS_TRG	TRIGGER	EMPLOYEES	TABLE
4 CHECK_SALARY_TRG	TRIGGER	EMPLOYEES	TABLE
5 EMP_DETAILS	VIEW	EMPLOYEES	TABLE
6 EMPLOYEE_REPORT	PROCEDURE	EMPLOYEES	TABLE

...

ORACLE

Copyright © 2013, Oracle und/oder verbundene Unternehmen. All rights reserved. Alle Rechte vorbehalten.

Um festzustellen, welche Datenbankobjekte manuell rekompiliert werden müssen, können Sie die direkten Abhängigkeiten über die Data Dictionary View USER_DEPENDENCIES anzeigen.

Die Views ALL_DEPENDENCIES und DBA_DEPENDENCIES enthalten die zusätzliche Spalte OWNER, die den Eigentümer des Objekts referenziert.

Data Dictionary View USER_DEPENDENCIES – Spalten

Die Data Dictionary View USER_DEPENDENCIES enthält folgende Spalten:

- NAME: Der Name des abhängigen Objekts
- TYPE: Der Typ des abhängigen Objekts (PROCEDURE, FUNCTION, PACKAGE, PACKAGE BODY, TRIGGER oder VIEW)
- REFERENCED_OWNER: Das Schema des referenzierten Objekts
- REFERENCED_NAME: Der Name des referenzierten Objekts
- REFERENCED_TYPE: Der Typ des referenzierten Objekts
- REFERENCED_LINK_NAME: Der Datenbanklink, über den auf das referenzierte Objekt zugegriffen wird

Objektstatus abfragen

Jedes Datenbankobjekt weist einen der folgenden Statuswerte auf:

Status	Beschreibung
VALID	Das Objekt wurde mithilfe der aktuellen Definition im Data Dictionary erfolgreich kompiliert.
COMPILED WITH ERRORS	Beim letzten Kompilierungsversuch für das Objekt sind Fehler aufgetreten.
INVALID	Das Objekt ist als ungültig markiert, weil ein referenziertes Objekt geändert wurde. (Nur abhängige Objekte können INVALID sein.)
UNAUTHORIZED	Eine Zugriffsberechtigung für ein referenziertes Objekt wurde entzogen. (Nur abhängige Objekte können UNAUTHORIZED sein.)

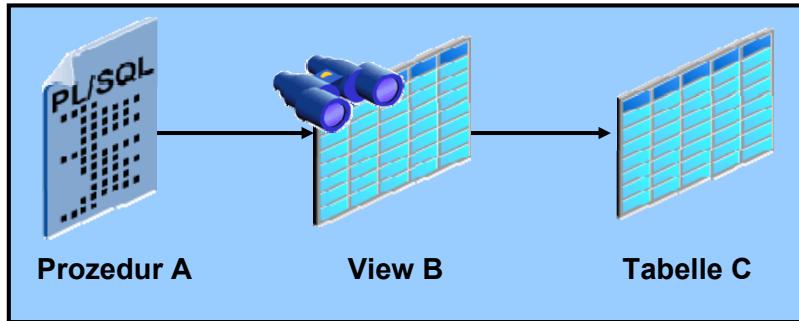
ORACLE®

Copyright © 2013, Oracle und/oder verbundene Unternehmen. All rights reserved. Alle Rechte vorbehalten.

Jedes Datenbankobjekt weist einen der in der Tabelle auf der Folie genannten Statuswerte auf.

Hinweis: Die statischen Data Dictionary Views USER_OBJECTS, ALL_OBJECTS und DBA_OBJECTS unterscheiden nicht zwischen Compiled with errors, Invalid und Unauthorized. Stattdessen wird in allen Fällen die Beschreibung INVALID angegeben.

Invalidierung abhängiger Objekte



- Prozedur A ist direkt abhängig von View B. View B ist direkt abhängig von Tabelle C. Prozedur A ist damit indirekt abhängig von Tabelle C.**
- Direkt abhängige Objekte werden nur durch Änderungen am referenzierten Objekt ungültig, von denen sie selbst betroffen sind.**
- Indirekt abhängige Objekte können auch durch Änderungen am Referenzobjekt ungültig werden, von denen sie selbst nicht betroffen sind.**

ORACLE

Copyright © 2013, Oracle und/oder verbundene Unternehmen. All rights reserved. Alle Rechte vorbehalten.

Wenn Objekt A von Objekt B abhängt und dieses wiederum von Objekt C abhängig ist, bedeutet dies, dass A direkt abhängig von B ist, B direkt abhängig von C und A indirekt abhängig von C.

Ab Oracle Database 11g werden direkt abhängige Objekte nur durch Änderungen am referenzierten Objekt ungültig, von denen sie selbst betroffen sind (d. h. Änderungen an der Signatur des referenzierten Objekts).

Indirekt abhängige Objekte können dagegen auch durch Änderungen am Referenzobjekt ungültig werden, von denen sie selbst nicht betroffen sind: Wird View B aufgrund einer Änderung an Tabelle C ungültig, invalidiert dies auch Prozedur A (und alle weiteren direkt oder indirekt von View B abhängigen Objekte). Dieser Vorgang wird als kaskadierende Invalidierung bezeichnet.

Angenommen, die Struktur der Tabelle, auf der eine View basiert, wird geändert. Wenn Sie die View mit dem SQL*Plus-Befehl `DESCRIBE` beschreiben, wird die Fehlermeldung `INVALID` für das Objekt angezeigt. Dies liegt daran, dass `DESCRIBE` kein SQL-Befehl und die View zu diesem Zeitpunkt ungültig ist, da die Struktur ihrer Basistabelle geändert wurde. Wenn Sie die View jetzt abfragen, wird sie automatisch rekompiliert, und Sie erhalten das gewünschte Ergebnis (sofern die Rekompilierung erfolgreich durchgeführt wird).

Schemaobjektänderungen, durch die einige Abhängigkeiten ungültig werden – Beispiel

```
CREATE VIEW commissioned AS
SELECT first_name, last_name, commission_pct FROM employees
WHERE commission_pct > 0.00;
```

```
CREATE VIEW six_figure_salary AS
SELECT * FROM employees
WHERE salary >= 100000;
```

```
SELECT object_name, status
FROM user_objects
WHERE object_type = 'VIEW';
```

OBJECT_NAME	STATUS
1 EMP_DETAILS_VIEW	VALID
2 EMP_DETAILS	VALID
3 COMMISSIONED	VALID
4 SIX FIGURE SALARY	VALID

ORACLE

Copyright © 2013, Oracle und/oder verbundene Unternehmen. All rights reserved. Alle Rechte vorbehalten.

Das Beispiel auf der Folie zeigt eine Schemaobjektänderung, durch die einige, aber nicht alle abhängigen Objekte ungültig werden. Die beiden neu erstellten Views basieren auf der Tabelle EMPLOYEES im Schema HR. Der Status der neu erstellten Views lautet VALID.

Schemaobjektänderungen, durch die einige Abhängigkeiten ungültig werden – Beispiel

```
ALTER TABLE employees MODIFY email VARCHAR2(50);

SELECT object_name, status
FROM user_objects
WHERE object_type = 'VIEW';
```

	OBJECT_NAME	STATUS
1	EMP_DETAILS_VIEW	VALID
2	EMP_DETAILS	VALID
3	COMMISSIONED	VALID
4	SIX FIGURE SALARY	INVALID

ORACLE

Copyright © 2013, Oracle und/oder verbundene Unternehmen. All rights reserved. Alle Rechte vorbehalten.

Angenommen, die Länge der Spalte EMAIL in der Tabelle EMPLOYEES soll von 25 auf 50 Zeichen erweitert werden, und Sie ändern die Tabelle wie auf der Folie oben gezeigt.

Da die Spalte EMAIL in der Liste SELECT der View COMMISSIONED nicht enthalten ist, wird diese View dadurch nicht ungültig. Die View SIXFIGURES wird dagegen durch die Änderung invalidiert, da alle Spalten aus der Tabelle gewählt sind.

Direkte und indirekte Abhangigkeiten anzeigen

- Fuhren Sie das Skript `utldtree.sql` aus, das die Objekte erstellt, mit denen Sie die direkten und indirekten Abhangigkeiten anzeigen konnen.**

```
@/home/oracle/labs/plpu/labs/utldtree.sql
```

- Fuhren Sie die Prozedur `DEPTREE_FILL` aus.**

```
EXECUTE deptree_fill('TABLE', 'ORA61', 'EMPLOYEES')
```

ORACLE

Copyright © 2013, Oracle und/oder verbundene Unternehmen. All rights reserved. Alle Rechte vorbehalten.

Direkte und indirekte Abhangigkeiten mit Oracle-Views anzeigen

Direkte und indirekte Abhangigkeiten zeigen Sie uber die zusatzlichen Benutzer-Views `DEPTREE` und `IDEPTREE` an, die von Oracle bereitgestellt werden.

Beispiel

- Vergewissern Sie sich, dass das Skript `utldtree.sql` ausgefuhrt wurde. Dieses Skript befindet sich im Ordner `$ORACLE_HOME/labs/plpu/labs`. Fuhren Sie das Skript wie folgt aus:

```
@?/labs/plpu/labs/utldtree.sql
```

Hinweis: In diesem Kurs ist das Skript im Ordner `labs` Ihrer Kursdateien abgelegt. Das oben gezeigte Codebeispiel verwendet den Teilnehmeraccount `ORA61`. (Das Beispiel bezieht sich auf eine Linux-Umgebung. Wird die Datei nicht gefunden, navigieren Sie in das Unterverzeichnis `labs`.)

- Fullen Sie die Tabelle `DEPTREE_TEMPTAB` mit Informationen fur ein bestimmtes referenziertes Objekt, indem Sie die Prozedur `DEPTREE_FILL` aufrufen. Es gibt drei Parameter fur diese Prozedur:

<code>object_type</code>	Typ des referenzierten Objekts
<code>object_owner</code>	Schema des referenzierten Objekts
<code>object_name</code>	Name des referenzierten Objekts

Abhängigkeiten anzeigen – View DEPTREE

```
SELECT    nested_level, type, name
FROM      deptree
ORDER BY  seq#;
```

NESTED_LEVEL	TYPE	NAME
1	0 TABLE	EMPLOYEES
2	1 VIEW	EMP_DETAILS_VIEW
3	1 TRIGGER	UPDATE_JOB_HISTORY
4	1 PROCEDURE	EMP_LIST
5	1 FUNCTION	GET_ANNUAL_COMP
6	1 PROCEDURE	ADD_EMPLOYEE
7	1 PACKAGE	EMP_PKG
8	2 PACKAGE BODY	EMP_PKG

...

ORACLE

Copyright © 2013, Oracle und/oder verbundene Unternehmen. All rights reserved. Alle Rechte vorbehalten.

Sie können eine tabellarische Darstellung aller abhängigen Objekte anzeigen, indem Sie die View DEPTREE abfragen. Dieselben Informationen lassen sich auch in einer eingerückten Darstellung anzeigen. Fragen Sie dazu wie folgt die View IDEPTREE ab, deren einzige Spalte DEPENDENCIES ist:

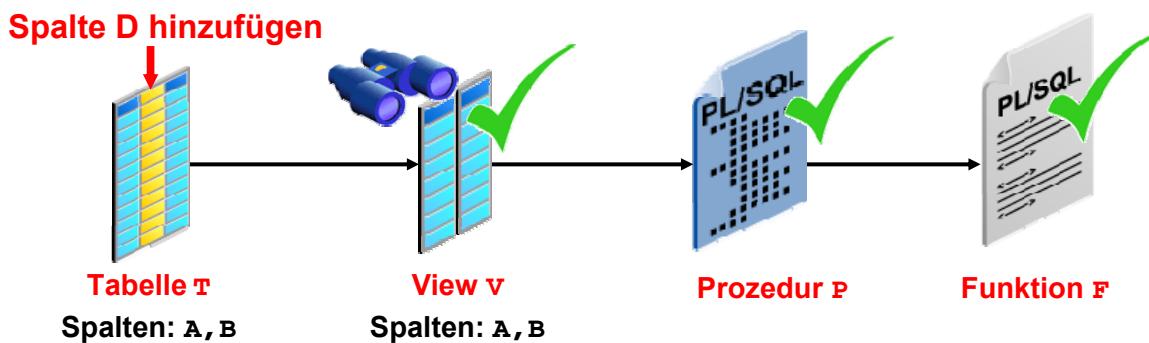
```
SELECT *
FROM   ideptree;
```

DEPENDENCIES
1 VIEW ORA61.EMP_DETAILS_VIEW
2 PACKAGE ORA61.EMP_PKG
3 VIEW ORA61.EMP_DETAILS
4 TRIGGER ORA61.DELETE_EMP_TRG
5 TRIGGER ORA61.UPDATE_JOB_HISTORY
6 TRIGGER ORA61.EMPLOYEE_INITJOBS_TRG
7 FUNCTION ORA61.GET_ANNUAL_COMP

...

Präzisere Abhängigkeitsmetadaten mit Oracle Database 11g

- Bei älteren Releases wurden abhängige Objekte durch Hinzufügen der Spalte **D** zur Tabelle **T** invalidiert.
- Oracle Database 11g zeichnet zusätzliche, feiner granulierte Informationen zur Abhängigkeitsverwaltung auf:
 - Durch Hinzufügen einer Spalte **D** zur Tabelle **T** wird die View **V** nicht beeinträchtigt. Außerdem werden die abhängigen Objekte nicht invalidiert.



ORACLE®

Copyright © 2013, Oracle und/oder verbundene Unternehmen. All rights reserved. Alle Rechte vorbehalten.

Fein granulierte Abhängigkeiten

Ab Oracle Database 11g haben Sie Zugriff auf Records, die präzisere Abhängigkeitsmetadaten beschreiben. Diese Funktionalität wird als fein granulierte Abhängigkeit bezeichnet. Damit können Sie anzeigen, ob abhängige Objekte ohne logische Anforderung invalidiert werden.

In früheren Releases der Oracle-Datenbank wurden Metadaten zu Abhängigkeiten für das gesamte Objekt aufgezeichnet. (Beispiel: PL/SQL-Einheit **P** hängt von PL/SQL-Einheit **F** ab, oder die View **V** hängt von der Tabelle **T** ab.) Das hatte zur Folge, dass abhängige Objekte gelegentlich ohne logischen Grund invalidiert wurden. Beispiel: View **V** ist nur von den Spalten **A** und **B** der Tabelle **T** abhängig. Wird die Tabelle **T** um eine Spalte **D** erweitert, ist die Gültigkeit der View **V** davon logisch nicht betroffen. Trotzdem wurde die View **V** in diesem Fall in Versionen vor Oracle Database Release 11.1 invalidiert. Ab Oracle Database Release 11.1 wird die View **V** durch Hinzufügen einer Spalte **D** zur Tabelle **T** dagegen nicht invalidiert. Analog dazu wird auch die Prozedur **P** nicht invalidiert, wenn sie nur von den Elementen **E1** und **E2** in einem Package abhängt und dem Package ein Element **E99** hinzugefügt wird.

Auf diese Weise werden weniger abhängige Objekte bei Änderungen an referenzierten Objekten für ungültig erklärt, was die Anwendungsverfügbarkeit in der Entwicklungsumgebung und bei Onlineupgrades der Anwendungen verbessert.

Fein granulierte Abhängigkeitsverwaltung

- **Ab Oracle Database 11g erfolgt die Überwachung von Abhängigkeiten auf *Elementebene innerhalb der Programmeinheit*.**
- **Die elementbasierte Abhängigkeitsverfolgung umfasst Folgendes:**
 - Abhängigkeit einer Single Table View von ihrer Basistabelle
 - Abhängigkeit einer PL/SQL-Programmeinheit (Packagespezifikation, Packagebody oder Unterprogramm) von:
 - anderen PL/SQL-Programmeinheiten
 - Tabellen
 - Views



ORACLE

Copyright © 2013, Oracle und/oder verbundene Unternehmen. All rights reserved. Alle Rechte vorbehalten.

Fein granulierte Abhängigkeitsverwaltung – 1. Beispiel

```
CREATE TABLE t2 (col_a NUMBER, col_b NUMBER, col_c NUMBER);
CREATE VIEW v AS SELECT col_a, col_b FROM t2;
```

```
SELECT ud.name, ud.type, ud.referenced_name,
       ud.referenced_type, uo.status
  FROM user_dependencies ud, user_objects uo
 WHERE ud.name = uo.object_name AND ud.name = 'V';
```

Results:

NAME	TYPE	REFERENCED_NAME	REFERENCED_TYPE	STATUS
1 V	VIEW	T2	TABLE	VALID

```
ALTER TABLE t2 ADD (col_d VARCHAR2(20));
```

```
SELECT ud.name, ud.type, ud.referenced_name,
       ud.referenced_type, uo.status
  FROM user_dependencies ud, user_objects uo
 WHERE ud.name = uo.object_name AND ud.name = 'V';
```

Results:

NAME	TYPE	REFERENCED_NAME	REFERENCED_TYPE	STATUS
1 V	VIEW	T2	TABLE	VALID

ORACLE

Copyright © 2013, Oracle und/oder verbundene Unternehmen. All rights reserved. Alle Rechte vorbehalten.

Abhängigkeit einer Single Table View von ihrer Basistabelle

Im ersten Beispiel auf der Folie wird die Tabelle T2 mit drei Spalten erstellt: COL_A, COL_B und COL_C. Eine View V wird auf Basis der Spalten COL_A und COL_B der Tabelle T2 erstellt. Die Abfrage der Dictionary Views zeigt, dass die View V von der Tabelle T abhängt und ihr Status VALID lautet.

Im dritten Beispiel wird die Tabelle T2 geändert. Eine neue Spalte COL_D wird hinzugefügt. Die Dictionary Views geben die View V nach wie vor als abhängig an, da durch die elementbasierte Abhängigkeitsverfolgung erkannt wird, dass die Spalten COL_A und COL_B nicht geändert wurden und die View deshalb nicht invalidiert werden muss.

Fein granulierte Abhängigkeitsverwaltung – 1. Beispiel

```
ALTER TABLE t2 MODIFY (col_a VARCHAR2(20));
SELECT ud.name, ud.referenced_name, ud.referenced_type,
       uo.status
  FROM user_dependencies ud, user_objects uo
 WHERE ud.name = uo.object_name AND ud.name = 'V';
```

Results:

	NAME	REFERENCED_NAME	REFERENCED_TYPE	STATUS
1	V	T2	TABLE	INVALID

ORACLE

Copyright © 2013, Oracle und/oder verbundene Unternehmen. All rights reserved. Alle Rechte vorbehalten.

Im Beispiel auf der Folie wird die View invalidiert, da ihr Element (COL_A) in der Tabelle geändert wurde, von der die View abhängig ist.

Fein granulierte Abhängigkeitsverwaltung – 2. Beispiel

```

CREATE OR REPLACE PACKAGE pkg IS
  PROCEDURE proc_1;
END pkg;
/
CREATE OR REPLACE PROCEDURE p IS
BEGIN
  pkg.proc_1();
END p;
/
CREATE OR REPLACE PACKAGE pkg
IS
  PROCEDURE proc_1;
  PROCEDURE unheard_of;
END pkg;
/

```

PACKAGE PKG compiled
 PROCEDURE P compiled
 PACKAGE PKG compiled

ORACLE

Copyright © 2013, Oracle und/oder verbundene Unternehmen. All rights reserved. Alle Rechte vorbehalten.

Im Beispiel auf der Folie erstellen Sie das Package PKG mit einer Deklaration der Prozedur PROC_1.

Die Prozedur P ruft PKG.PROC_1 auf.

Die Definition des Packages PKG wird geändert, und die Packagedeklaration wird um eine weitere Subroutine ergänzt.

Wenn Sie die Dictionary View USER_OBJECTS nach dem Status der Prozedur P abfragen, wird dieser, wie unten gezeigt, immer noch als VALID angezeigt, da das in die Definition von PKG aufgenommene Element durch die Prozedur P nicht referenziert wird.

```

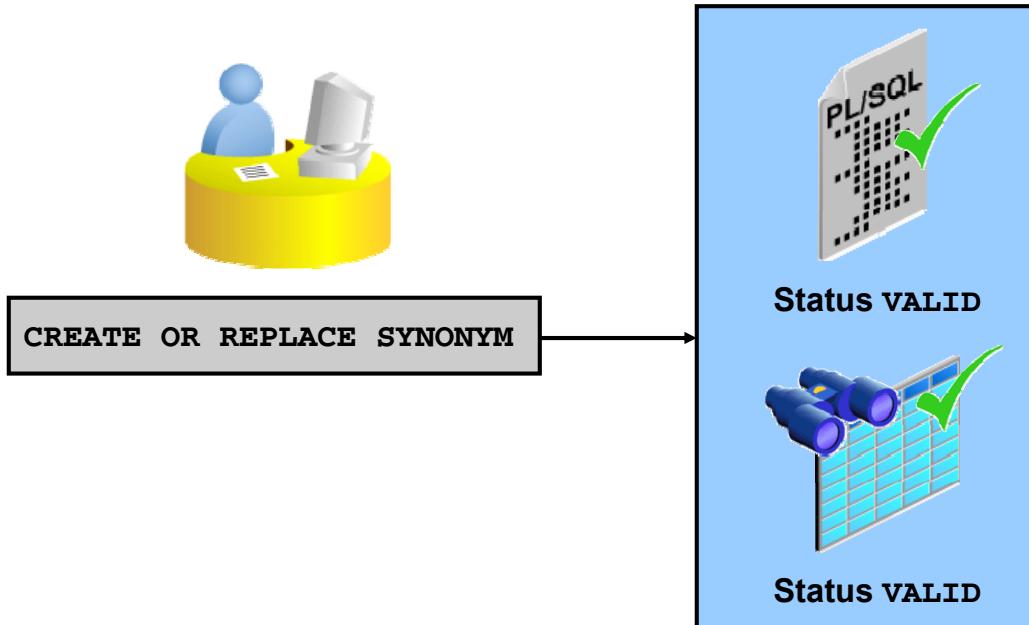
SELECT status FROM user_objects
WHERE object_name = 'P';

```

Results:

STATUS
1 VALID

Änderungen bei Synonymabhängigkeiten



ORACLE

Copyright © 2013, Oracle und/oder verbundene Unternehmen. All rights reserved. Alle Rechte vorbehalten.

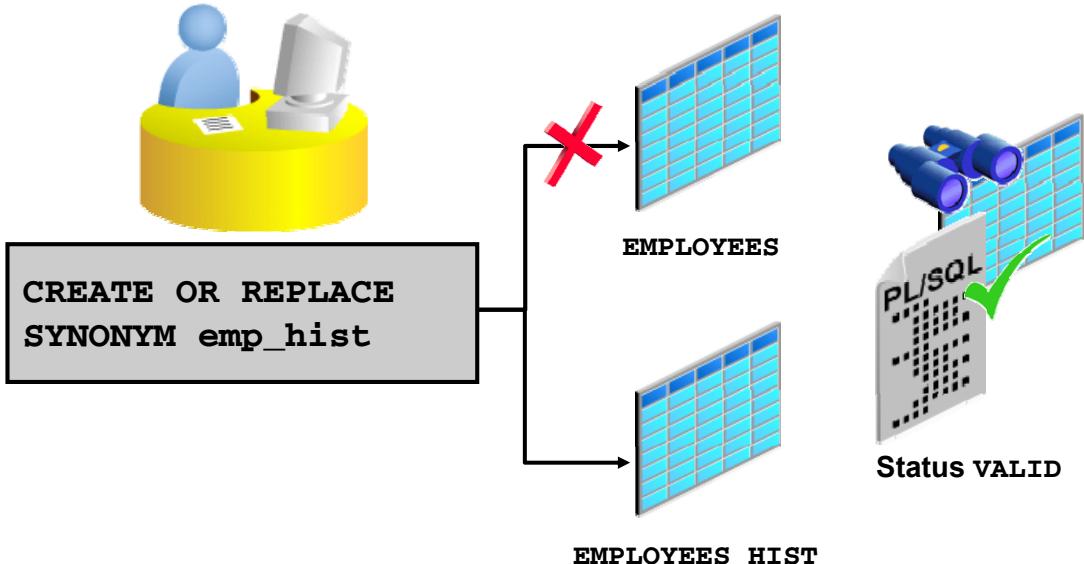
Mit Oracle Database können Sie Ausfallzeiten bei Codeupgrades oder Schemazusammenführungen minimieren.

Wenn bestimmte Bedingungen für Spalten, Berechtigungen, Partitionen usw. erfüllt sind, wird eine Tabelle oder ein Objekttyp als äquivalent betrachtet, und abhängige Objekte werden nicht mehr invalidiert.

In Oracle Database 10g wurde der Befehl `CREATE OR REPLACE SYNONYM` erweitert, um die Invalidierung abhängiger PL/SQL-Programmeinheiten und referenzierender Views so gering wie möglich zu halten. Damit entfallen zeitaufwändige Rekompilierungen der Programmeinheiten nach einer Neudefinition der Synonyme oder während der Ausführung der Einheiten. Um diese Funktionalität zu aktivieren, müssen Sie weder Parameter einstellen noch spezielle Befehle absetzen. Die Invalidierungen werden automatisch auf ein Minimum beschränkt.

Hinweis: Diese Weiterentwicklung gilt nur für Synonyme, die auf Tabellen verweisen.

Gültige PL/SQL-Programmeinheiten und -Views beibehalten



ORACLE

Copyright © 2013, Oracle und/oder verbundene Unternehmen. All rights reserved. Alle Rechte vorbehalten.

Gültige PL/SQL-Programmeinheiten beibehalten

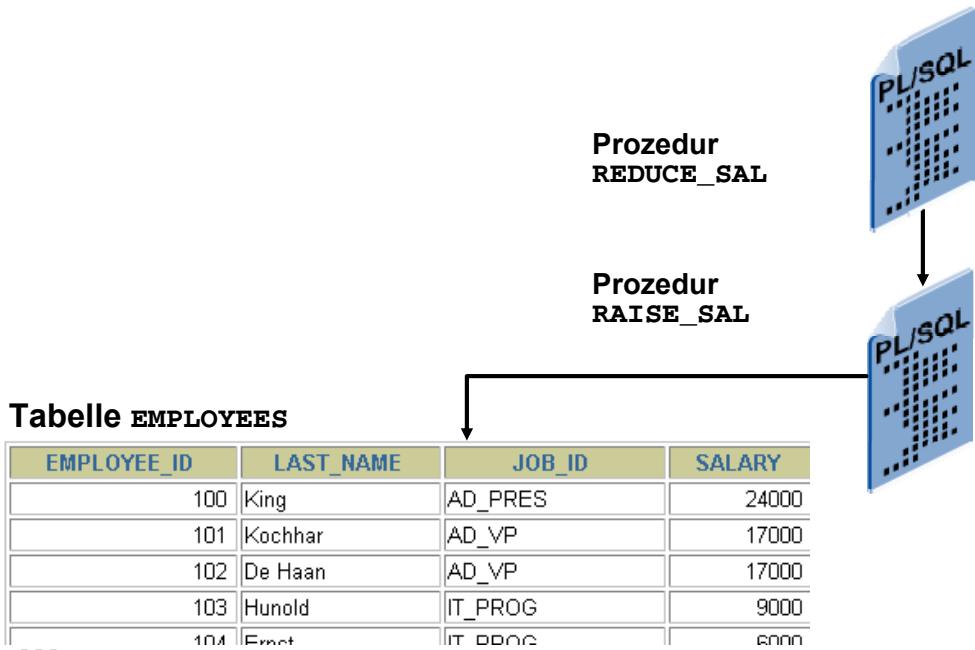
Seit Oracle Database 10g Release 2 können Sie die Definition von Synonymen unter folgenden Bedingungen ändern, ohne dass abhängige PL/SQL-Programmeinheiten ihre Gültigkeit verlieren:

- Die Reihenfolge sowie die Namen und Datentypen der Spalten in den Tabellen sind identisch.
- Die Berechtigungen der neu referenzierten Tabelle und ihrer Spalten sind eine Obermenge der Berechtigungsgruppe für die Originaltabelle. Diese Berechtigungen dürfen nicht ausschließlich durch Rollen abgeleitet werden.
- Die Namen und Typen von Partitionen und Unterpartitionen sind identisch.
- Die Tabellen sind auf gleiche Weise angeordnet.
- Objektspalten weisen denselben Typ auf.

Gültige Views beibehalten: Analog zu abhängigen PL/SQL-Programmeinheiten behalten auch abhängige Views ihre Gültigkeit, wenn Sie die Definition von Synonymen unter den oben genannten Bedingungen ändern. Anders als bei abhängigen PL/SQL-Programmeinheiten müssen bei der Neudeinition von Synonymen zusätzlich folgende Bedingungen eingehalten werden, damit abhängige Views den Status **VALID** beibehalten:

- Spalten und Spaltenreihenfolgen, die für Primärschlüsselindizes und eindeutige Indizes sowie Constraints vom Typ `NOT NULL`, `PRIMARY KEY` oder `UNIQUE` definiert wurden, müssen identisch sein.
- Die abhängige View darf keine referenziellen Constraints enthalten.

Lokale Abhängigkeiten – Weiteres Szenario



ORACLE

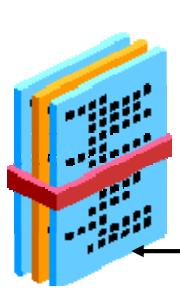
Copyright © 2013, Oracle und/oder verbundene Unternehmen. All rights reserved. Alle Rechte vorbehalten.

Das Beispiel auf der Folie zeigt die Auswirkungen einer Änderung in der Definition einer Prozedur auf die Rekompilierung einer abhängigen Prozedur. Angenommen, die Prozedur `RAISE_SAL` aktualisiert die Tabelle `EMPLOYEES` direkt, und die Prozedur `REDUCE_SAL` aktualisiert die Tabelle `EMPLOYEES` indirekt über `RAISE_SAL`.

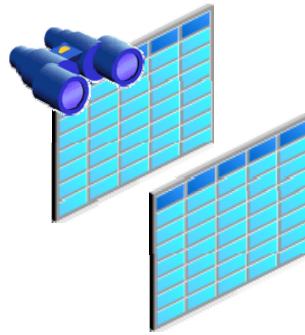
- Bei Änderung der internen Logik der Prozedur `RAISE_SAL` wird `REDUCE_SAL` erfolgreich rekompiliert, sofern `RAISE_SAL` erfolgreich kompiliert wurde.
- Werden die formalen Parameter für die Prozedur `RAISE_SAL` gelöscht, wird `REDUCE_SAL` nicht erfolgreich rekompiliert.

Richtlinien zur Reduzierung für ungültig erklärter Objekte

Anzahl der für ungültig erklärt abhängigen Objekte einschränken:



**Neue Elemente am
Packageende hinzufügen**



**Jede Tabelle über eine
View referenzieren**

ORACLE

Copyright © 2013, Oracle und/oder verbundene Unternehmen. All rights reserved. Alle Rechte vorbehalten.

Neue Elemente am Packageende hinzufügen

Fügen Sie neue Elemente stets am Ende des Packages hinzu. Auf diese Weise werden Slot- und Entry Point-Nummern der vorhandenen übergeordneten Packagelemente beibehalten und diese Elemente nicht für ungültig erklärt. Beispiel:

```
CREATE OR REPLACE PACKAGE pkg1 IS
  FUNCTION get_var RETURN VARCHAR2;
  PROCEDURE set_var (v VARCHAR2);
END;
```

Wenn Sie am Ende des Packages `pkg1` ein Element einfügen, hat dies keine Auswirkung auf die Gültigkeit von abhängigen Objekten, die `get_var` referenzieren. Fügen Sie das Element dagegen zwischen der Funktion `get_var` und der Prozedur `set_var` ein, werden die abhängigen Objekte, die die Funktion `set_var` referenzieren, dadurch invalidiert.

Jede Tabelle durch eine View referenzieren

Referenzieren Sie Tabellen mithilfe von Views indirekt. Auf diese Weise können Sie:

- der Tabelle Spalten hinzufügen, ohne dass dadurch abhängige Views oder PL/SQL-Objekte ungültig werden
- Spalten, die nicht von der View referenziert werden, ändern oder löschen, ohne dadurch die Gültigkeit abhängiger Objekte zu beeinträchtigen

Objekte neu validieren

- Ein referenziertes Objekt, das ungültig ist, muss validiert werden, ehe es verwendet werden kann.
- Die Validierung erfolgt automatisch bei Referenzierung eines Objekts; ein Benutzereingriff ist nicht erforderlich.
- Ungültige Objekte können den Status **COMPILED WITH ERRORS**, **UNAUTHORIZED** oder **INVALID** aufweisen.

ORACLE®

Copyright © 2013, Oracle und/oder verbundene Unternehmen. All rights reserved. Alle Rechte vorbehalten.

Der Compiler kann Objekte, die mit Fehlern kompiliert wurden, nicht automatisch neu validieren. Stattdessen rekompiliert der Compiler das Objekt. Treten dabei keine Fehler auf, wird das Objekt neu validiert. Andernfalls bleibt es ungültig.

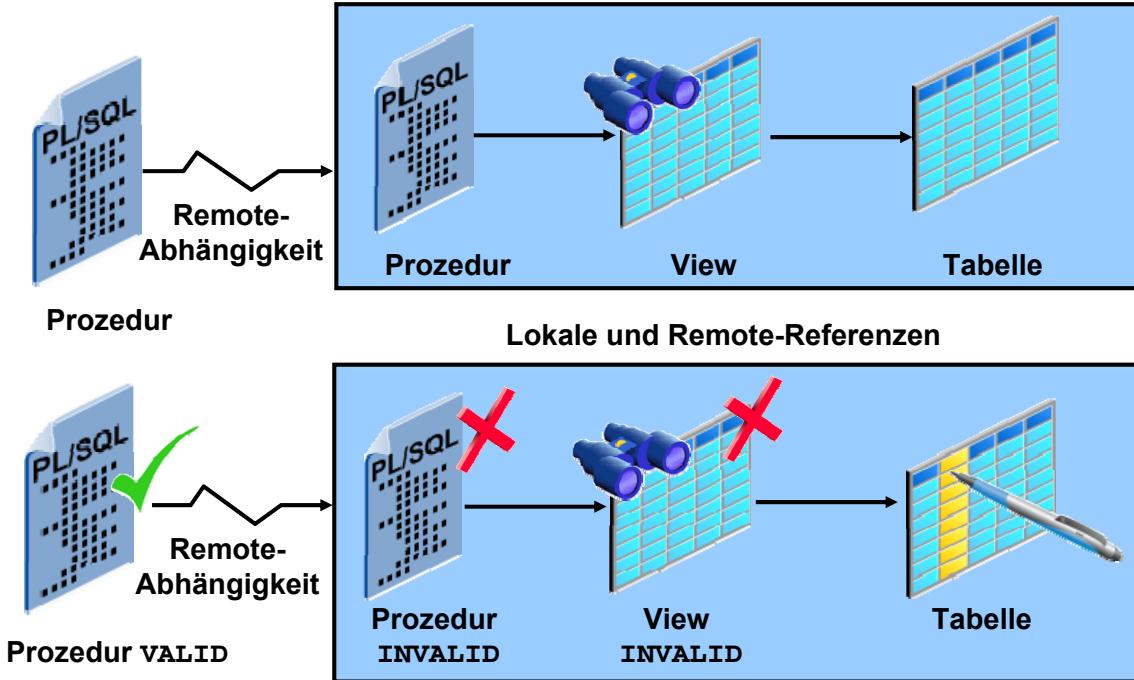
Der Compiler prüft, ob das Objekt mit dem Status UNAUTHORIZED über Zugriffsberechtigungen auf alle von ihm referenzierten Objekte verfügt. Ist dies der Fall, validiert der Compiler das Objekt ohne Rekompilierung neu. Andernfalls gibt der Compiler entsprechende Fehlermeldungen aus.

Der SQL-Compiler rekompiliert das ungültige Objekt. Treten dabei keine Fehler auf, wird das Objekt neu validiert. Andernfalls bleibt es ungültig.

Bei einer ungültigen PL/SQL-Programmeinheit (Prozedur, Funktion oder Package) prüft der PL/SQL-Compiler, ob an einem referenzierten Objekt Änderungen vorgenommen wurden, die sich auf das ungültige Objekt auswirken.

- Ist dies der Fall, rekompiliert der Compiler das ungültige Objekt. Treten dabei keine Fehler auf, wird das Objekt neu validiert. Andernfalls bleibt es ungültig.
- Wurden keine Änderungen an einem referenzierten Objekt vorgenommen, die sich auf das ungültige Objekt auswirken, validiert der Compiler das Objekt ohne Rekompilierung neu. Objekte, die aufgrund einer kaskadierenden Invalidierung für ungültig erklärt wurden, werden in der Regel rasch neu validiert.

Remote-Abhängigkeiten



Copyright © 2013, Oracle und/oder verbundene Unternehmen. All rights reserved. Alle Rechte vorbehalten.

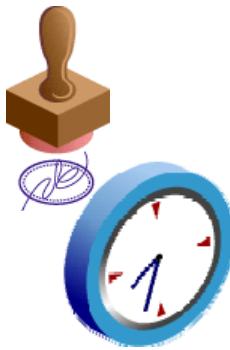
Bei Remote-Abhängigkeiten befinden sich die Objekte auf separaten Knoten. Der Oracle-Server verwaltet außer Abhängigkeiten zwischen lokalen und Remote-Prozeduren (einschließlich Funktionen, Packages und Trigger) keine Abhängigkeiten unter Objekten aus Remote-Schemas. Die lokale Stored Procedure und alle von ihr abhängigen Objekte werden invalidiert, jedoch beim ersten Aufruf nicht automatisch rekompiliert.

Rekompilierung abhängiger Objekte: Lokal und remote

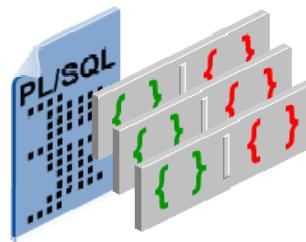
- Um zu ermitteln, ob die explizite Rekompilierung der abhängigen Remote-Prozeduren und die implizite Rekompilierung der abhängigen lokalen Prozeduren erfolgreich durchgeführt wurde, prüfen Sie den Status dieser Prozeduren in der View `USER_OBJECTS`.
- Wurde die automatische implizite Rekompilierung der abhängigen lokalen Prozeduren nicht erfolgreich durchgeführt, lautet der Status weiterhin `INVALID`, und der Oracle-Server gibt einen Laufzeitfehler aus. Um Betriebsunterbrechungen zu vermeiden, empfiehlt es sich daher dringend, abhängige lokale Objekte manuell zu rekompilieren, statt sich auf den automatischen Mechanismus zu verlassen.

Remote-Abhängigkeiten – Konzepte

Remote-Abhängigkeiten werden durch den vom Benutzer gewählten Modus gesteuert:



Zeitstempelprüfung



Signaturprüfung

ORACLE

Copyright © 2013, Oracle und/oder verbundene Unternehmen. All rights reserved. Alle Rechte vorbehalten.

Zeitstempelprüfung

Jede PL/SQL-Programmeinheit trägt einen Zeitstempel, der beim Erstellen oder Rekompilieren vergeben wird. Wenn Sie eine PL/SQL-Programmeinheit oder ein relevantes Schemaobjekt ändern, werden alle davon abhängigen Programmeinheiten als ungültig (INVALID) markiert und müssen vor ihrer Ausführung rekompiliert werden. Der eigentliche Zeitstempelvergleich erfolgt, wenn eine Anweisung im Body einer lokalen Prozedur eine Remote-Prozedur aufruft.

Signaturprüfung

Für jede PL/SQL-Programmeinheit werden Zeitstempel und Signatur gespeichert. Die Signatur eines PL/SQL-Konstrukts enthält folgende Informationen:

- Name des Konstrukts (Prozedur, Funktion oder Package)
- Basistypen der Parameter des Konstrukts
- Modi der Parameter (`IN`, `OUT` oder `IN OUT`)
- Anzahl der Parameter

Der aufgezeichnete Zeitstempel der aufrufenden Programmeinheit wird mit dem aktuellen Zeitstempel der aufgerufenen Remote-Programmeinheit verglichen. Stimmen die Zeitstempel überein, wird der Aufruf fortgesetzt. Andernfalls führt die RPC-(Remote Procedure Call-)Schicht einen einfachen Vergleich der Signatur durch, um festzustellen, ob der Aufruf die Sicherheit gefährdet. Wurde die Signatur nicht in unzulässiger Weise geändert, wird die Ausführung fortgesetzt. Andernfalls wird eine Fehlermeldung zurückgegeben.

Parameter **REMOTE_DEPENDENCIES_MODE** einstellen

- **Als Parameter in init.ora:**

```
REMOTE_DEPENDENCIES_MODE = TIMESTAMP |  
SIGNATURE
```

- **Auf Systemebene:**

```
ALTER SYSTEM SET REMOTE_DEPENDENCIES_MODE =  
TIMESTAMP | SIGNATURE
```

- **Auf Sessionebene:**

```
ALTER SESSION SET REMOTE_DEPENDENCIES_MODE =  
TIMESTAMP | SIGNATURE
```

ORACLE

Copyright © 2013, Oracle und/oder verbundene Unternehmen. All rights reserved. Alle Rechte vorbehalten.

Parameter **REMOTE_DEPENDENCIES_MODE**

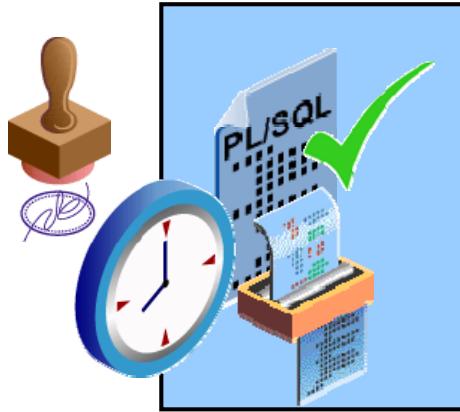
REMOTE_DEPENDENCIES_MODE einstellen

value TIMESTAMP
 SIGNATURE

Den Wert des Parameters **REMOTE_DEPENDENCIES_MODE** legen Sie mit einer der drei auf der Folie gezeigten Methoden fest.

Hinweis: Das Abhängigkeitsmodell richtet sich nach der aufrufenden Site.

Remote-Prozedur B wird um 8 Uhr kompiliert



Remote-Prozedur B:
Wird kompiliert und ist
um 8 Uhr gültig

ORACLE

Copyright © 2013, Oracle und/oder verbundene Unternehmen. All rights reserved. Alle Rechte vorbehalten.

Lokale Prozeduren, die Remote-Prozeduren referenzieren

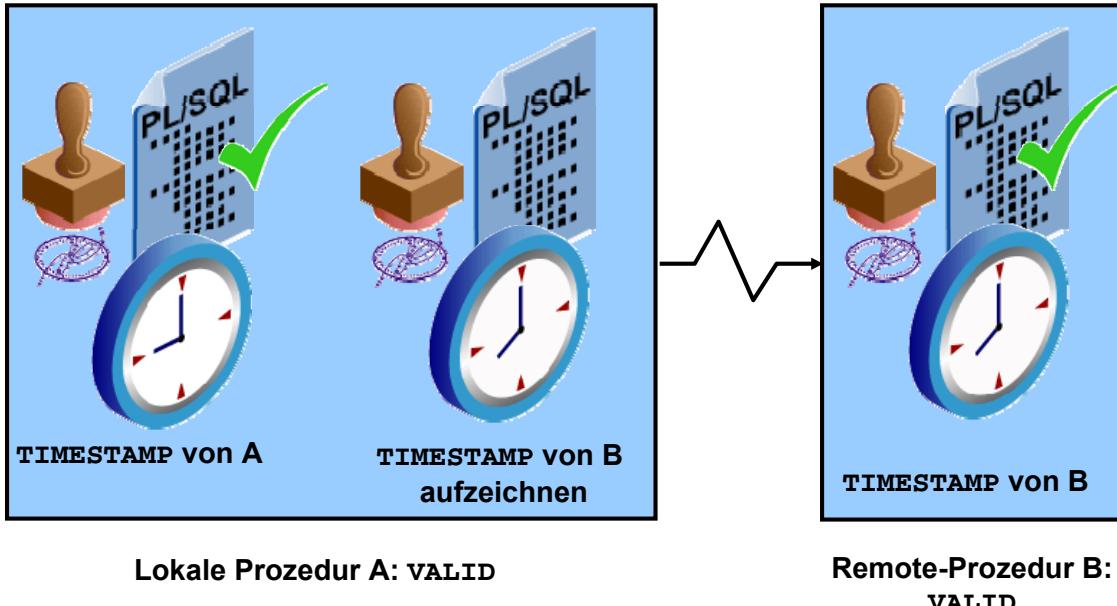
Lokale Prozeduren, die Remote-Prozeduren referenzieren, werden vom Oracle-Server invalidiert, wenn die Remote-Prozedur nach der Kompilierung der lokalen Prozedur rekompliert wird.

Automatischer Mechanismus für Remote-Abhängigkeiten

Bei der Kompilierung einer Prozedur zeichnet der Oracle-Server den Zeitstempel im Code `P` der Prozedur auf.

Bei der erfolgreichen Kompilierung der Remote-Prozedur B um 8 Uhr im Beispiel auf der Folie wird diese Uhrzeit als Zeitstempel gespeichert.

Lokale Prozedur A wird um 9 Uhr kompiliert



Copyright © 2013, Oracle und/oder verbundene Unternehmen. All rights reserved. Alle Rechte vorbehalten.

ORACLE

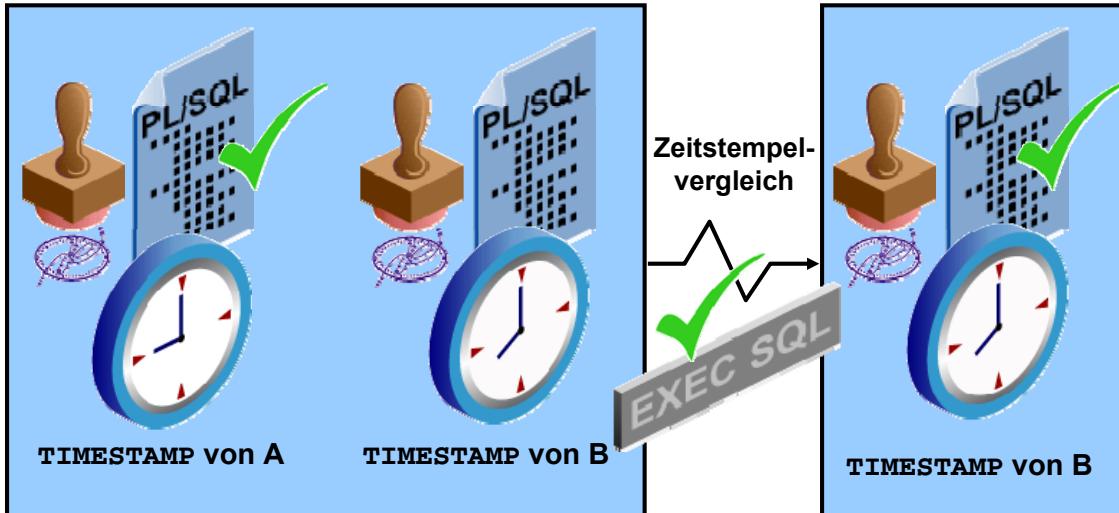
Lokale Prozeduren, die Remote-Prozeduren referenzieren

Automatischer Mechanismus für Remote-Abhängigkeiten (Fortsetzung)

Bei der Kompilierung einer lokalen Prozedur, die eine Remote-Prozedur referenziert, zeichnet der Oracle-Server auch den Zeitstempel der Remote-Prozedur im Code P der lokalen Prozedur auf.

Im Beispiel auf der Folie wird die lokale Prozedur A (die von der Remote-Prozedur B abhängt) um 9 Uhr erfolgreich kompiliert. Die Zeitstempel von Prozedur A und Remote-Prozedur B werden im Code P von Prozedur A gespeichert.

Prozedur A ausführen



Lokale Prozedur A: VALID

Remote-Prozedur B:
VALID

ORACLE

Copyright © 2013, Oracle und/oder verbundene Unternehmen. All rights reserved. Alle Rechte vorbehalten.

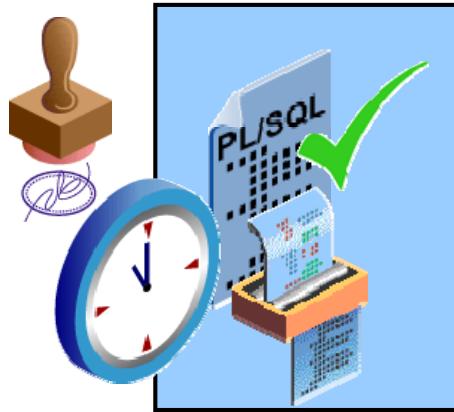
Automatischer Mechanismus für Remote-Abhängigkeiten

Wenn die lokale Prozedur zur Laufzeit aufgerufen wird, vergleicht der Oracle-Server die beiden Zeitstempel der referenzierten Remote-Prozedur.

Stimmen die Zeitstempel überein (was bedeutet, dass die Remote-Prozedur nicht rekompiliert wurde), führt der Oracle-Server die lokale Prozedur aus.

Im Beispiel auf der Folie ist der im Code P der Remote-Prozedur B aufgezeichnete Zeitstempel mit dem Zeitstempel identisch, der mit der lokalen Prozedur A gespeichert wurde. Die lokale Prozedur A ist damit gültig.

Remote-Prozedur B wird um 11 Uhr rekompiliert



**Remote-Prozedur B:
Wird rekompiliert und ist
um 11 Uhr gültig**

ORACLE

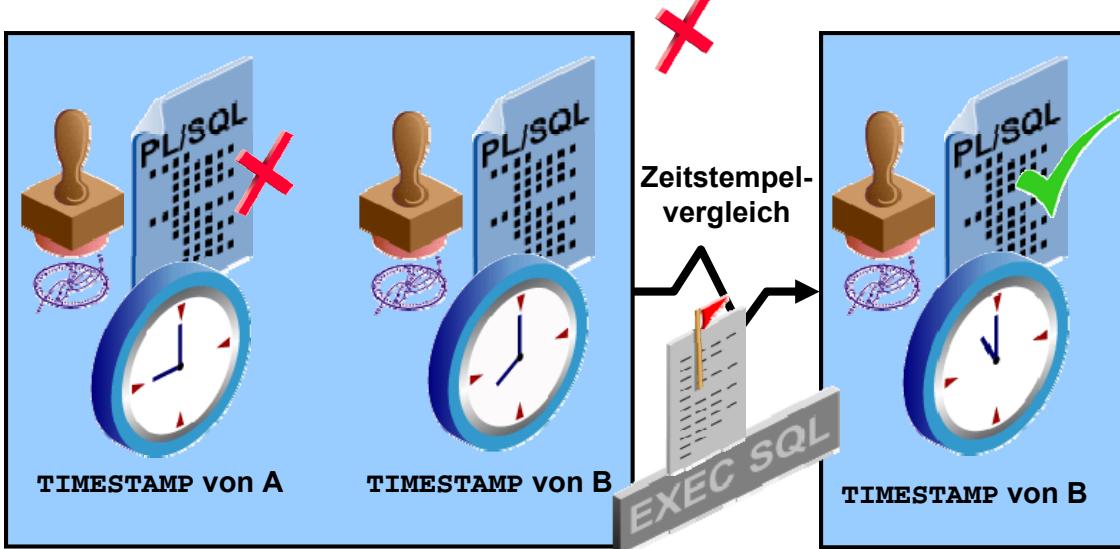
Copyright © 2013, Oracle und/oder verbundene Unternehmen. All rights reserved. Alle Rechte vorbehalten.

Lokale Prozeduren, die Remote-Prozeduren referenzieren

Die Remote-Prozedur B wird um 11 Uhr erfolgreich rekompiliert. Der neue Zeitstempel wird zusammen mit dem Code `P` der Prozedur gespeichert.

Prozedur A ausführen

Gespeicherter TIMESTAMP von B != COMPILE TIME von B



Copyright © 2013, Oracle und/oder verbundene Unternehmen. All rights reserved. Alle Rechte vorbehalten.

Automatischer Mechanismus für Remote-Abhängigkeiten

Stimmen die Zeitstempel nicht überein (was bedeutet, dass die Remote-Prozedur rekompiliert wurde), invalidiert der Oracle-Server die lokale Prozedur und gibt einen Laufzeitfehler zurück. Wird die (jetzt als ungültig gekennzeichnete) lokale Prozedur ein zweites Mal aufgerufen, rekompiliert der Oracle-Server sie vor der Ausführung gemäß dem automatischen Mechanismus für lokale Abhängigkeiten.

Hinweis: Gibt eine lokale Prozedur bei ihrem ersten Aufruf einen Laufzeitfehler zurück, der anzeigt, dass sich der Zeitstempel der Remote-Prozedur geändert hat, sollten Sie eine Strategie für den erneuten Aufruf der lokalen Prozedur entwickeln.

Im Beispiel auf der Folie wird die Remote-Prozedur um 11 Uhr rekompiliert. Diese Uhrzeit wird als Zeitstempel im Code `P` gespeichert. Im Code `P` der lokalen Prozedur A ist immer noch 8 Uhr als Zeitstempel für die Remote-Prozedur B eingestellt. Da der mit dem Code `P` der lokalen Prozedur A gespeicherte Zeitstempel vom Zeitstempel der Remote-Prozedur B abweicht, wird die lokale Prozedur als ungültig markiert. Wird die lokale Prozedur ein zweites Mal aufgerufen, kann sie erfolgreich kompiliert und als gültig (VALID) gekennzeichnet werden.

Ein Nachteil des Zeitstempelmodus besteht darin, dass er unnötig restriktiv ist. Häufig werden nicht unbedingt notwendige Rekomplizierungen abhängiger Objekte im Netzwerk durchgeführt, was die Performance beeinträchtigt.

Signaturmodus

- **Die Signatur einer Prozedur umfasst:**
 - Name der Prozedur
 - Datentypen der Parameter
 - Modi der Parameter
- **Die Signatur der Remote-Prozedur wird in der lokalen Prozedur gespeichert.**
- **Bei Ausführung einer abhängigen Prozedur wird die Signatur der referenzierten Remote-Prozedur verglichen.**

ORACLE

Copyright © 2013, Oracle und/oder verbundene Unternehmen. All rights reserved. Alle Rechte vorbehalten.

Signaturen

Mit dem Signaturmodell lassen sich einige der Probleme des nur auf Zeitstempeln basierenden Abhängigkeitenmodells umgehen. Bei diesem Modell kann die Remote-Prozedur rekompiliert werden, ohne dass sich dies auf die lokalen Prozeduren auswirkt. Dies ist bei verteilten Datenbanken wichtig.

Die Signatur eines Unterprogramms enthält folgende Informationen:

- Name des Unterprogramms
- Datentypen der Parameter
- Modi der Parameter
- Anzahl der Parameter
- Datentyp des Rückgabewertes einer Funktion

Wenn ein Remote-Programm geändert und rekompiliert wird, die Signatur dabei aber gleich bleibt, kann die lokale Prozedur die Remote-Prozedur ausführen. Im Zeitstempelmodus würde ein Fehler zurückgegeben, da die Zeitstempel nicht übereinstimmen.

PL/SQL-Programmeinheiten rekompilieren

Rekompilierung:

- **wird durch implizite Laufzeitrekompilierung automatisch durchgeführt**
- **wird durch explizite Rekompilierung mit der Anweisung ALTER durchgeführt**

```
ALTER PROCEDURE [SCHEMA.]procedure_name COMPILE;
```

```
ALTER FUNCTION [SCHEMA.]function_name COMPILE;
```

```
ALTER PACKAGE [SCHEMA.]package_name  
COMPILE [PACKAGE | SPECIFICATION | BODY];
```

```
ALTER TRIGGER trigger_name [COMPILE [DEBUG]] ;
```

ORACLE

Copyright © 2013, Oracle und/oder verbundene Unternehmen. All rights reserved. Alle Rechte vorbehalten.

PL/SQL-Objekte rekompilieren

Bei erfolgreicher Rekompilierung wird das Objekt als gültig (VALID) gekennzeichnet. Andernfalls gibt der Oracle-Server einen Fehler zurück, und das Objekt bleibt ungültig. Bei der Rekompilierung eines PL/SQL-Objekts rekompiliert der Oracle-Server zunächst alle ungültigen Objekte, von denen dieses Objekt abhängt.

Prozeduren: Alle lokalen Objekte, die von einer Prozedur abhängen (z. B. Prozeduren, die die rekompilierte Prozedur aufrufen, oder Packagebodys, die die Prozeduren zum Aufruf der rekompilierten Prozedur definieren), werden ebenfalls invalidiert.

Packages: Die Option COMPILE PACKAGE rekompiliert sowohl die Packagespezifikation als auch den Body, unabhängig davon, ob er ungültig ist. Die Option COMPILE SPECIFICATION rekompiliert die Packagespezifikation. Durch die Rekompilierung einer Packagespezifikation werden alle lokalen Objekte invalidiert, die von der Spezifikation abhängig sind, etwa Unterprogramme, die das Package verwenden. Auch der Body eines Packages hängt von der Packagespezifikation ab. Die Option COMPILE BODY rekompiliert nur den Packagebody.

Trigger: Explizites Rekompilieren macht die implizite Laufzeitrekompilierung unnötig und verhindert damit verbundene Kompilierungsfehler zur Laufzeit und Performanceoverhead.

Die Option DEBUG weist den PL/SQL-Compiler an, den vom PL/SQL-Debugger verwendeten Code zu generieren und zu speichern.

Nicht erfolgreiche Rekomplilierung

Die Rekomplilierung abhängiger Prozeduren und Funktionen ist in folgenden Fällen nicht erfolgreich:

- **Das referenzierte Objekt wurde gelöscht oder umbenannt.**
- **Der Datentyp der referenzierten Spalte wurde geändert.**
- **Die referenzierte Spalte wurde gelöscht.**
- **Eine referenzierte View wurde durch eine View mit anderen Spalten ersetzt.**
- **Die Parameterliste einer referenzierten Prozedur wurde geändert.**

ORACLE

Copyright © 2013, Oracle und/oder verbundene Unternehmen. All rights reserved. Alle Rechte vorbehalten.

Gelegentlich ist die Rekomplilierung abhängiger Prozeduren nicht erfolgreich (etwa weil eine referenzierte Tabelle gelöscht oder umbenannt wurde).

Der Erfolg der Rekomplilierung basiert auf der exakten Abhängigkeit. Wenn eine referenzierte View erneut erstellt wird, müssen alle Objekte rekompliiert werden, die von dieser View abhängig sind. Der Erfolg der Rekomplilierung hängt davon ab, welche Spalten die View jetzt enthält und welche Spalten die abhängigen Objekte für ihre Ausführung benötigen. Sind die erforderlichen Spalten nicht in der neuen View enthalten, bleibt das Objekt ungültig.

Erfolgreiche Rekomplilierung

Die Rekomplilierung abhängiger Prozeduren und Funktionen ist in folgenden Fällen erfolgreich:

- **Die referenzierte Tabelle enthält neue Spalten.**
- **Der Datentyp referenzierter Spalten ist gleich.**
- **Eine private Tabelle wurde gelöscht, doch es gibt eine öffentliche Tabelle mit demselben Namen und derselben Struktur.**
- **Der PL/SQL-Body einer referenzierten Prozedur wurde geändert und erfolgreich rekompiliert.**

ORACLE

Copyright © 2013, Oracle und/oder verbundene Unternehmen. All rights reserved. Alle Rechte vorbehalten.

Die Rekomplilierung abhängiger Objekte ist erfolgreich, wenn:

- einer referenzierten Tabelle neue Spalten hinzugefügt werden
- alle `INSERT`-Anweisungen eine Spaltenliste enthalten
- keine neue Spalte als `NOT NULL` definiert wird

Wird eine private Tabelle gelöscht, die von einer abhängigen Prozedur referenziert wird, ändert sich der Status der abhängigen Prozedur in `INVALID`. Ist bei der (expliziten oder impliziten) Rekomplilierung der Prozedur eine äquivalente öffentliche Tabelle vorhanden, kann die Prozedur erfolgreich rekompiliert werden. Sie ist dann jedoch von der öffentlichen Tabelle abhängig. Die Rekomplilierung ist nur erfolgreich, wenn die öffentliche Tabelle die von der Prozedur benötigten Spalten enthält. Andernfalls lautet der Status der Prozedur weiterhin `INVALID`.

Prozeduren rekompilieren

Abhängigkeitsfehler minimieren:

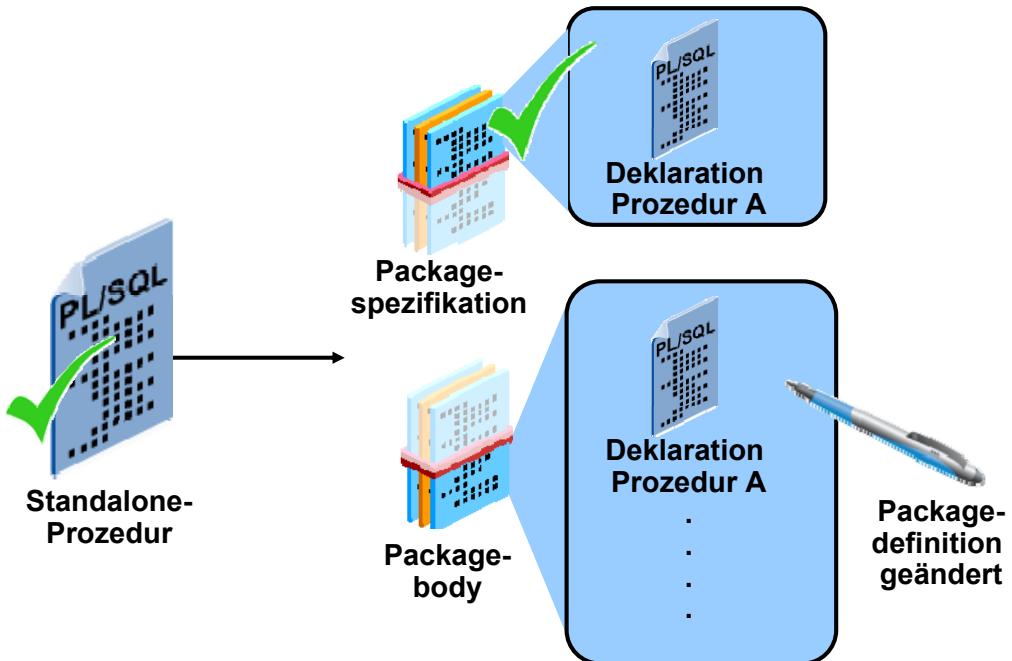
- **Records mit dem Attribut %ROWTYPE deklarieren**
- **Variablen mit dem Attribut %TYPE deklarieren**
- **Bei Abfragen die Notation SELECT * verwenden**
- **Eine Spaltenliste in Anweisungen vom Typ INSERT aufnehmen**

ORACLE

Copyright © 2013, Oracle und/oder verbundene Unternehmen. All rights reserved. Alle Rechte vorbehalten.

Die auf der Folie beschriebenen Richtlinien tragen dazu bei, Rekompilierungsfehler zu vermeiden.

Packages und Abhängigkeiten – Unterprogramm referenziert das Package



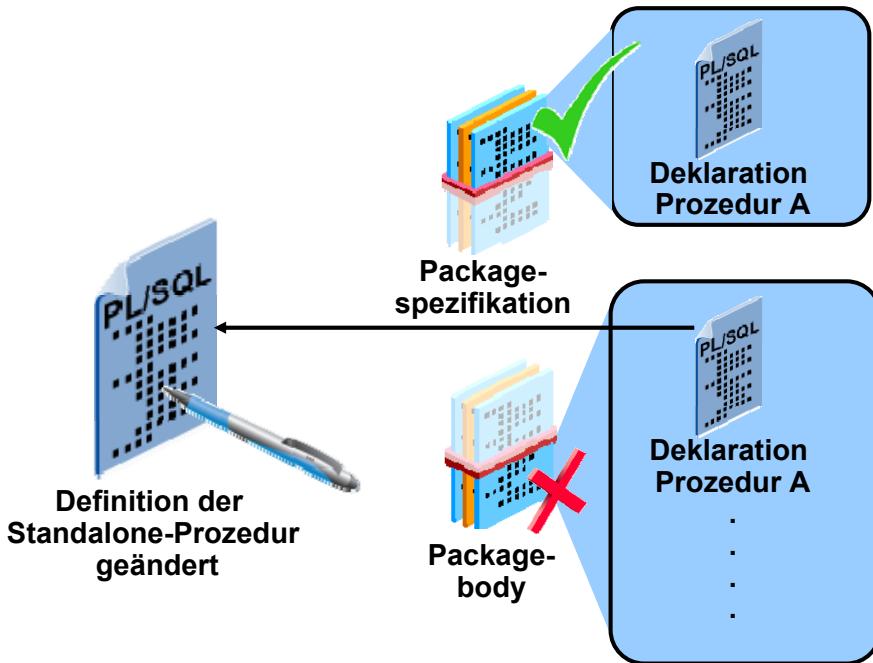
ORACLE

Copyright © 2013, Oracle und/oder verbundene Unternehmen. All rights reserved. Alle Rechte vorbehalten.

Sie können die Abhängigkeitsverwaltung bei Packages vereinfachen, wenn Sie eine Packageprozedur oder -funktion in einer Standalone-Prozedur oder -Funktion referenzieren.

- Wenn sich der Packagebody ändert und die Packagespezifikation unverändert bleibt, ist die Standalone-Prozedur, die ein Packagekonstrukt referenziert, weiterhin gültig.
- Bei Änderung der Packagespezifikation werden die externe Prozedur, die ein Packagekonstrukt referenziert, und auch der Packagebody invalidiert.

Packages und Abhängigkeiten – Packageunterprogramm referenziert Prozedur



ORACLE

Copyright © 2013, Oracle und/oder verbundene Unternehmen. All rights reserved. Alle Rechte vorbehalten.

Wenn sich eine im Package referenzierte Standalone-Prozedur ändert, wird der gesamte Packagebody invalidiert. Die Packagespezifikation bleibt jedoch gültig. Es empfiehlt sich daher, die Prozedur in das Package zu integrieren.

Quiz

Sie können direkte und indirekte Abhängigkeiten anzeigen, indem Sie das Skript `utldtree.sql` ausführen, die Tabelle `DEPTREE_TEMP TAB` mit Informationen für ein bestimmtes referenziertes Objekt füllen und die View `DEPTREE` oder `IDEPTREE` abfragen.

- a. Richtig**
- b. Falsch**

ORACLE

Copyright © 2013, Oracle und/oder verbundene Unternehmen. All rights reserved. Alle Rechte vorbehalten.

Richtige Antwort: a

Direkte und indirekte Abhängigkeiten anzeigen

Um direkte und indirekte Abhängigkeiten anzuzeigen, gehen Sie wie folgt vor:

1. Führen Sie das Skript `utldtree.sql` aus, um die Objekte zu erstellen, mit denen Sie die direkten und indirekten Abhängigkeiten anzeigen können.
2. Füllen Sie die Tabelle `DEPTREE_TEMP TAB` mit Informationen für ein bestimmtes referenziertes Objekt, indem Sie die Prozedur `DEPTREE_FILL` ausführen.
3. Fragen Sie die View `DEPTREE` oder `IDEPTREE` ab.

Zusammenfassung

In dieser Lektion haben Sie Folgendes gelernt:

- **Prozedurale Abhangigkeiten verfolgen**
- **Auswirkungen von nderungen an Datenbankobjekten auf Prozeduren und Funktionen abschatzen**
- **Prozedurale Abhangigkeiten verwalten**



Copyright © 2013, Oracle und/oder verbundene Unternehmen. All rights reserved. Alle Rechte vorbehalten.

Betriebsunterbrechungen lassen sich vermeiden, indem Sie abhangige Prozeduren berwachen und bei nderungen an der Definition eines Datenobjekts so bald wie mglich manuell rekompilieren.

Übungen zu Lektion 12 – Überblick: Abhängigkeiten im Schema verwalten

Diese Übungen behandeln folgende Themen:

- **Abhängigkeiten mithilfe von DEPTREE_FILL und IDEPTREE anzeigen**
- **Prozeduren, Funktionen und Packages rekompilieren**

ORACLE

Copyright © 2013, Oracle und/oder verbundene Unternehmen. All rights reserved. Alle Rechte vorbehalten.

In dieser Übung untersuchen Sie Abhängigkeiten in Ihrem Schema mithilfe der Prozedur DEPTREE_FILL und der View IDEPTREE. Außerdem rekompilieren Sie ungültige Prozeduren, Funktionen, Packages und Views.

Tabellenbeschreibungen

ORACLE

Copyright © 2013, Oracle und/oder verbundene Unternehmen. All rights reserved. Alle Rechte vorbehalten.

Schemabeschreibung

Gesamtbeschreibung

Das Beispielunternehmen aus den Beispielschemas von Oracle Database arbeitet weltweit und liefert viele unterschiedliche Produkte aus. Das Unternehmen hat drei Abteilungen:

- **Human Resources:** Verfolgt Informationen zu Mitarbeitern und Einrichtungen
- **Order Entry:** Verfolgt den Bestand und den Verkauf der Produkte über verschiedene Kanäle
- **Sales History:** Verfolgt Unternehmensstatistiken, um fundierte Geschäftsentscheidungen zu unterstützen

Jede dieser Abteilungen wird durch ein Schema dargestellt. In diesem Kurs haben Benutzer Zugriff auf die Objekte in sämtlichen Schemas. Der Schwerpunkt in den Beispielen, Demonstrationen und Übungen liegt jedoch auf dem Schema Human Resources (HR).

Alle für die Erstellung der Beispielschemas erforderlichen Skripte befinden sich im Ordner \$ORACLE_HOME/demo/schema/.

Human Resources (HR)

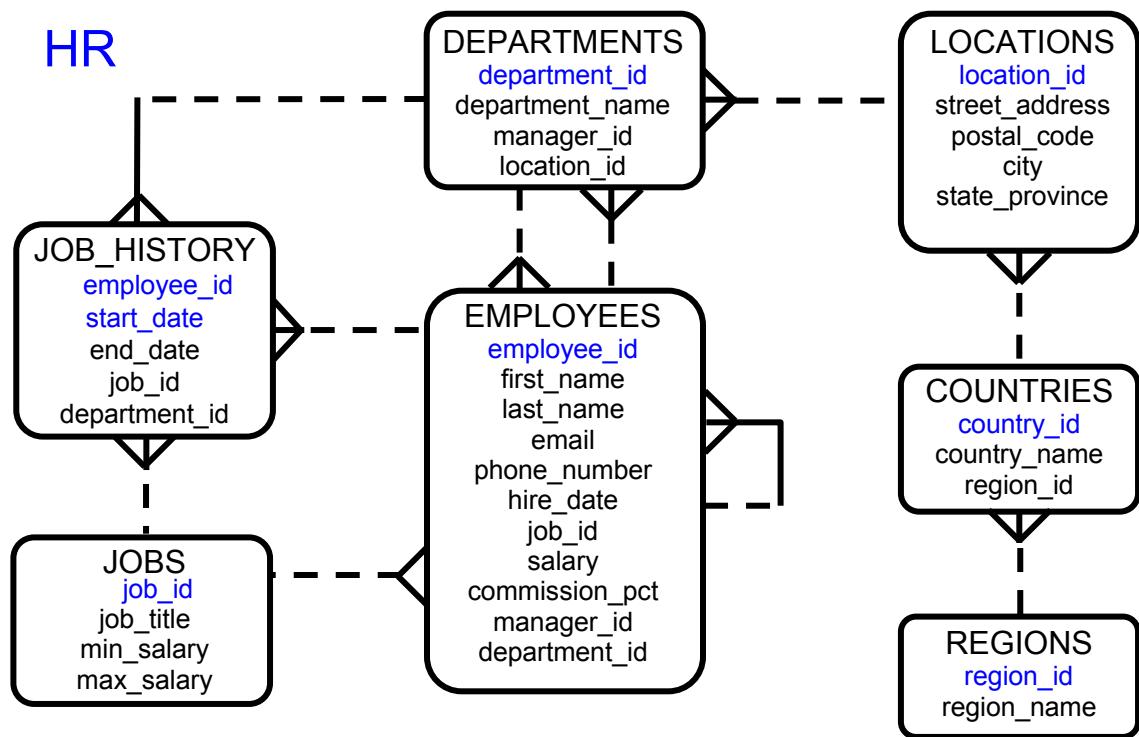
Dieses Schema wird im aktuellen Kurs verwendet. In den Datensätzen des Schemas Human Resources (HR) ist jeder Mitarbeiter mit ID, E-Mail-Adresse, Tätigkeits-ID, Gehalt und Manager verzeichnet. Neben ihrem Gehalt bekommen einige Mitarbeiter Provisionen.

Das Unternehmen zeichnet auch Informationen über Tätigkeiten innerhalb der Organisation auf. Jeder Tätigkeit ist eine ID und eine Tätigkeitsbezeichnung zugewiesen. Außerdem werden Angaben zum jeweiligen Mindest- und Höchstgehalt erfasst. Einige Mitarbeiter sind schon sehr lange im Unternehmen und hatten verschiedene Positionen inne. Wenn ein Mitarbeiter ausscheidet, werden Beschäftigungsdauer, Tätigkeits-ID und Abteilung aufgezeichnet.

Das Beispielunternehmen ist an verschiedenen Standorten tätig. Daher werden die Lager- und Abteilungsstandorte überwacht. Jeder Mitarbeiter ist einer Abteilung zugewiesen. Jede Abteilung wird entweder anhand einer eindeutigen Abteilungsnummer oder anhand einer Kurzbezeichnung identifiziert. Jede Abteilung ist einem Standort zugewiesen. Jeder Standort besitzt eine vollständige Adresse mit Straßenname, Postleitzahl, Ort, Bundesland, Bundesstaat, Provinz oder Kanton sowie Länderkennung.

Zu den Abteilungs- und Lagerstandorten zeichnet das Unternehmen Details wie den Namen des Landes, das Währungssymbol, den Namen der Währung sowie die Region auf, der das Land geografisch zuzuordnen ist.

HR – Entity Relationship-Diagramm



Human Resources (HR) – Tabellenbeschreibungen

```
DESCRIBE countries
```

Name	Null?	Type
COUNTRY_ID	NOT NULL	CHAR(2)
COUNTRY_NAME		VARCHAR2(40)
REGION_ID		NUMBER

```
SELECT * FROM countries;
```

#	COUNTRY_ID	COUNTRY_NAME	REGION_ID
1	AR	Argentina	2
2	AU	Australia	3
3	BE	Belgium	1
4	BR	Brazil	2
5	CA	Canada	2
6	CH	Switzerland	1
7	CN	China	3
8	DE	Germany	1
9	DK	Denmark	1
10	EG	Egypt	4
11	FR	France	1
12	HK	HongKong	3
13	IL	Israel	4
14	IN	India	3
15	IT	Italy	1
16	JP	Japan	3
17	KW	Kuwait	4
18	LB	Lebanon	4
19	MX	Mexico	2
20	NG	Nigeria	4
21	NL	Netherlands	1
22	SG	Singapore	3
23	UK	United Kingdom	1

24	US	United States of America	2
25	ZM	Zambia	4
26	ZW	Zimbabwe	4

DESCRIBE departments

Name	Null?	Type
DEPARTMENT_ID	NOT NULL	NUMBER(4)
DEPARTMENT_NAME	NOT NULL	VARCHAR2(30)
MANAGER_ID		NUMBER(6)
LOCATION_ID		NUMBER(4)

SELECT * FROM departments;

	DEPARTMENT_ID	DEPARTMENT_NAME	MANAGER_ID	LOCATION_ID
1	280	TRAINING	(null)	2500
2	290	Unknown	(null)	1700
3	300	ADVERTISING	(null)	1200
4	310	Unknown	(null)	1200
5	980	Education	(null)	2500
6	460	Training	(null)	2400
7	10	Administration	200	1700
8	20	Marketing	201	1800
9	30	Purchasing	114	1700
10	40	Human Resources	203	2400
11	50	Shipping	121	1500
12	60	IT	103	1400
13	70	Public Relations	204	2700
14	80	Sales	145	2500
15	90	Executive	100	1700
16	100	Finance	108	1700
17	110	Accounting	205	1700
18	120	Treasury	(null)	1700
19	130	Corporate Tax	(null)	1700
20	140	Control And Credit	(null)	1700
21	150	Shareholder Serv...	(null)	1700
22	160	Benefits	(null)	1700

23	170	Manufacturing	(null)	1700
24	180	Construction	(null)	1700
25	190	Contracting	(null)	1700
26	200	Operations	(null)	1700
27	210	IT Support	(null)	1700
28	220	NOC	(null)	1700
29	230	IT Helpdesk	(null)	1700
30	240	Government Sales	(null)	1700
31	250	Retail Sales	(null)	1700
32	260	Recruiting	(null)	1700
33	270	Payroll	(null)	1700

DESCRIBE employees

Name	Null?	Type
EMPLOYEE_ID	NOT NULL	NUMBER(6)
FIRST_NAME		VARCHAR2(20)
LAST_NAME	NOT NULL	VARCHAR2(25)
EMAIL	NOT NULL	VARCHAR2(25)
PHONE_NUMBER		VARCHAR2(20)
HIRE_DATE	NOT NULL	DATE
JOB_ID	NOT NULL	VARCHAR2(10)
SALARY		NUMBER(8,2)
COMMISSION_PCT		NUMBER(2,2)
MANAGER_ID		NUMBER(6)
DEPARTMENT_ID		NUMBER(4)

SELECT * FROM employees;

#	EMPLOYEE_ID	FIRST_NAME	LAST_NAME	EMAIL	PHONE_NUMBER	HIRE_DATE	JOB_ID	SALARY	COMMISSION_PCT	MANAGER_ID	DEPARTMENT_ID	TOTSAL
1	100	Steven	King	SKING	515.123.4567	17-JUN-03	AD_PRES	29040	(null)	(null)	90	(null)
2	101	Neena	Kochhar	NKOCHHAR	515.123.4568	21-SEP-05	AD_VP	17000	(null)	100	90	(null)
3	102	Lex	De Haan	LDEHAAN	515.123.4569	13-JAN-01	AD_VP	20570	(null)	100	90	(null)
4	103	Alexander	Hunold	AHUNOLD	590.423.4567	03-JAN-06	IT_PROG	9000	(null)	102	60	(null)
5	104	Bruce	Ernst	BERNST	590.423.4568	21-MAY-07	IT_PROG	7260	(null)	103	60	(null)
6	105	David	Austin	DAUSTIN	590.423.4569	25-JUN-05	IT_PROG	5000	(null)	103	60	(null)
7	106	Valli	Pataballa	VPATABAL	590.423.4560	05-FEB-06	IT_PROG	5000	(null)	103	60	(null)
8	107	Diana	Lorentz	DLORENTZ	590.423.5567	07-FEB-07	IT_PROG	5000	(null)	103	60	(null)
9	108	Nancy	Greenberg	NGREENBE	515.124.4569	17-AUG-02	FI_MGR	13208.8	(null)	101	100	(null)
10	109	Daniel	Faviet	DFAVIET	515.124.4169	16-AUG-02	FI_ACCOUNT	9000	(null)	108	100	(null)
11	110	John	Chen	JCHEN	515.124.4269	28-SEP-05	FI_ACCOUNT	9922	(null)	108	100	(null)
12	111	Ismael	Sciarra	ISCIARRA	515.124.4369	30-SEP-05	FI_ACCOUNT	7700	(null)	108	100	(null)
13	112	Jose Manuel	Urman	JMURMAN	515.124.4469	07-MAR-06	FI_ACCOUNT	7800	(null)	108	100	(null)
14	113	Luis	Popp	LPOPP	515.124.4567	07-DEC-07	FI_ACCOUNT	6900	(null)	108	100	(null)
15	114	Den	Raphaely	DRAPHEAL	515.127.4561	07-DEC-02	PU_MAN	12100	(null)	100	30	(null)
16	115	Alexander	Khoo	AKHOO	515.127.4562	18-MAY-03	PU_CLERK	2800	(null)	114	30	(null)
17	116	Shelli	Baida	SBAIDA	515.127.4563	24-DEC-05	PU_CLERK	3190	(null)	114	30	(null)
18	117	Sigal	Tobias	STOBIAS	515.127.4564	24-JUL-05	PU_CLERK	3080	(null)	114	30	(null)
19	118	Guy	Himuro	GHIMURO	515.127.4565	15-NOV-06	PU_CLERK	2860	(null)	114	30	(null)
20	119	Karen	Colmenares	KCOLMENA	515.127.4566	10-AUG-07	PU_CLERK	2750	(null)	114	30	(null)

#	EMPLOYEE_ID	FIRST_NAME	LAST_NAME	EMAIL	PHONE_NUMBER	HIRE_DATE	JOB_ID	SALARY	COMMISSION_PCT	MANAGER_ID	DEPARTMENT_ID	TOTSAL
21	120	Matthew	Weiss	MWEISS	650.123.1234	18-JUL-04	ST_MAN	8000	(null)	100	50	(null)
22	121	Adam	Fripp	AFRIPP	650.123.2234	10-APR-05	ST_MAN	8200	(null)	100	50	(null)
23	122	Payam	Kaufling	PKAUFLIN	650.123.3234	01-MAY-03	ST_MAN	7900	(null)	100	50	(null)
24	123	Shanta	Vollman	SVOLLMAN	650.123.4234	10-OCT-05	ST_MAN	6500	(null)	100	50	(null)
25	124	Kevin	Mourgos	KMOURGOS	650.123.5234	16-NOV-07	ST_MAN	5800	(null)	100	50	(null)
26	125	Juila	Nayer	JNAYER	650.124.1214	16-JUL-05	ST_CLERK	3200	(null)	120	50	(null)
27	126	Irene	Mikkilineni	IMIKKILILI	650.124.1224	28-SEP-06	ST_CLERK	2700	(null)	120	50	(null)
28	127	James	Landry	JLANDRY	650.124.1334	14-JAN-07	ST_CLERK	2400	(null)	120	50	(null)
29	128	Steven	Markle	SMARKLE	650.124.1434	08-MAR-08	ST_CLERK	2200	(null)	120	50	(null)
30	129	Laura	Bissot	LBISSTOT	650.124.5234	20-AUG-05	ST_CLERK	3300	(null)	121	50	(null)
31	130	Mozhe	Atkinson	MATKINSO	650.124.6234	30-OCT-05	ST_CLERK	2800	(null)	121	50	(null)
32	131	James	Marlow	JAMRLOW	650.124.7234	16-FEB-05	ST_CLERK	2500	(null)	121	50	(null)
33	132	TJ	Olson	TJOLSON	650.124.8234	10-APR-07	ST_CLERK	2100	(null)	121	50	(null)
34	133	Jason	Mallin	JMALLIN	650.127.1934	14-JUN-04	ST_CLERK	3300	(null)	122	50	(null)
35	134	Michael	Rogers	MRGERS	650.127.1834	26-AUG-06	ST_CLERK	2900	(null)	122	50	(null)
36	135	Ki	Gee	KGEE	650.127.1734	12-DEC-07	ST_CLERK	2400	(null)	122	50	(null)
37	136	Hazel	Philtanker	HPHILTAN	650.127.1634	06-FEB-08	ST_CLERK	2200	(null)	122	50	(null)
38	137	Renske	Ladwig	RLADWIG	650.121.1234	14-JUL-03	ST_CLERK	3600	(null)	123	50	(null)
39	138	Stephen	Stiles	SSTILES	650.121.2034	26-OCT-05	ST_CLERK	3400	(null)	123	50	(null)
40	139	John	Seo	JSEO	650.121.2019	12-FEB-06	ST_CLERK	2700	(null)	123	50	(null)

Employees (Fortsetzung)

#	EMPLOYEE_ID	FIRST_NAME	LAST_NAME	EMAIL	PHONE_NUMBER	HIRE_DATE	JOB_ID	SALARY	COMMISSION_PCT	MANAGER_ID	DEPARTMENT_ID	TOTSAL
41	140	Joshua	Patel	JPATEL	650.121.1834	06-APR-06	ST_CLERK	2500	(null)	123	50	(null)
42	141	Trenna	Rajs	TRAJS	650.121.8009	17-OCT-03	ST_CLERK	3500	(null)	124	50	(null)
43	142	Curtis	Davies	CDAVIES	650.121.2994	29-JAN-05	ST_CLERK	3100	(null)	124	50	(null)
44	143	Randall	Matos	RMATOS	650.121.2874	15-MAR-06	ST_CLERK	2600	(null)	124	50	(null)
45	144	Peter	Vargas	PVARGAS	650.121.2004	09-JUL-06	ST_CLERK	2500	(null)	124	50	(null)
46	145	John	Russell	JRUSSEL	011.44.1344.429268	01-OCT-04	SA_MAN	14000	0.4	100	80	(null)
47	146	Karen	Partners	KPARTNER	011.44.1344.467268	05-JAN-05	SA_MAN	13500	0.3	100	80	(null)
48	147	Alberto	Errazuriz	AERRAZUR	011.44.1344.429278	10-MAR-05	SA_MAN	12000	0.3	100	80	(null)
49	148	Gerald	Cambrault	GCAMBRAU	011.44.1344.619268	15-OCT-07	SA_MAN	11000	0.3	100	80	(null)
50	149	Eleni	Zlotkey	EZLOTKEY	011.44.1344.429018	29-JAN-08	SA_MAN	10500	0.2	100	80	(null)
51	150	Peter	Tucker	PTUCKER	011.44.1344.129268	30-JAN-05	SA REP	10000	0.3	145	80	(null)
52	151	David	Bernstein	DBERNSTE	011.44.1344.345268	24-MAR-05	SA REP	9500	0.25	145	80	(null)
53	152	Peter	Hall	PHALL	011.44.1344.478968	20-AUG-05	SA REP	9000	0.25	145	80	(null)
54	153	Christopher	Olsen	COLSEN	011.44.1344.498718	30-MAR-06	SA REP	8000	0.2	145	80	(null)
55	154	Nanette	Cambrault	NCAMBRAU	011.44.1344.987668	09-DEC-06	SA REP	7500	0.2	145	80	(null)
56	155	Oliver	Tuvault	OTUVVAULT	011.44.1344.486508	23-NOV-07	SA REP	7000	0.15	145	80	(null)
57	156	Janette	King	JKING	011.44.1345.429268	30-JAN-04	SA REP	10000	0.35	146	80	(null)
58	157	Patrick	Sully	PSULLY	011.44.1345.929268	04-MAR-04	SA REP	9500	0.35	146	80	(null)
59	158	Allan	McEwen	AMCEWEN	011.44.1345.829268	01-AUG-04	SA REP	9000	0.35	146	80	(null)
60	159	Lindsey	Smith	LSMITH	011.44.1345.729268	10-MAR-05	SA REP	8000	0.3	146	80	(null)

#	EMPLOYEE_ID	FIRST_NAME	LAST_NAME	EMAIL	PHONE_NUMBER	HIRE_DATE	JOB_ID	SALARY	COMMISSION_PCT	MANAGER_ID	DEPARTMENT_ID	TOTSAL
61	160	Louise	Doran	LDORAN	011.44.1345.629268	15-DEC-05	SA REP	7500	0.3	146	80	(null)
62	161	Sarah	Sewall	SSEWALL	011.44.1345.529268	03-NOV-06	SA REP	7000	0.25	146	80	(null)
63	162	Clara	Vishney	CVISHNEY	011.44.1346.129268	11-NOV-05	SA REP	10500	0.25	147	80	(null)
64	163	Danielle	Greene	DGREENE	011.44.1346.229268	19-MAR-07	SA REP	9500	0.15	147	80	(null)
65	164	Mattea	Marvins	MNARVINS	011.44.1346.329268	24-JAN-08	SA REP	7200	0.1	147	80	(null)
66	165	David	Lee	DLEE	011.44.1346.529268	23-FEB-08	SA REP	6800	0.1	147	80	(null)
67	166	Sundar	Ande	SANDE	011.44.1346.629268	24-MAR-08	SA REP	6400	0.1	147	80	(null)
68	167	Amit	Banda	ABANDA	011.44.1346.729268	21-APR-08	SA REP	6200	0.1	147	80	(null)
69	168	Lisa	Ozer	LOZER	011.44.1343.929268	11-MAR-05	SA REP	11500	0.25	148	80	(null)
70	169	Harrison	Bloom	HBLOOM	011.44.1343.829268	23-MAR-06	SA REP	10000	0.2	148	80	(null)
71	170	Taylor	Fox	TFOX	011.44.1343.729268	24-JAN-06	SA REP	9600	0.2	148	999	(null)
72	171	William	Smith	WSMITH	011.44.1343.629268	23-FEB-07	SA REP	7400	0.15	148	80	(null)
73	172	Elizabeth	Bates	EBATES	011.44.1343.529268	24-MAR-07	SA REP	7300	0.15	148	80	(null)
74	173	Sundita	Kumar	SKUMAR	011.44.1343.329268	21-APR-08	SA REP	6100	0.1	148	80	(null)
75	174	Ellen	Abel	EABEL	011.44.1644.429267	11-MAY-04	SA REP	11000	0.3	149	80	(null)
76	175	Alyssa	Hutton	AHUTTON	011.44.1644.429266	19-MAR-05	SA REP	8800	0.25	149	80	(null)
77	176	Jonathon	Taylor	JTAYLOR	011.44.1644.429265	24-MAR-06	SA REP	10406	0.2	149	80	(null)
78	177	Jack	Livingston	JLIVINGS	011.44.1644.429264	23-APR-06	SA REP	8400	0.2	149	80	(null)
79	179	Charles	Johnson	CJOHNSON	011.44.1644.429262	04-JAN-08	SA REP	6200	0.1	149	80	(null)
80	180	Winston	Taylor	WTAYLOR	650.507.9876	24-JAN-06	SH_CLERK	3200	(null)	120	50	(null)

Employees (Fortsetzung)

#	EMPLOYEE_ID	FIRST_NAME	LAST_NAME	EMAIL	PHONE_NUMBER	HIRE_DATE	JOB_ID	SALARY	COMMISSION_PCT	MANAGER_ID	DEPARTMENT_ID	TOTSAL
81	181	Jean	Fleur	JFLEUR	650.507.9877	23-FEB-06	SH_CLERK	3100	(null)	120	50	(null)
82	182	Martha	Sullivan	MSULLIVA	650.507.9878	21-JUN-07	SH_CLERK	2500	(null)	120	50	(null)
83	183	Girard	Geoni	GGEOINI	650.507.9879	03-FEB-08	SH_CLERK	2800	(null)	120	50	(null)
84	184	Nandita	Sarchand	NSARCHAN	650.509.1876	27-JAN-04	SH_CLERK	4200	(null)	121	50	(null)
85	185	Alexis	Bull	ABULL	650.509.2876	20-FEB-05	SH_CLERK	4100	(null)	121	50	(null)
86	186	Julia	Dellinger	JDELLING	650.509.3876	24-JUN-06	SH_CLERK	3400	(null)	121	50	(null)
87	187	Anthony	Cabrio	ACABRIO	650.509.4876	07-FEB-07	SH_CLERK	3000	(null)	121	50	(null)
88	188	Kelly	Chung	KCHUNG	650.505.1876	14-JUN-05	SH_CLERK	3800	(null)	122	50	(null)
89	189	Jennifer	Dilly	JDILLY	650.505.2876	13-AUG-05	SH_CLERK	3600	(null)	122	50	(null)
90	190	Timothy	Gates	TGATES	650.505.3876	11-JUL-06	SH_CLERK	2900	(null)	122	50	(null)
91	191	Randal	Perkins	RPERKINS	650.505.4876	19-DEC-07	SH_CLERK	2500	(null)	122	50	(null)
92	192	Sarah	Bell	SBELL	650.501.1876	04-FEB-04	SH_CLERK	4000	(null)	123	50	(null)
93	193	Britney	Everett	BEVERETT	650.501.2876	03-MAR-05	SH_CLERK	3900	(null)	123	50	(null)
94	194	Samuel	McCain	SMCCAIN	650.501.3876	01-JUL-06	SH_CLERK	3200	(null)	123	50	(null)
95	195	Vance	Jones	VJONES	650.501.4876	17-MAR-07	SH_CLERK	2800	(null)	123	50	(null)
96	196	Alana	Walsh	AWALSH	650.507.9811	24-APR-06	SH_CLERK	3100	(null)	124	50	(null)
97	197	Kevin	Feeley	KFEENEY	650.507.9822	23-MAY-06	SH_CLERK	3000	(null)	124	50	(null)
98	198	Donald	O'Connell	DOCONNEL	650.507.9833	21-JUN-07	SH_CLERK	2600	(null)	124	50	(null)
99	199	Douglas	Grant	DGRANT	650.507.9844	13-JAN-08	SH_CLERK	2600	(null)	124	50	(null)
100	200	Jennifer	Whalen	JWHALEN	515.123.4444	17-SEP-03	AD_ASST	4400	(null)	101	10	(null)

#	EMPLOYEE_ID	FIRST_NAME	LAST_NAME	EMAIL	PHONE_NUMBER	HIRE_DATE	JOB_ID	SALARY	COMMISSION_PCT	MANAGER_ID	DEPARTMENT_ID	TOTSAL
101	201	Michael	Hartstein	MHARTSTE	515.123.5555	17-FEB-04	MK_MAN	13000	(null)	100	20	(null)
102	202	Pat	Fay	PFAY	603.123.6666	17-AUG-05	MK_REP	6000	(null)	201	20	(null)
103	203	Susan	Mavris	SWAVRIS	515.123.7777	07-JUN-02	HR_REP	6500	(null)	101	40	(null)
104	204	Hermann	Baer	HBAER	515.123.8888	07-JUN-02	PR_REP	10000	(null)	101	70	(null)
105	205	Shelley	Higgins	SHIGGINS	515.123.8080	07-JUN-02	AC_MGR	12008	(null)	101	110	(null)
106	206	William	Gietz	WGIEITZ	515.123.8181	07-JUN-02	AC_ACCOUNT	8300	(null)	205	110	(null)
107	207	Joe	Harris	JAHARRIS	(null)	16-NOV-12	SA_REP	1000	0	145	80	(null)
108	208	David	Smith	DASHWITH	(null)	18-NOV-12	SA_REP	1000	0	145	80	(null)
109	209	Samuel	Joplin	SJOPLIN	(null)	18-NOV-12	SA_REP	1100	0	145	30	(null)

```
DESCRIBE job_history
```

Name	Null?	Type
EMPLOYEE_ID	NOT NULL	NUMBER(6)
START_DATE	NOT NULL	DATE
END_DATE	NOT NULL	DATE
JOB_ID	NOT NULL	VARCHAR2(10)
DEPARTMENT_ID		NUMBER(4)

```
SELECT * FROM job_history;
```

#	EMPLOYEE_ID	START_DATE	END_DATE	JOB_ID	DEPARTMENT_ID
1	102	13-JAN-01	24-JUL-06	IT_PROG	60
2	101	21-SEP-97	27-OCT-01	AC_ACCOUNT	110
3	101	28-OCT-01	15-MAR-05	AC_MGR	110
4	201	17-FEB-04	19-DEC-07	MK_REP	20
5	114	24-MAR-06	31-DEC-07	ST_CLERK	50
6	122	01-JAN-07	31-DEC-07	ST_CLERK	50
7	200	17-SEP-95	17-JUN-01	AD_ASST	90
8	176	24-MAR-06	31-DEC-06	SA_REP	80
9	176	01-JAN-07	31-DEC-07	SA_MAN	80
10	200	01-JUL-02	31-DEC-06	AC_ACCOUNT	90

```
DESCRIBE jobs
```

Name	Null?	Type
JOB_ID	NOT NULL	VARCHAR2(10)
JOB_TITLE	NOT NULL	VARCHAR2(35)
MIN_SALARY		NUMBER(6)
MAX_SALARY		NUMBER(6)

```
SELECT * FROM jobs;
```

#	JOB_ID	JOB_TITLE	MIN_SALARY	MAX_SALARY
1	AD_PRES	President	20080	40000
2	AD_VP	Administration Vice President	15000	30000
3	AD_ASST	Administration Assistant	3000	6000
4	FI_MGR	Finance Manager	8200	16000
5	FI_ACCOUNT	Accountant	4200	9000
6	AC_MGR	Accounting Manager	8200	16000
7	AC_ACCOUNT	Public Accountant	4200	9000
8	SA_MAN	Sales Manager	10000	20080
9	SA_REP	Sales Representative	6000	12008
10	PU_MAN	Purchasing Manager	8000	15000
11	PU_CLERK	Purchasing Clerk	2500	5500
12	ST_MAN	Stock Manager	5500	8500
13	ST_CLERK	Stock Clerk	2008	5000
14	SH_CLERK	Shipping Clerk	2500	5500
15	IT_PROG	Programmer	5000	10000
16	MK_MAN	Marketing Manager	9000	15000
17	MK_REP	Marketing Representative	4000	9000
18	HR REP	Human Resources Representative	4000	9000
19	PR REP	Public Relations Representative	4500	10500

DESCRIBE locations

Name	Null?	Type
JOB_ID	NOT NULL	VARCHAR2(10)
JOB_TITLE	NOT NULL	VARCHAR2(35)
MIN_SALARY		NUMBER(6)
MAX_SALARY		NUMBER(6)

SELECT * FROM locations;

LOCATION_ID	STREET_ADDRESS	POSTAL_CODE	CITY	STATE_PROVINCE	COUNTRY_ID
1	1000 1297 Via Cola di Rie	00989	Roma	(null)	IT
2	1100 93091 Calle della Testa	10934	Venice	(null)	IT
3	1200 2017 Shinjuku-ku	1689	Tokyo	Tokyo Prefecture	JP
4	1300 9450 Kamiya-cho	6823	Hiroshima	(null)	JP
5	1400 2014 Jabberwocky Rd	26192	Southlake	Texas	US
6	1500 2011 Interiors Blvd	99236	South San Francisco	California	US
7	1600 2007 Zagora St	50090	South Brunswick	New Jersey	US
8	1700 2004 Charade Rd	98199	Seattle	Washington	US
9	1800 147 Spadina Ave	M5V 2L7	Toronto	Ontario	CA
10	1900 6092 Boxwood St	Y5W 9T2	Whitehorse	Yukon	CA
11	2000 40-5-12 Laogianggen	190518	Beijing	(null)	CN
12	2100 1298 Vileparle (E)	490231	Bombay	Maharashtra	IN
13	2200 12-98 Victoria Street	2901	Sydney	New South Wales	AU
14	2300 198 Clementi North	540198	Singapore	(null)	SG
15	2400 8204 Arthur St	(null)	London	(null)	UK
16	2500 Magdalen Centre, The Oxford Science Park	OX9 9ZB	Oxford	Oxford	UK
17	2600 9702 Chester Road	09629850293	Stretford	Manchester	UK
18	2700 Schwanthalerstr. 7031	80925	Munich	Bavaria	DE
19	2800 Rua Frei Caneca 1360	01307-002	Sao Paulo	Sao Paulo	BR
20	2900 20 Rue des Corps-Saints	1730	Geneva	Geneve	CH
21	3000 Murtenstrasse 921	3095	Bern	BE	CH
22	3100 Pieter Breughelstraat 837	3029SK	Utrecht	Utrecht	NL
23	3200 Mariano Escobedo 9991	11932	Mexico City	Distrito Federal	MX

```
DESCRIBE regions
```

Name	Null?	Type
REGION_ID	NOT NULL	NUMBER
REGION_NAME		VARCHAR2(25)

```
SELECT * FROM regions
```

REGION_ID	REGION_NAME
1	1 Europe
2	2 Americas
3	3 Asia
4	4 Middle East and Africa

JB SQL Developer

ORACLE

Copyright © 2013, Oracle und/oder verbundene Unternehmen. All rights reserved. Alle Rechte vorbehalten.

Ziele

Nach Beendigung dieses Anhangs haben Sie folgende Ziele erreicht:

- **Schlüsselfeatures von Oracle SQL Developer auflisten**
- **Menüoptionen von Oracle SQL Developer angeben**
- **Datenbankverbindungen erstellen**
- **Datenbankobjekte verwalten**
- **SQL Worksheet verwenden**
- **SQL-Skripte speichern und ausführen**
- **Berichte erstellen und speichern**
- **Data Modeler-Optionen in SQL Developer durchsuchen**

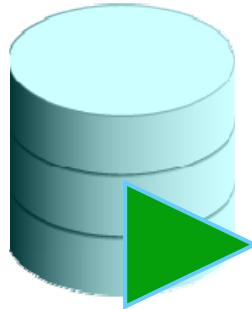


Copyright © 2013, Oracle und/oder verbundene Unternehmen. All rights reserved. Alle Rechte vorbehalten.

In diesem Anhang wird das grafische Tool SQL Developer vorgestellt. Sie lernen, wie Sie SQL Developer für Aufgaben im Bereich der Datenbankentwicklung verwenden. Außerdem wird beschrieben, wie Sie SQL-Anweisungen und SQL-Skripte mit dem SQL Worksheet ausführen.

Was ist Oracle SQL Developer?

- **Oracle SQL Developer ist ein grafisches Tool, das die Produktivität erhöht und Aufgaben der Datenbankentwicklung vereinfacht.**
- **Sie können sich mit der standardmäßigen Oracle-Datenbankauthentifizierung bei jedem Oracle-Zieldatenbankschema anmelden.**



SQL Developer

ORACLE®

Copyright © 2013, Oracle und/oder verbundene Unternehmen. All rights reserved. Alle Rechte vorbehalten.

Oracle SQL Developer ist ein kostenloses grafisches Tool, das Ihre Produktivität steigert und Routineaufgaben der Datenbankentwicklung vereinfacht. Mit wenigen Mausklicks können Sie problemlos Stored Procedures erstellen und debuggen, SQL-Anweisungen testen und Optimizer-Pläne anzeigen.

SQL Developer, das visuelle Tool zur Datenbankentwicklung, vereinfacht folgende Aufgaben:

- Datenbankobjekte durchsuchen und verwalten
- SQL-Anweisungen und -Skripte ausführen
- PL/SQL-Anweisungen bearbeiten und debuggen
- Berichte erstellen

Sie können sich mit der standardmäßigen Oracle-Datenbankauthentifizierung bei jedem Oracle-Zieldatenbankschema anmelden. Nach der Anmeldung können Sie Vorgänge für die Objekte in der Datenbank ausführen.

SQL Developer ist die Verwaltungsschnittstelle für den Oracle Application Express Listener. Über die neue Schnittstelle können Sie globale Einstellungen und Einstellungen für mehrere Datenbanken mit unterschiedlichen Datenbankverbindungen für den Application Express Listener festlegen. Objekte lassen sich in SQL Developer nach Tabellen- oder Spaltenname mit Drag & Drop in das Worksheet ziehen. Neben verbesserten DB Diff-Vergleichsoptionen werden auch GRANT-Anweisungen im SQL-Editor und DB Doc-Berichte unterstützt. Zusätzlich unterstützt SQL Developer Funktionen aus Oracle Database 12c.

SQL Developer – Spezifikationen

- **Im Lieferumfang von Oracle Database 12c Release 1 enthalten**
- **In Java entwickelt**
- **Unterstützt die Plattformen Windows, Linux und Mac OS X**
- **Standardkonnektivität durch den JDBC-Thin-Treiber**
- **Anmeldung bei Oracle Database Version 9.2.0.1 oder höher möglich**



Copyright © 2013, Oracle und/oder verbundene Unternehmen. All rights reserved. Alle Rechte vorbehalten.

Oracle SQL Developer ist standardmäßig im Lieferumfang von Oracle Database 12c Release 1 enthalten. SQL Developer wurde mithilfe der Oracle JDeveloper-IDE (Integrated Development Environment, integrierte Entwicklungsumgebung) in Java entwickelt und ist damit ein plattformübergreifendes Tool, das unter Windows, Linux und Mac OS X ausgeführt werden kann.

Die Standardkonnektivität zur Datenbank erfolgt über den Java Database Connectivity-(JDBC-)Thin-Treiber, sodass kein Oracle Home-Verzeichnis erforderlich ist. Für SQL Developer benötigen Sie kein Installationsprogramm. Sie dekomprimieren einfach die heruntergeladene Datei. Mit SQL Developer können sich Benutzer bei Oracle Database 9.2.0.1 oder höher sowie bei allen Oracle-Datenbankeditionen einschließlich Express Edition anmelden.

Hinweis

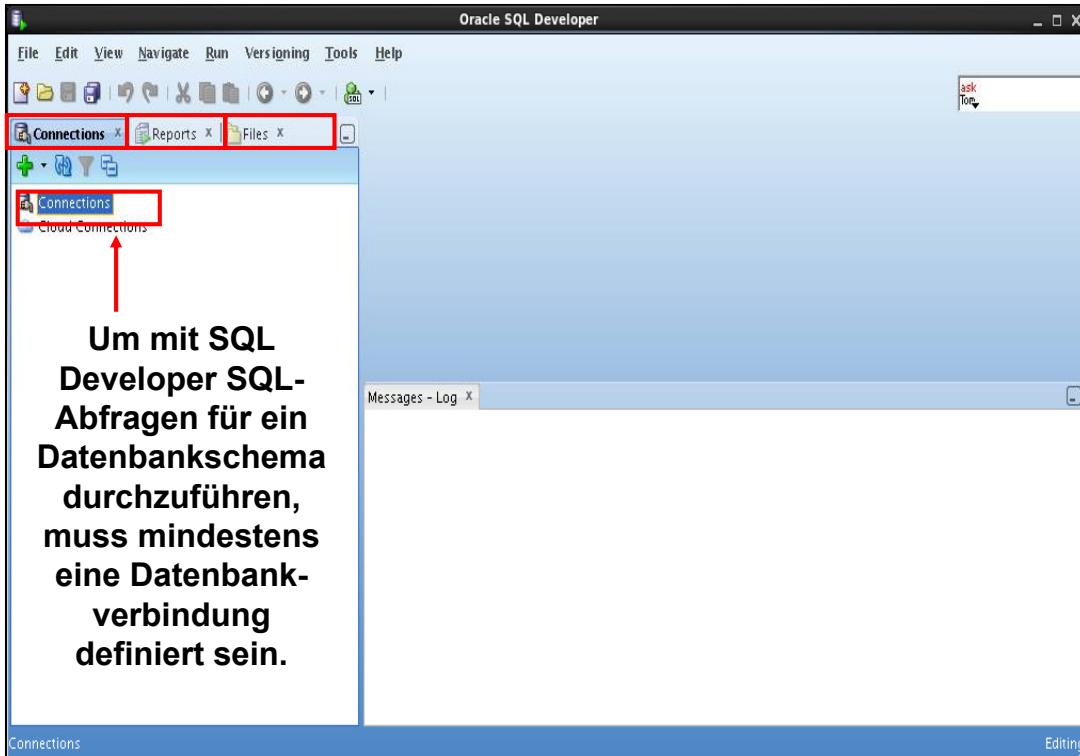
Für Oracle Database 12c Release 1 müssen Sie SQL Developer herunterladen und installieren. SQL Developer kann kostenlos über folgenden Link heruntergeladen werden:

<http://www.oracle.com/technetwork/developer-tools/sql-developer/downloads/index.html>

Anweisungen zur Installation von SQL Developer finden Sie auf der Website unter:

<http://www.oracle.com/technetwork/developer-tools/sql-developer/overview/index.html>

SQL Developer 3.2 – Benutzeroberfläche



Copyright © 2013, Oracle und/oder verbundene Unternehmen. All rights reserved. Alle Rechte vorbehalten.

Die Benutzeroberfläche von SQL Developer umfasst (von links nach rechts) drei Haupt-navigationsregisterkarten:

- **Connections:** In dieser Registerkarte können Sie Datenbankobjekte und Benutzer durchsuchen, auf die Sie Zugriff haben.
- **Reports:** In dieser durch das Symbol **Reports** dargestellten Registerkarte führen Sie vordefinierte Berichte aus oder erstellen und fügen eigene Berichte hinzu.
- **Files:** In dieser durch das Symbol des Ordners **Files** dargestellten Registerkarte können Sie von Ihrem lokalen Rechner aus auf Dateien zugreifen, ohne über das Menü **File > Open** navigieren zu müssen.

Allgemeine Navigation und Verwendung

Bei SQL Developer dient die linke Seite zur Navigation, zum Suchen und Wählen von Objekten. Auf der rechten Seite werden Informationen über die gewählten Objekte angezeigt. Über Voreinstellungen können Sie viele Aspekte der Darstellung und des Verhaltens von SQL Developer anpassen.

Hinweis: Sie müssen mindestens eine Verbindung definieren, damit Sie sich bei einem Datenbankschema anmelden und SQL-Abfragen bzw. Prozeduren oder Funktionen ausführen können.

Menüs

Folgende Menüs enthalten Standardeinträge sowie zusätzliche Einträge für spezifische Features von SQL Developer:

- **View:** Enthält Optionen, die bestimmen, was in der Benutzeroberfläche von SQL Developer angezeigt wird
- **Navigate:** Enthält Optionen für die Navigation zu verschiedenen Bereichen und die Ausführung von Unterprogrammen
- **Run:** Enthält die Optionen **Run File** und **Execution Profile**, die von Bedeutung sind, wenn eine Funktion oder Prozedur gewählt wurde, sowie Debugging-Optionen.
- **Versioning:** Bietet integrierte Unterstützung für folgende Versionierungs- und Quellkontrollsysteme: Concurrent Versions System (CVS) und Subversion
- **Tools:** Ruft SQL Developer-Tools wie SQL*Plus, Preferences oder SQL Worksheet auf und enthält Optionen für die Migration von Fremddatenbanken zu Oracle

Hinweis: Das Menü **Run** enthält auch Optionen, die von Bedeutung sind, wenn eine Funktion oder Prozedur für das Debugging gewählt wurde.

Datenbankverbindungen erstellen

- Um SQL Developer verwenden zu können, ist mindestens eine Datenbankverbindung erforderlich.
- Sie können Verbindungen erstellen und testen für:
 - mehrere Datenbanken
 - mehrere Schemas
- SQL Developer importiert automatisch alle Verbindungen, die in der Datei `tnsnames.ora` auf dem System definiert sind.
- Verbindungen können in eine Extensible Markup Language-(XML-)Datei exportiert werden.
- Jede zusätzlich erstellte Datenbankverbindung wird in der Connections Navigator-Hierarchie aufgelistet.

ORACLE®

Copyright © 2013, Oracle und/oder verbundene Unternehmen. All rights reserved. Alle Rechte vorbehalten.

Eine Verbindung ist ein SQL Developer-Objekt, das die nötigen Informationen angibt, mit denen Sie sich als bestimmter Benutzer bei einer bestimmten Datenbank anmelden können. Um SQL Developer zu verwenden, muss mindestens eine Datenbankverbindung vorhanden sein bzw. erstellt oder importiert werden.

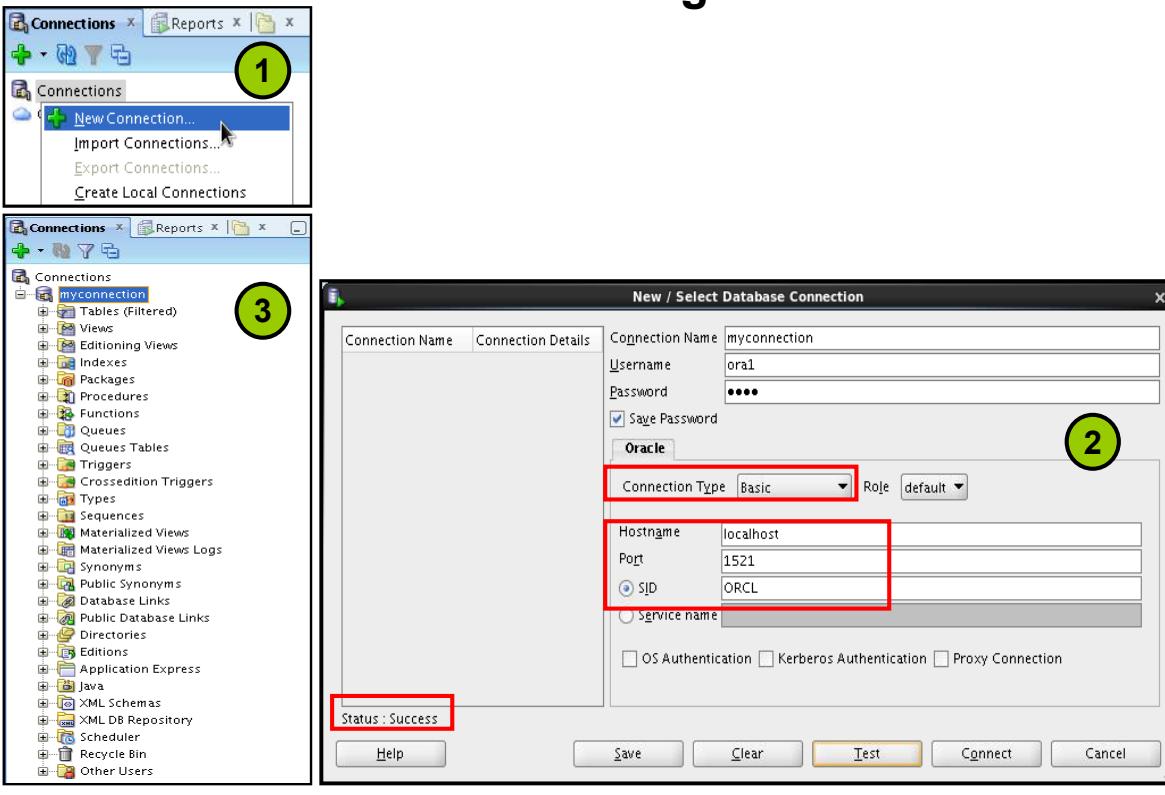
Sie können Verbindungen für mehrere Datenbanken und mehrere Schemas erstellen und testen. Standardmäßig befindet sich die Datei `tnsnames.ora` im Verzeichnis `$ORACLE_HOME/network/admin`. Die Datei kann jedoch auch in einem durch die Umgebungsvariable `TNS_ADMIN` oder durch einen Registrierungswert angegebenen Verzeichnis gespeichert werden. Wenn Sie SQL Developer starten und das Dialogfeld **Database Connections** anzeigen, importiert SQL Developer automatisch alle Verbindungen, die in der Datei `tnsnames.ora` auf dem System definiert sind.

Hinweis: Wenn die in der Datei `tnsnames.ora` definierten Verbindungen unter Windows nicht von SQL Developer verwendet werden, definieren Sie `TNS_ADMIN` als Systemumgebungsvariable.

Sie können Verbindungen zur späteren Wiederverwendung in eine XML-Datei exportieren.

Außerdem haben Sie die Möglichkeit, sich mehrfach als jeweils unterschiedlicher Benutzer bei derselben Datenbank anzumelden oder sich bei verschiedenen Datenbanken anzumelden.

Datenbankverbindungen erstellen



Um eine Datenbankverbindung zu erstellen, gehen Sie wie folgt vor:

1. Klicken Sie in der Registerkarte **Connections** mit der rechten Maustaste auf **Connections**, und wählen Sie **New Connection**.
2. Geben Sie im Fenster **New>Select Database Connection** den Verbindungsnamen ein. Geben Sie den Benutzernamen und das Kennwort des Schemas ein, bei dem Sie sich anmelden möchten.
 - a. Wählen Sie in der Dropdown-Liste **Role** entweder *default* oder **SYSDBA**. (**SYSDBA** wird für den Benutzer **sys** oder einen Benutzer mit DBA-Berechtigungen gewählt.)
 - b. Für den Verbindungstyp stehen folgende Optionen zur Wahl:
 - Basic:** Geben Sie für diesen Typ den Hostnamen und die SID für die Datenbank ein, bei der Sie sich anmelden möchten. **Port** ist bereits auf 1521 festgelegt. Sie können aber auch den Servicenamen direkt eingeben, wenn Sie eine Remote-Datenbankverbindung verwenden.
 - TNS:** Sie können jeden Datenbankalias wählen, der aus der Datei `tnsnames.ora` importiert wurde.
 - LDAP:** Sie können Datenbankservices in Oracle Internet Directory, einer Komponente von Oracle Identity Management, nachschlagen.
 - Advanced:** Sie können einen benutzerdefinierten Java Database Connectivity-(JDBC-)URL für die Anmeldung bei der Datenbank definieren.

Local/Bequeath: Befinden sich Client und Datenbank auf demselben Rechner, kann die Clientverbindung direkt an einen dedizierten Serverprozess übergeben werden, ohne dass der Listener einbezogen wird.

- c. Klicken Sie auf **Test**, um sicherzustellen, dass die Verbindung richtig festgelegt wurde.
- d. Klicken Sie auf **Connect**.

Bei Aktivierung des Kontrollkästchens **Save Password** wird das Kennwort in einer XML-Datei gespeichert. Wenn Sie dann die SQL Developer-Verbindung beenden und wieder öffnen, werden Sie nicht mehr zur Eingabe des Kennwortes aufgefordert.

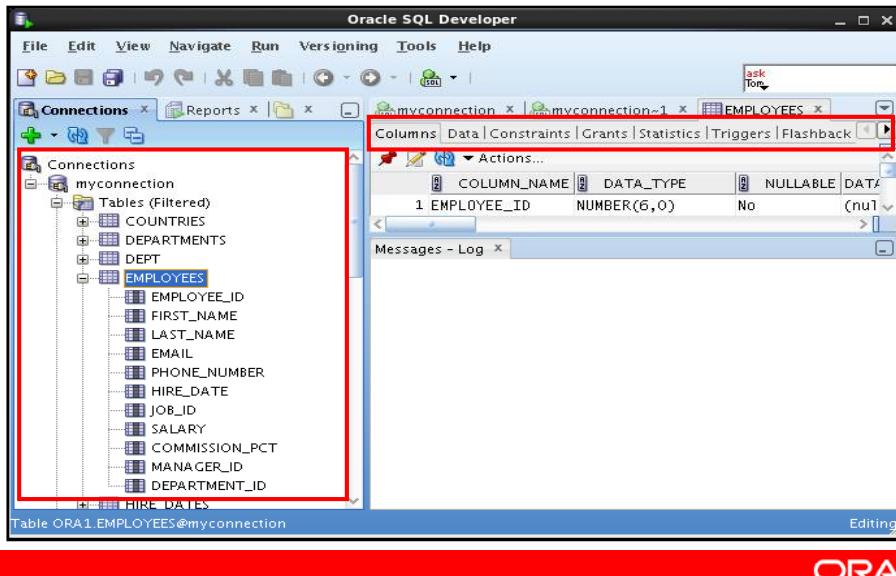
3. Die Verbindung wird dem Connections Navigator hinzugefügt. Sie können sie einblenden, um die Datenbankobjekte und Objektdefinitionen anzuzeigen, etwa Abhängigkeiten, Details und Statistiken.

Hinweis: Im Fenster **New>Select Database Connection** können Sie über die Registerkarten **Access**, **MySQL** und **SQL Server** auch Verbindungen zu Fremddatenquellen definieren. Diese Verbindungen sind allerdings Read-Only-Verbindungen, mit denen Sie nur Objekte und Daten in der jeweiligen Datenquelle durchsuchen können.

Datenbankobjekte durchsuchen

Mit dem Connections Navigator lassen sich:

- viele Objekte in einem Datenbankschema durchsuchen
- Objektdefinitionen auf einen Blick prüfen



Copyright © 2013, Oracle und/oder verbundene Unternehmen. All rights reserved. Alle Rechte vorbehalten.

Wenn Sie eine Datenbankverbindung erstellt haben, können Sie mit dem Connections Navigator viele Objekte in einem Datenbankschema durchsuchen, darunter Tabellen, Views, Indizes, Packages, Prozeduren, Trigger und Typen.

Bei SQL Developer dient die linke Seite der Navigation, also dem Suchen und Wählen von Objekten. Auf der rechten Seite werden Informationen zu den gewählten Objekten angezeigt. Über Voreinstellungen können Sie zahlreiche Aspekte der Darstellung von SQL Developer anpassen.

Die Definitionen der Objekte werden in verschiedenen Registerkarten angezeigt, die Informationen aus dem Data Dictionary enthalten. Wenn Sie beispielsweise eine Tabelle im Navigator wählen, werden alle Einzelheiten über Spalten, Constraints, Berechtigungen, Statistiken, Trigger usw. in einem übersichtlichen, in Registerkarten unterteilten Fenster angezeigt.

Um die Definition der Tabelle `EMPLOYEES` anzuzeigen (siehe Folie), gehen Sie wie folgt vor:

1. Blenden Sie im Connections Navigator den Knoten **Connections** ein.
2. Blenden Sie **Tables** ein.
3. Klicken Sie auf `EMPLOYEES`. Standardmäßig ist die Registerkarte **Columns** mit einer Spaltenbeschreibung der Tabelle aktiviert. Mithilfe der Registerkarte **Data** können Sie die Tabellendaten anzeigen und außerdem neue Zeilen eingeben, Daten aktualisieren und diese Änderungen in der Datenbank festschreiben.

Tabellenstrukturen anzeigen

Struktur einer Tabelle mit dem Befehl **DESCRIBE** anzeigen:

The screenshot shows the SQL Developer interface with a connection named 'myconnection'. In the 'Worksheet' tab, the command 'DESC EMPLOYEES' is entered. Below the worksheet, the 'Script Output' tab displays the results of the describe command. The output is a table showing the columns of the EMPLOYEES table:

Name	Null	Type
EMPLOYEE_ID	NOT NULL	NUMBER(6)
FIRST_NAME		VARCHAR2(20)
LAST_NAME	NOT NULL	VARCHAR2(25)
EMAIL	NOT NULL	VARCHAR2(25)
PHONE_NUMBER		VARCHAR2(20)
HIRE_DATE	NOT NULL	DATE
JOB_ID	NOT NULL	VARCHAR2(10)
SALARY		NUMBER(8,2)
COMMISSION_PCT		NUMBER(2,2)
MANAGER_ID		NUMBER(6)
DEPARTMENT_ID		NUMBER(4)

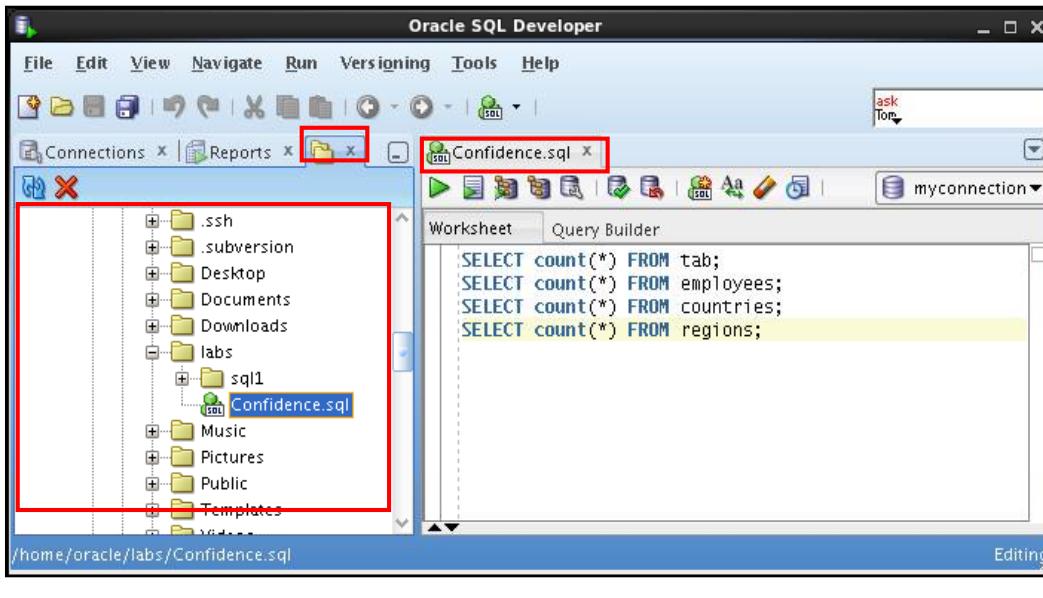
ORACLE

Copyright © 2013, Oracle und/oder verbundene Unternehmen. All rights reserved. Alle Rechte vorbehalten.

In SQL Developer können Sie die Struktur einer Tabelle mit dem Befehl **DESCRIBE** anzeigen. Daraufhin werden die Spaltennamen und Datentypen sowie ein Hinweis darauf angezeigt, ob eine Spalte Daten enthalten muss.

Dateien durchsuchen

Mit dem File Navigator das Dateisystem untersuchen und Systemdateien öffnen



ORACLE

Copyright © 2013, Oracle und/oder verbundene Unternehmen. All rights reserved. Alle Rechte vorbehalten.

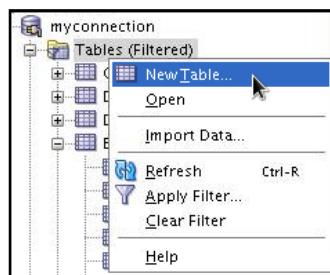
Datenbankobjekte durchsuchen

Mit dem File Navigator können Sie Systemdateien durchsuchen und öffnen.

- Um den File Navigator anzuzeigen, klicken Sie auf die Registerkarte **View**, und wählen Sie **Files**. Alternativ können Sie zu **View > Files** navigieren.
- Um den Inhalt einer Datei anzuzeigen, doppelklicken Sie auf einen Dateinamen. Der Inhalt der Datei wird dann im SQL Worksheet-Bereich angezeigt.

Schemaobjekte erstellen

- **SQL Developer unterstützt die Erstellung beliebiger Schemaobjekte:**
 - durch Ausführung einer SQL-Anweisung im SQL Worksheet
 - mithilfe des Kontextmenüs
- **Objekte in einem Bearbeitungsdialogfeld oder über eines der zahlreichen Kontextmenüs bearbeiten**
- **DDL (Data Definition Language) für Anpassungen wie die Erstellung eines neuen Objekts oder die Bearbeitung eines vorhandenen Schemaobjekts anzeigen**



ORACLE

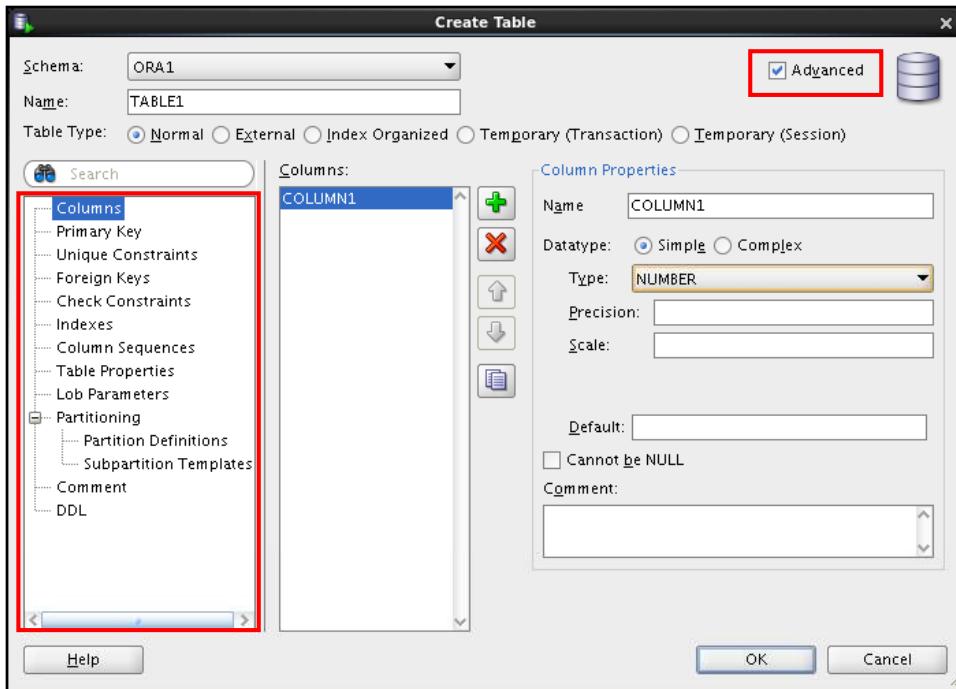
Copyright © 2013, Oracle und/oder verbundene Unternehmen. All rights reserved. Alle Rechte vorbehalten.

SQL Developer unterstützt die Erstellung beliebiger Schemaobjekte durch Ausführung einer SQL-Anweisung im SQL Worksheet. Alternativ können Sie Objekte mithilfe der Kontextmenüs erstellen. Nach ihrer Erstellung können Sie die Objekte in einem Bearbeitungsdialogfeld oder über eines der zahlreichen Kontextmenüs bearbeiten.

Bei der Erstellung neuer Objekte oder der Bearbeitung vorhandener Objekte kann für Prüfzwecke die DDL angezeigt werden. Die Option **Export DDL** ist verfügbar, wenn Sie den vollständigen DDL-Code für ein oder mehrere Objekte im Schema erstellen möchten.

Auf der Folie wird gezeigt, wie Sie eine Tabelle über das Kontextmenü erstellen. Um ein Dialogfeld zur Erstellung einer neuen Tabelle zu öffnen, klicken Sie mit der rechten Maustaste auf **Tables** und wählen **New Table**. Die Dialogfelder zum Erstellen und Bearbeiten von Datenbankobjekten verfügen über mehrere Registerkarten, die jeweils eine logische Zusammenstellung von Eigenschaften für den entsprechenden Objekttyp enthalten.

Neue Tabellen erstellen – Beispiel



ORACLE

Copyright © 2013, Oracle und/oder verbundene Unternehmen. All rights reserved. Alle Rechte vorbehalten.

Wenn Sie im Dialogfeld **Create Table** das Kontrollkästchen **Advanced** deaktivieren, können Sie schnell eine Tabelle erstellen, indem Sie Spalten und einige häufig verwendete Features angeben.

Bei Aktivierung des Kontrollkästchens **Advanced** werden im Dialogfeld **Create Table** mehrere Optionen eingeblendet, mit denen Sie beim Erstellen der Tabelle erweiterte Features festlegen können.

Das Beispiel auf der Folie zeigt, wie die Tabelle **DEPENDENTS** durch Aktivierung des Kontrollkästchens **Advanced** erstellt wird.

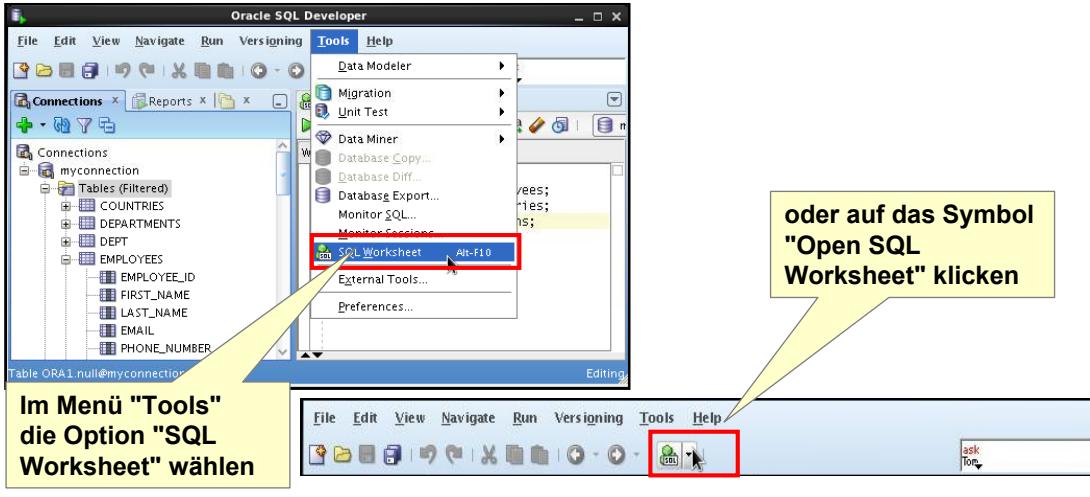
Um eine neue Tabelle zu erstellen, gehen Sie wie folgt vor:

1. Klicken Sie im Connections Navigator mit der rechten Maustaste auf **Tables**, und wählen Sie **Create TABLE**.
2. Aktivieren Sie im Dialogfeld **Create Table** das Kontrollkästchen **Advanced**.
3. Geben Sie Spalteninformationen an.
4. Klicken Sie auf **OK**.

Es empfiehlt sich, darüber hinaus in der Registerkarte **Primary Key** einen Primärschlüssel anzugeben. Um eine erstellte Tabelle zu bearbeiten, klicken Sie im Connections Navigator mit der rechten Maustaste auf die gewünschte Tabelle, und wählen Sie **Edit**.

SQL Worksheet

- **Mit dem SQL Worksheet SQL-, PL/SQL- und SQL*Plus-Anweisungen eingeben und ausführen**
- **Aktionen angeben, die von der mit dem Worksheet verknüpften Datenbankverbindung verarbeitet werden können**



Copyright © 2013, Oracle und/oder verbundene Unternehmen. All rights reserved. Alle Rechte vorbehalten.

Wenn Sie sich bei einer Datenbank anmelden, wird automatisch ein SQL Worksheet-Fenster für diese Verbindung geöffnet. Im SQL Worksheet können Sie SQL-, PL/SQL- und SQL*Plus-Anweisungen eingeben und ausführen. Das SQL Worksheet unterstützt nicht alle SQL*Plus-Anweisungen. Nicht unterstützte SQL*Plus-Anweisungen werden ignoriert und nicht an die Datenbank übergeben.

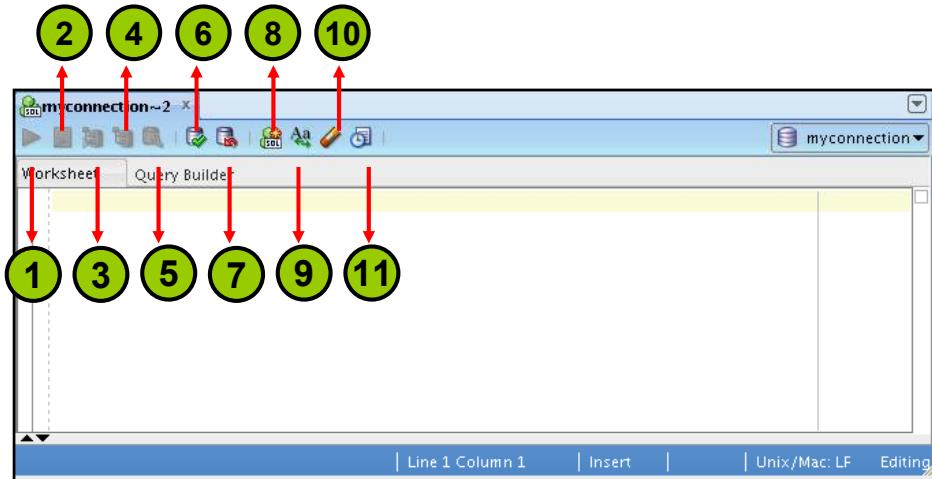
Sie können angeben, welche Aktionen die mit dem Worksheet verknüpfte Datenbankverbindung verarbeiten kann. Beispiele:

- Tabelle erstellen
- Daten einfügen
- Trigger erstellen und bearbeiten
- Daten aus einer Tabelle wählen
- Gewählte Daten in einer Datei speichern

Sie haben folgende Möglichkeiten, ein SQL Worksheet-Fenster anzuzeigen:

- Im Menü **Tools** die Option **SQL Worksheet** wählen
- Auf das Symbol **Open SQL Worksheet** klicken

SQL Worksheet



ORACLE

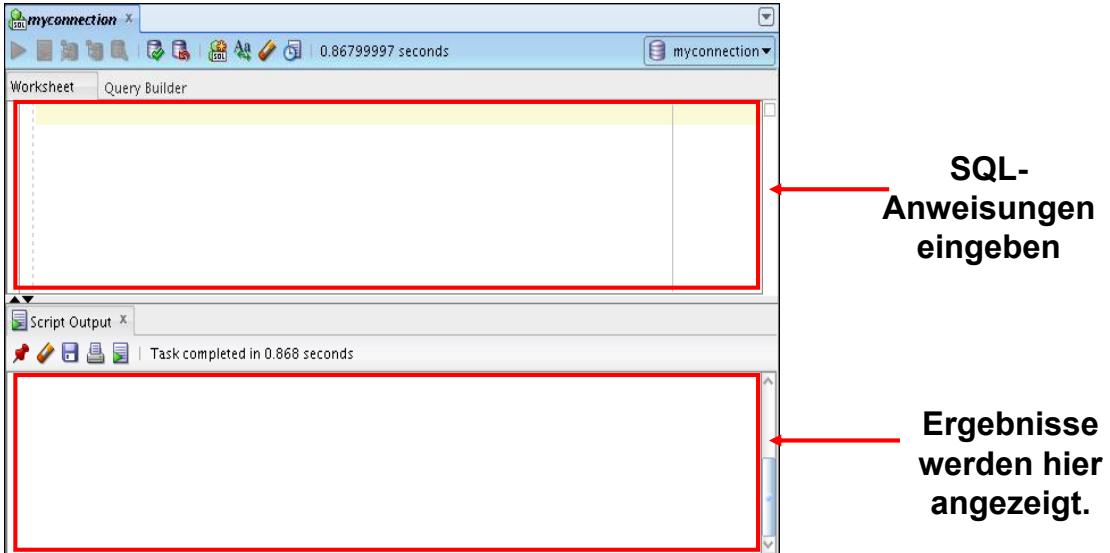
Copyright © 2013, Oracle und/oder verbundene Unternehmen. All rights reserved. Alle Rechte vorbehalten.

Möglicherweise möchten Sie Tasturbefehle oder Symbole für bestimmte Aufgaben wie die Ausführung von SQL-Anweisungen und Skripten oder die Anzeige der Historie von bereits ausgeführten SQL-Anweisungen verwenden. Die Symbolleiste des SQL Worksheets enthält folgende Symbole:

1. **Run Statement:** Führt die Anweisung an der Cursorposition im Feld **Enter SQL Statement** aus. Sie können Bind-Variablen, jedoch keine Austauschvariablen in den SQL-Anweisungen verwenden.
2. **Run Script:** Führt mithilfe von Script Runner alle Anweisungen im Feld **Enter SQL Statement** aus. Sie können Austauschvariablen, jedoch keine Bind-Variablen in den SQL-Anweisungen verwenden.
3. **Autotrace:** Generiert Traceinformationen für die Anweisung
4. **Explain Plan:** Generiert den Ausführungsplan, den Sie in der Registerkarte **Explain** anzeigen können
5. **SQL Tuning Advisory:** Analysiert großvolumige SQL-Anweisungen und bietet Tuning-Empfehlungen
6. **Commit:** Schreibt alle Änderungen in der Datenbank fest und beendet die Transaktion
7. **Rollback:** Verwirft alle Änderungen in der Datenbank, ohne sie festzuschreiben, und beendet die Transaktion

8. **Unshared SQL Worksheet:** Erstellt ein separates, nicht gemeinsam genutztes SQL Worksheet für eine Verbindung
9. **To Upper/Lower/InitCap:** Ändert die Schreibweise des gewählten Textes in Groß- oder Kleinbuchstaben bzw. die Verwendung großer Anfangsbuchstaben
10. **Clear:** Löscht die Anweisungen im Feld **Enter SQL Statement**
11. **SQL History:** Zeigt ein Dialogfeld mit Informationen zu den bereits ausgeführten SQL-Anweisungen an

SQL Worksheet



ORACLE

Copyright © 2013, Oracle und/oder verbundene Unternehmen. All rights reserved. Alle Rechte vorbehalten.

Wenn Sie sich bei einer Datenbank anmelden, wird automatisch ein SQL Worksheet-Fenster für diese Verbindung geöffnet. Im SQL Worksheet können Sie SQL-, PL/SQL- und SQL*Plus-Anweisungen eingeben und ausführen. Alle SQL- und PL/SQL-Befehle werden unterstützt und direkt vom SQL Worksheet an die Oracle-Datenbank übergeben. In SQL Developer verwendete SQL*Plus-Befehle müssen vor der Übergabe an die Datenbank vom SQL Worksheet interpretiert werden.

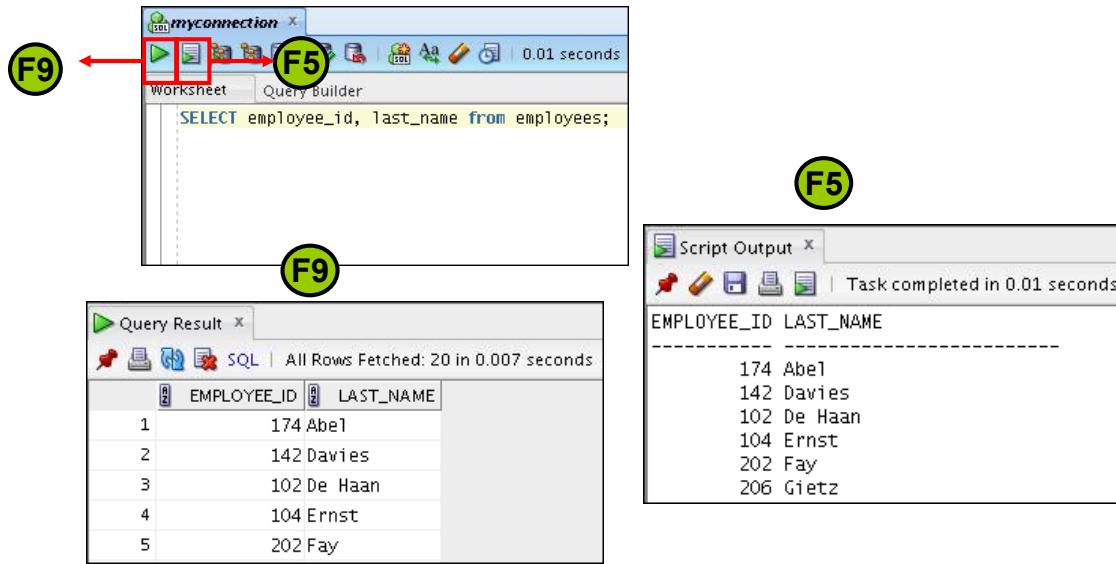
Das SQL Worksheet unterstützt derzeit eine Reihe von SQL*Plus-Befehlen. Nicht unterstützte SQL*Plus-Befehle werden ignoriert und nicht an die Oracle-Datenbank gesendet. Mit dem SQL Worksheet lassen sich SQL-Anweisungen und einige SQL*Plus-Befehle ausführen.

Um ein SQL Worksheet anzuzeigen, verwenden Sie eine der beiden folgenden Optionen:

- Im Menü **Tools** die Option **SQL Worksheet** wählen
- Auf das Symbol **Open SQL Worksheet** klicken

SQL-Anweisungen ausführen

Im Feld "Enter SQL Statement" eine oder mehrere SQL-Anweisungen eingeben

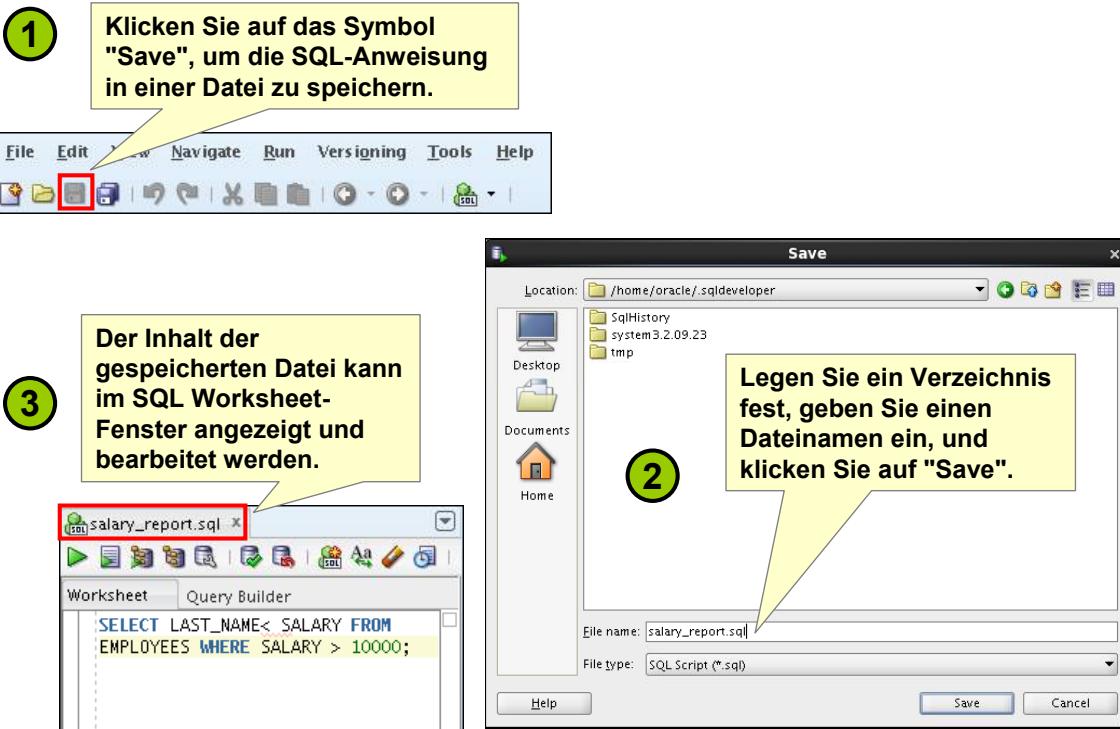


ORACLE

Copyright © 2013, Oracle und/oder verbundene Unternehmen. All rights reserved. Alle Rechte vorbehalten.

Das Beispiel auf der Folie zeigt die unterschiedliche Ausgabe derselben Abfrage, einmal für die Taste F9 oder das Symbol **Execute Statement** und im Vergleich dazu für die Taste F5 oder das Symbol **Run Script**.

SQL-Skripte speichern



ORACLE

Copyright © 2013, Oracle und/oder verbundene Unternehmen. All rights reserved. Alle Rechte vorbehalten.

Sie können die SQL-Anweisungen im SQL Worksheet in einer Textdatei speichern. Um den Inhalt des Feldes **Enter SQL Statement** zu speichern, gehen Sie wie folgt vor:

1. Klicken Sie auf das Symbol **Save**, oder wählen Sie die Menüoption **File > Save**.
2. Geben Sie im Dialogfeld **Save** einen Dateinamen und das gewünschte Verzeichnis für die Datei ein.
3. Klicken Sie auf **Save**.

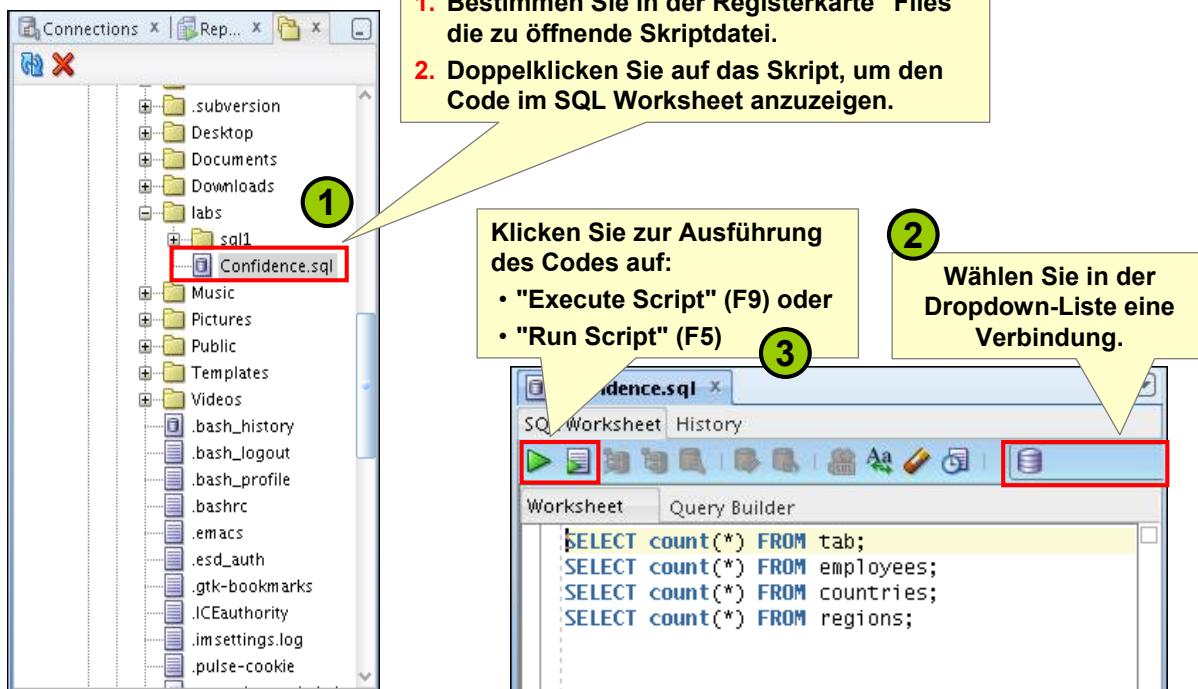
Nachdem Sie den Inhalt in einer Datei gespeichert haben, wird im Fenster **Enter SQL Statement** eine Registerkarte mit dem Dateinhalt angezeigt. Es können mehrere Dateien gleichzeitig geöffnet sein. Jede Datei wird in Form einer Registerkarte angezeigt.

Skriptpfad

Sie können einen Standardpfad für die Suche nach Skripten und deren Speicherung wählen. Geben Sie hierfür unter **Tools > Preferences > Database > Worksheet Parameters** einen Wert im Feld **Select default path to look for scripts** ein.

Gespeicherte Skriptdateien ausführen –

1. Methode



Copyright © 2013, Oracle und/oder verbundene Unternehmen. All rights reserved. Alle Rechte vorbehalten.

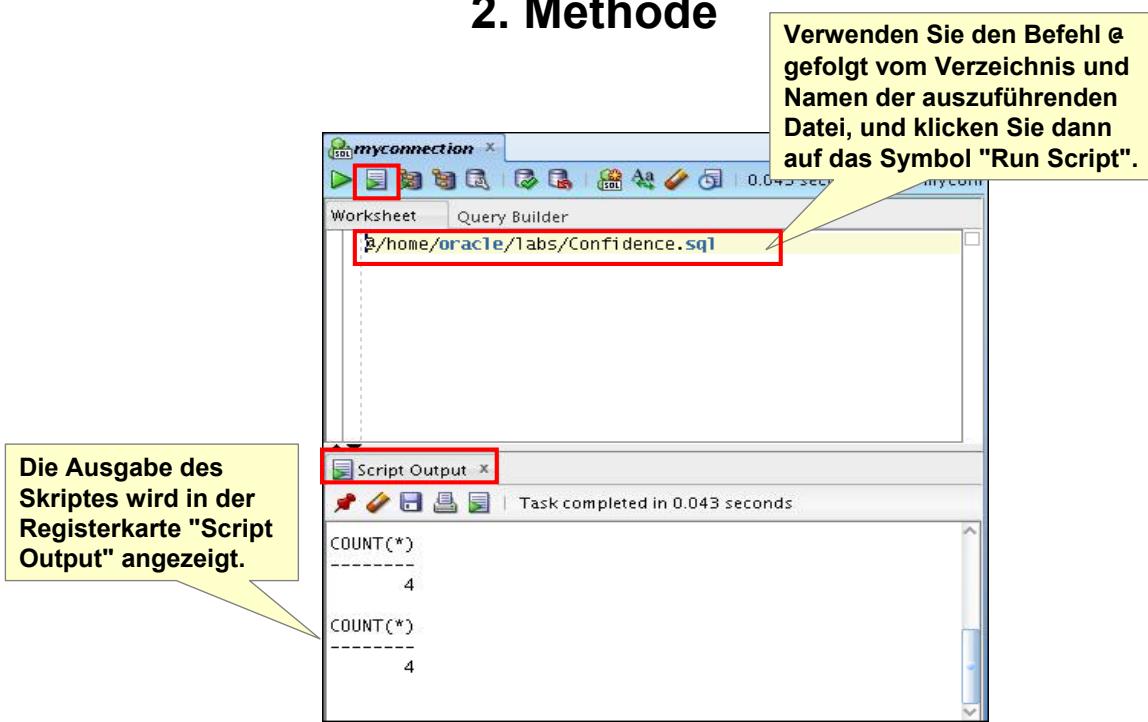
Um eine Skriptdatei zu öffnen und den Code im SQL Worksheet-Bereich anzuzeigen, gehen Sie wie folgt vor:

1. Wählen Sie im File Navigator die zu öffnende Skriptdatei, oder navigieren Sie zur gewünschten Datei.
2. Öffnen Sie die Datei mit einem Doppelklick. Der Code der Skriptdatei wird im SQL Worksheet-Bereich angezeigt.
3. Wählen Sie in der Dropdown-Liste **Connection** eine Verbindung.
4. Um den Code auszuführen, klicken Sie in der SQL Worksheet-Symbolleiste auf das Symbol **Run Script** (F5). Wenn Sie in der Dropdown-Liste **Connection** keine Verbindung gewählt haben, wird das Dialogfeld **Connection** angezeigt. Wählen Sie die gewünschte Verbindung für die Ausführung des Skriptes.

Alternativ können Sie auch wie folgt vorgehen:

1. Wählen Sie **File > Open**. Das Dialogfeld **Open** wird angezeigt.
2. Wählen Sie im Dialogfeld **Open** die zu öffnende Skriptdatei, oder navigieren Sie zur gewünschten Datei.
3. Klicken Sie auf **Open**. Der Code der Skriptdatei wird im SQL Worksheet-Bereich angezeigt.
4. Wählen Sie in der Dropdown-Liste **Connection** eine Verbindung.
5. Um den Code auszuführen, klicken Sie in der SQL Worksheet-Symbolleiste auf das Symbol **Run Script** (F5). Wenn Sie in der Dropdown-Liste **Connection** keine Verbindung gewählt haben, wird das Dialogfeld **Connection** angezeigt. Wählen Sie die gewünschte Verbindung für die Ausführung des Skriptes.

Gespeicherte Skriptdateien ausführen – 2. Methode



ORACLE

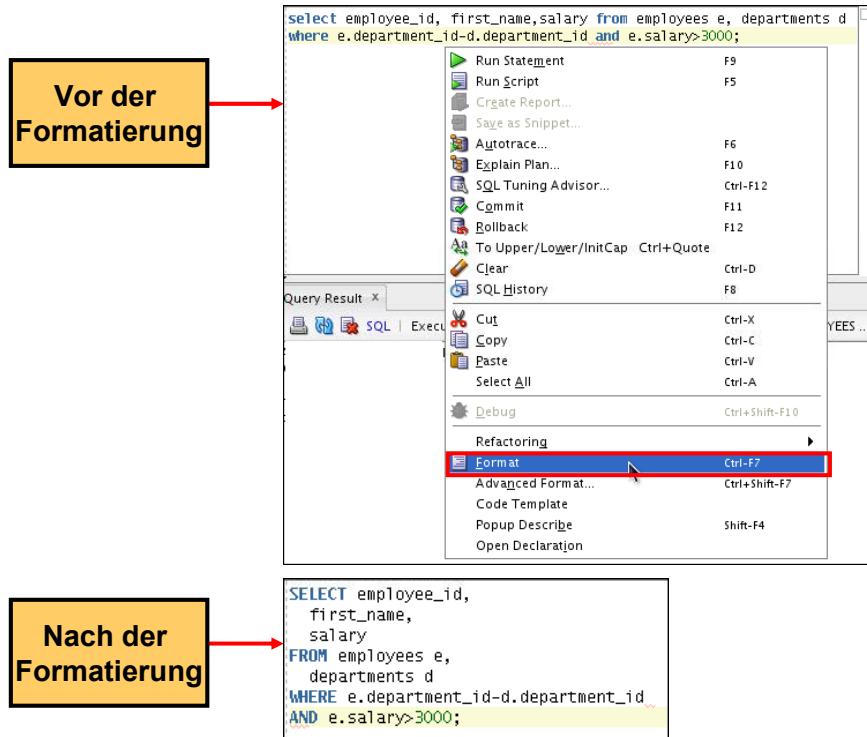
Copyright © 2013, Oracle und/oder verbundene Unternehmen. All rights reserved. Alle Rechte vorbehalten.

Um ein gespeichertes SQL-Skript auszuführen, gehen Sie wie folgt vor:

1. Verwenden Sie im Fenster **Enter SQL Statement** den Befehl `@` gefolgt vom Verzeichnis und Namen der auszuführenden Datei.
2. Klicken Sie auf das Symbol **Run Script**.

Die Ergebnisse der Skriptausführung werden in der Registerkarte **Script Output** angezeigt. Sie können die Skriptausgabe auch speichern, indem Sie in der Registerkarte **Script Output** auf das Symbol **Save** klicken. Im angezeigten Dialogfeld **File Save** können Sie Name und Verzeichnis für die Datei angeben.

SQL-Code formatieren



ORACLE

Copyright © 2013, Oracle und/oder verbundene Unternehmen. All rights reserved. Alle Rechte vorbehalten.

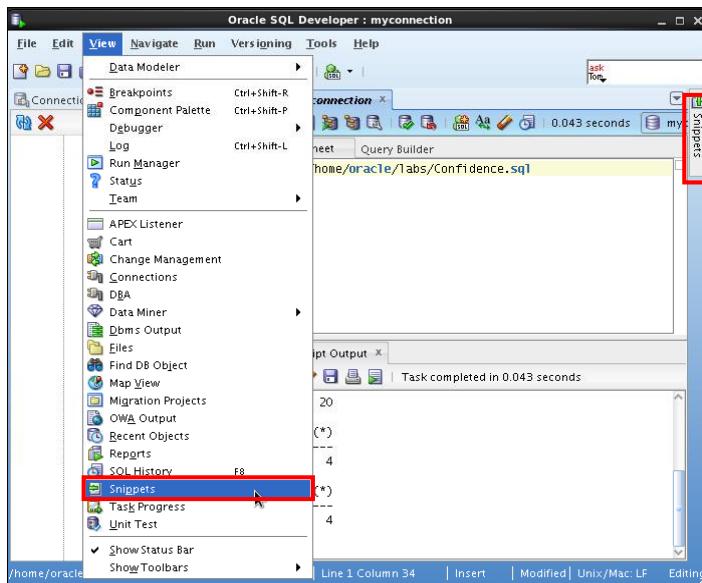
Es kann sinnvoll sein, die Einrückungen, Zeichenabstände, Groß-/Kleinschreibung und Zeilenabstände im SQL-Code übersichtlicher zu gestalten. SQL Developer bietet ein Feature zur Formatierung von SQL-Code.

Um SQL-Code zu formatieren, klicken Sie mit der rechten Maustaste in den Anweisungsbereich und wählen **Format SQL**.

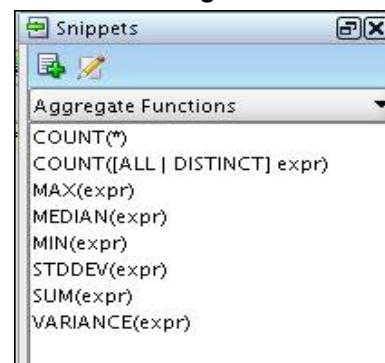
Das Beispiel auf der Folie zeigt, dass die Schlüsselwörter im SQL-Code vor der Formatierung nicht in Großbuchstaben dargestellt werden und die Anweisung nicht die richtigen Einrückungen aufweist. Nach der Formatierung ist der SQL-Code übersichtlicher, da die Schlüsselwörter groß geschrieben sind und die Anweisung mit den richtigen Einrückungen versehen ist.

Snippets

Snippets sind Codefragmente, die unter Umständen nur Syntax oder Beispiele enthalten.



Wenn Sie den Cursor hier platzieren, wird das Fenster "Snippets" angezeigt. Aus der Dropdown-Liste können Sie die gewünschte Funktionskategorie wählen.



ORACLE

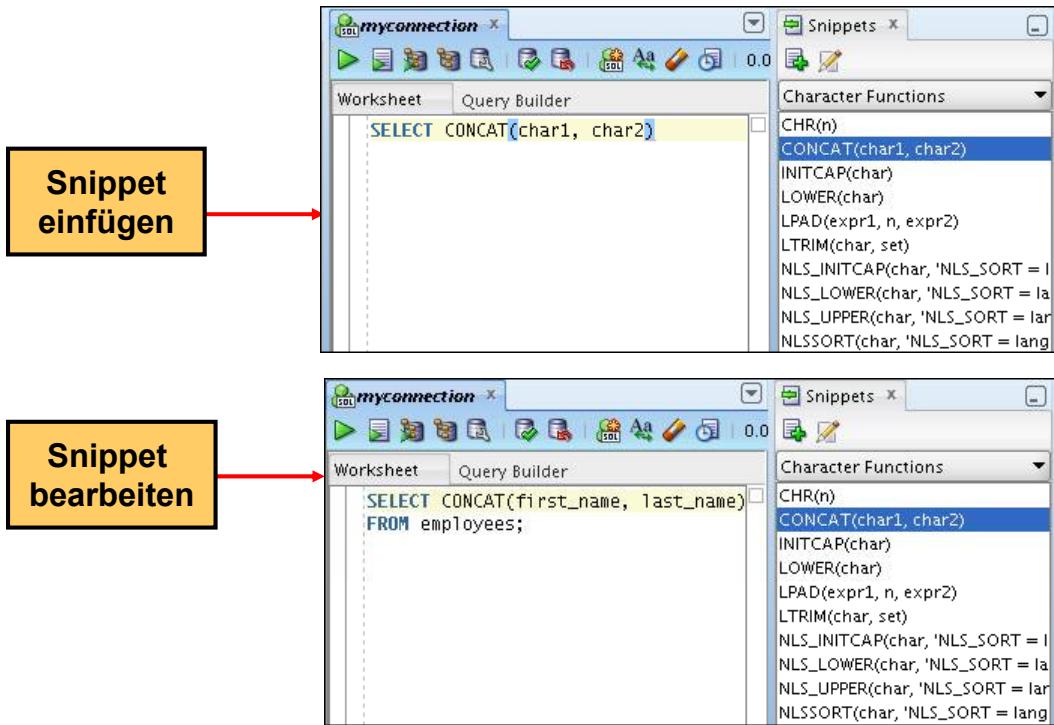
Copyright © 2013, Oracle und/oder verbundene Unternehmen. All rights reserved. Alle Rechte vorbehalten.

Bei der Arbeit mit dem SQL Worksheet bzw. der Erstellung oder Bearbeitung von PL/SQL-Funktion oder -Prozeduren möchten Sie eventuell bestimmte Codefragmente verwenden. SQL Developer verfügt hierzu über das Feature Snippets. Snippets sind Codefragmente wie SQL-Funktionen, Optimizer Hints oder verschiedene PL/SQL-Programmiertechniken. Sie können Snippets in das Editorfenster ziehen.

Um Snippets anzuzeigen, wählen Sie **View > Snippets**.

Daraufhin wird rechts das Fenster **Snippets** angezeigt. Wählen Sie eine Gruppe aus der Dropdown-Liste. Über die Schaltfläche **Snippets** am rechten Fensterrand können Sie das ausgeblendete Fenster **Snippets** einblenden.

Snippets – Beispiel



ORACLE

Copyright © 2013, Oracle und/oder verbundene Unternehmen. All rights reserved. Alle Rechte vorbehalten.

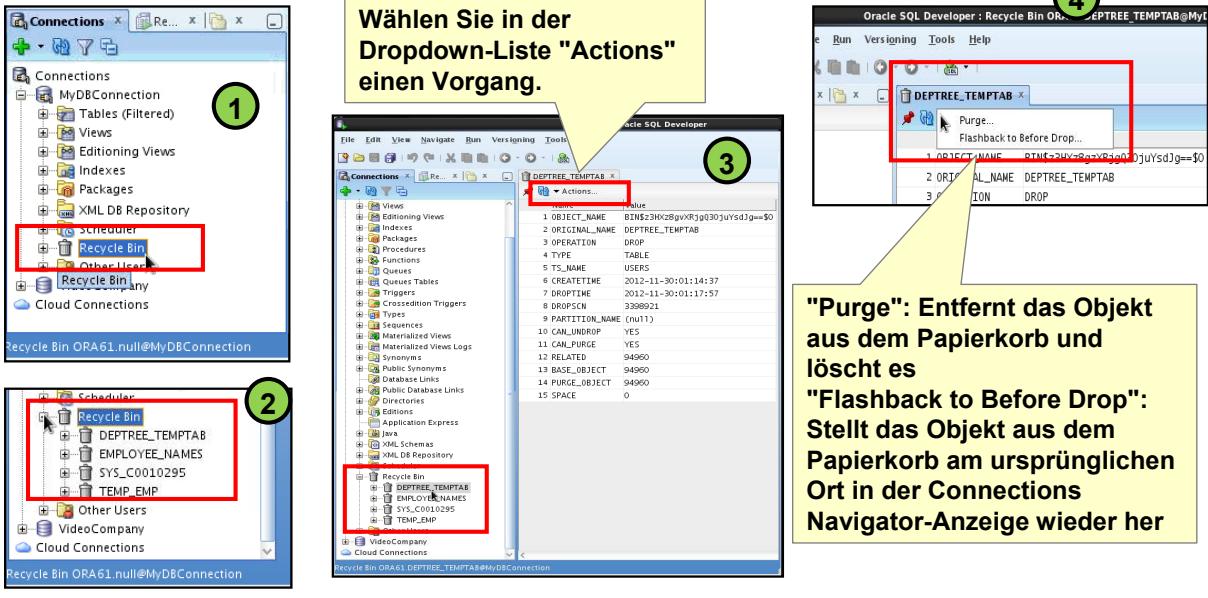
Um ein Snippet in den Code in einem SQL Worksheet oder in eine PL/SQL-Funktion oder -Prozedur einzufügen, ziehen Sie es aus dem Fenster **Snippets** an die gewünschte Stelle im Code. Danach können Sie die Syntax bearbeiten, sodass die SQL-Funktion im aktuellen Kontext gültig ist. Um eine kurze Beschreibung einer SQL-Funktion als QuickInfo anzeigen, platzieren Sie den Cursor über dem Funktionsnamen.

Im Beispiel auf der Folie wird `CONCAT(char1, char2)` aus der Gruppe **Character Functions** in das Fenster **Snippets** gezogen. Anschließend wird die Syntax der Funktion `CONCAT` bearbeitet und der restliche Teil der Anweisung wie folgt hinzugefügt:

```
SELECT CONCAT(first_name, last_name)
FROM employees;
```

Papierkorb

Der Papierkorb enthält gelöschte Objekte.



Copyright © 2013, Oracle und/oder verbundene Unternehmen. All rights reserved. Alle Rechte vorbehalten.

ORACLE

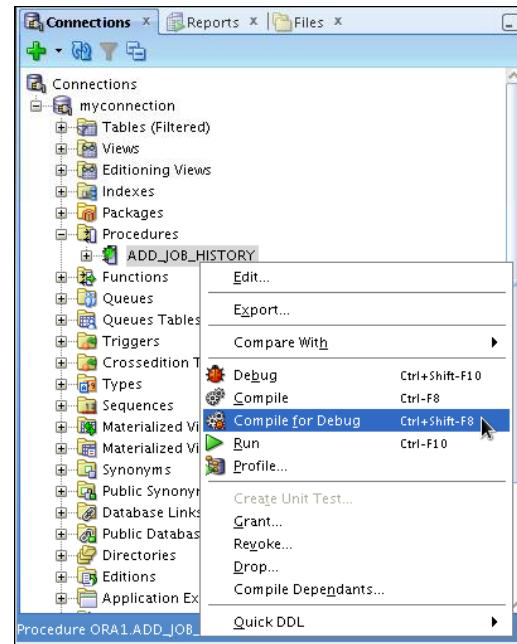
Der Papierkorb ist eine Data Dictionary-Tabelle, die Informationen zu gelöschten Objekten enthält. Gelöschte Tabellen und alle zugehörigen Objekte (wie Indizes, Constraints und Nested Tables) werden nicht entfernt und belegen weiterhin Speicherplatz. Sie werden so lange in den Speicherplatz-Quotas für Benutzer berücksichtigt, bis sie explizit dauerhaft aus dem Papierkorb gelöscht werden oder die unwahrscheinliche Situation eintritt, dass sie von der Datenbank aufgrund von Tablespace-Speicherplatzbeschränkungen dauerhaft gelöscht werden müssen.

Um den Papierkorb zu verwenden, gehen Sie wie folgt vor:

1. Wählen Sie im Connections Navigator den Papierkorb (oder navigieren Sie dorthin).
2. Erweitern Sie den Papierkorb, und klicken Sie auf den Objektnamen. Die Objektdetails werden im SQL Worksheet-Bereich angezeigt.
3. Wählen Sie in der Dropdown-Liste **Actions** den Vorgang, den Sie für das Objekt ausführen möchten.

Prozeduren und Funktionen debuggen

- **Mit SQL Developer PL/SQL-Funktionen und -Prozeduren debuggen**
- **Mit der Option "Compile for Debug" eine PL/SQL-Kompilierung ausführen, sodass Sie die Prozedur debuggen können**
- **Mit der Menüoption "Debug" Breakpoints setzen und die Schritte "Step into" und "Step over" ausführen**



ORACLE

Copyright © 2013, Oracle und/oder verbundene Unternehmen. All rights reserved. Alle Rechte vorbehalten.

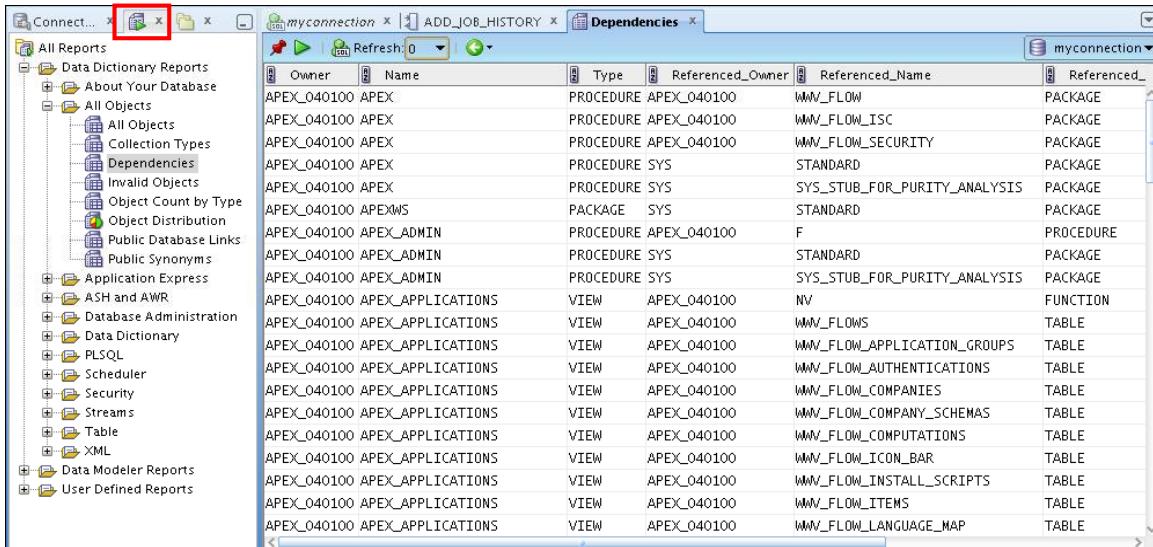
In SQL Developer können Sie PL/SQL-Prozeduren und -Funktionen debuggen. Mit den Menüoptionen zum Debuggen können Sie folgende Debugging-Aufgaben ausführen:

- **Find Execution Point** geht zum nächsten Ausführungspunkt.
- **Resume** setzt die Ausführung fort.
- **Step Over** umgeht die nächste Methode und geht zur nächsten Anweisung nach der Methode.
- **Step Into** geht zur ersten Anweisung in der nächsten Methode.
- **Step Out** verlässt die aktuelle Methode und geht zur nächsten Anweisung.
- **Step to End of Method** geht zur letzten Anweisung der aktuellen Methode.
- **Pause** unterbricht die Ausführung, beendet sie jedoch nicht. Sie können die Ausführung wieder aufnehmen.
- **Terminate** unterbricht die Ausführung und beendet sie. Sie können die Ausführung ab diesem Punkt nicht wieder aufnehmen. Um die Funktion oder Prozedur von Anfang an auszuführen oder zu debuggen, klicken Sie stattdessen in der Symbolleiste der Registerkarte **Source** auf das Symbol **Run** oder **Debug**.
- **Garbage Collection** entfernt ungültige Objekte aus dem Cache, um für gültige Objekte Platz zu machen, auf die häufiger zugegriffen wird.

Diese Optionen stehen auch als Symbole in der Symbolleiste **Debugging** zur Verfügung.

Datenbankberichte

SQL Developer bietet mehrere vordefinierte Berichte zur Datenbank und ihren Objekten.



The screenshot shows the Oracle SQL Developer interface. The left sidebar contains a tree view of report categories under 'All Reports'. The 'Reports' tab is highlighted with a red box. The main pane displays a table titled 'Dependencies' with columns: Owner, Name, Type, Referenced_Owner, Referenced_Name, and Referenced_Type. The table lists various database objects like procedures, views, and tables from the APEX_040100 schema, such as APEX_040100.APEX, APEX_040100.APEX, etc., and their dependencies on other objects like MMV_FLOW, MMV_FLOW_ISC, etc.

Owner	Name	Type	Referenced_Owner	Referenced_Name	Referenced_Type
APEX_040100	APEX	PROCEDURE	APEX_040100	MMV_FLOW	PACKAGE
APEX_040100	APEX	PROCEDURE	APEX_040100	MMV_FLOW_ISC	PACKAGE
APEX_040100	APEX	PROCEDURE	APEX_040100	MMV_FLOW_SECURITY	PACKAGE
APEX_040100	APEX	PROCEDURE	SYS	STANDARD	PACKAGE
APEX_040100	APEX	PROCEDURE	SYS	SYS_STUB_FOR_PURITY_ANALYSIS	PACKAGE
APEX_040100	APEXWS	PACKAGE	SYS	STANDARD	PACKAGE
APEX_040100	APEX_ADMIN	PROCEDURE	APEX_040100	F	PROCEDURE
APEX_040100	APEX_ADMIN	PROCEDURE	SYS	STANDARD	PACKAGE
APEX_040100	APEX_ADMIN	PROCEDURE	SYS	SYS_STUB_FOR_PURITY_ANALYSIS	PACKAGE
APEX_040100	APEX_APPLICATIONS	VIEW	APEX_040100	NV	FUNCTION
APEX_040100	APEX_APPLICATIONS	VIEW	APEX_040100	MMV_FLOWS	TABLE
APEX_040100	APEX_APPLICATIONS	VIEW	APEX_040100	MMV_FLOW_APPLICATION_GROUPS	TABLE
APEX_040100	APEX_APPLICATIONS	VIEW	APEX_040100	MMV_FLOW_AUTHENTICATIONS	TABLE
APEX_040100	APEX_APPLICATIONS	VIEW	APEX_040100	MMV_FLOW_COMPANIES	TABLE
APEX_040100	APEX_APPLICATIONS	VIEW	APEX_040100	MMV_FLOW_COMPANY_SCHEMAS	TABLE
APEX_040100	APEX_APPLICATIONS	VIEW	APEX_040100	MMV_FLOW_COMPUTATIONS	TABLE
APEX_040100	APEX_APPLICATIONS	VIEW	APEX_040100	MMV_FLOW_ICON_BAR	TABLE
APEX_040100	APEX_APPLICATIONS	VIEW	APEX_040100	MMV_FLOW_INSTALL_SCRIPTS	TABLE
APEX_040100	APEX_APPLICATIONS	VIEW	APEX_040100	MMV_FLOW_ITEMS	TABLE
APEX_040100	APEX_APPLICATIONS	VIEW	APEX_040100	MMV_FLOW_LANGUAGE_MAP	TABLE

ORACLE

Copyright © 2013, Oracle und/oder verbundene Unternehmen. All rights reserved. Alle Rechte vorbehalten.

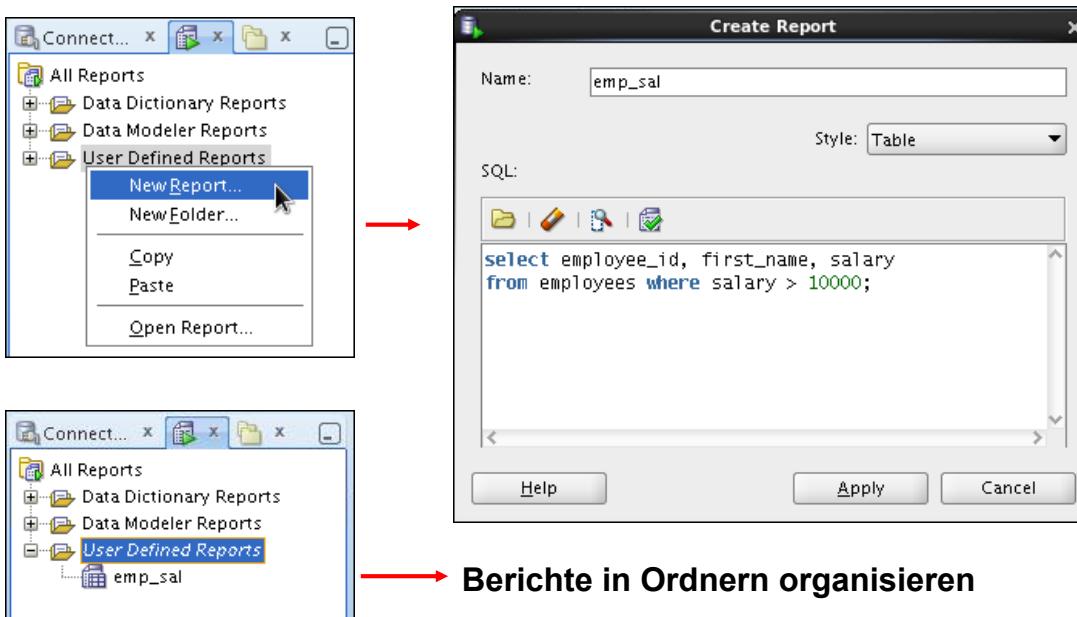
SQL Developer bietet zahlreiche Berichte zur Datenbank und ihren Objekten. Diese Berichte sind in folgende Kategorien unterteilt:

- About Your Database
- Database Administration
- Table
- PLSQL
- Security
- XML
- Jobs
- Streams
- All Objects
- Data Dictionary
- User-Defined Reports

Um Berichte anzuzeigen, klicken Sie links im Fenster auf die Registerkarte **Reports**. Die einzelnen Berichte werden rechts im Fenster in Tabbed Panes angezeigt. Bei jedem Bericht können Sie (in einer Dropdown-Liste) die Datenbankverbindung wählen, für die der Bericht angezeigt werden soll. Bei Berichten zu Objekten werden nur die Objekte angezeigt, die der Datenbankbenutzer mit der gewählten Datenbankverbindung sehen kann. Die Zeilen werden normalerweise nach dem Eigentümer (Owner) sortiert. Sie können auch eigene benutzerdefinierte Berichte erstellen.

Benutzerdefinierte Berichte erstellen

Benutzerdefinierte Berichte zur späteren Wieder-verwendung erstellen und speichern



ORACLE®

Copyright © 2013, Oracle und/oder verbundene Unternehmen. All rights reserved. Alle Rechte vorbehalten.

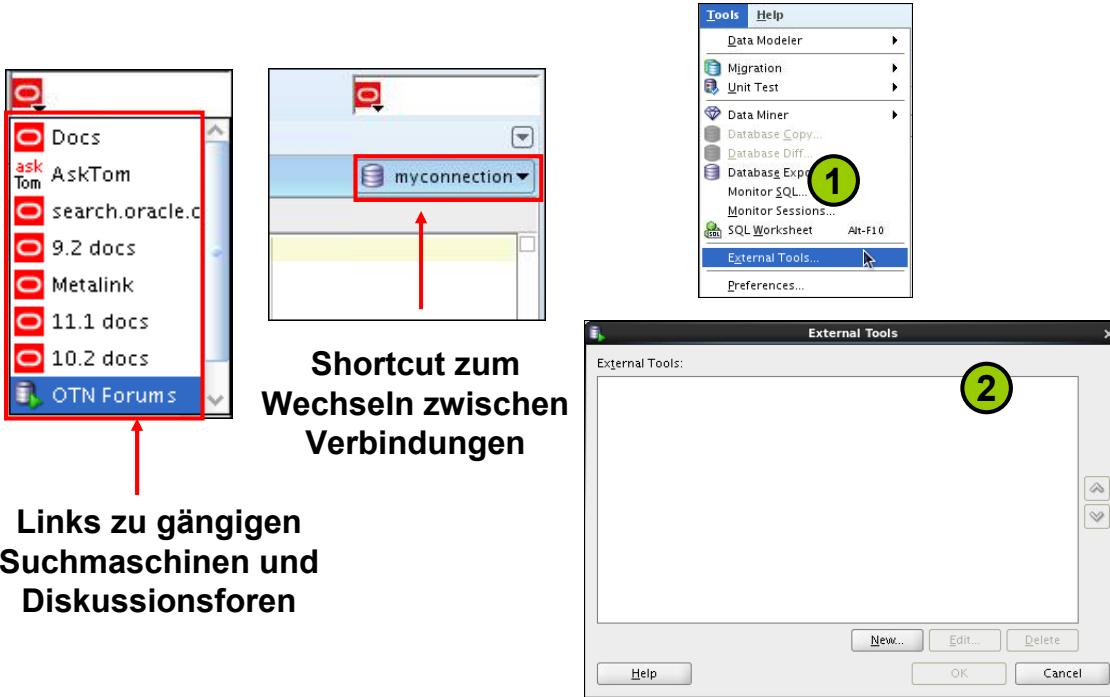
Benutzerdefinierte Berichte sind Berichte, die von SQL Developer-Benutzern erstellt werden. Um einen benutzerdefinierten Bericht zu erstellen, gehen Sie wie folgt vor:

1. Klicken Sie mit der rechten Maustaste unter **All Reports** auf den Knoten **User Defined Reports**, und wählen Sie **Add Report**.
2. Geben Sie im Dialogfeld **Create Report** den Berichtsnamen und die SQL-Abfrage an, um die Informationen für den Bericht abzurufen. Klicken Sie anschließend auf **Apply**.

Im Beispiel auf der Folie lautet der Name des Berichts `emp_sal`. Außerdem wurde eine optionale Beschreibung angegeben, aus der hervorgeht, dass der Bericht Einzelheiten zu Mitarbeitern mit einem Gehalt von mindestens 10.000 enthält (`salary >= 10000`). Die vollständige SQL-Anweisung zum Abruf der Informationen für den benutzerdefinierten Bericht ist im Feld **SQL** angegeben. Sie können auch eine optionale QuickInfo aufnehmen, die angezeigt wird, wenn der Cursor in der Navigatoranzeige für **Reports** über dem Berichtsnamen platziert wird.

Sie können benutzerdefinierte Berichte in Ordnern organisieren und eine Hierarchie von Ordnern und Unterordnern erstellen. Um einen Ordner für benutzerdefinierte Berichte zu erstellen, klicken Sie mit der rechten Maustaste auf den Knoten **User Defined Reports** oder einen beliebigen Ordnernamen unter diesem Knoten und wählen **Add Folder**. Informationen über benutzerdefinierte Berichte, einschließlich der Ordner für diese Berichte, werden im Verzeichnis für benutzerspezifische Informationen in der Datei `UserReports.xml` gespeichert.

Suchmaschinen und externe Tools



Copyright © 2013, Oracle und/oder verbundene Unternehmen. All rights reserved. Alle Rechte vorbehalten.

ORACLE

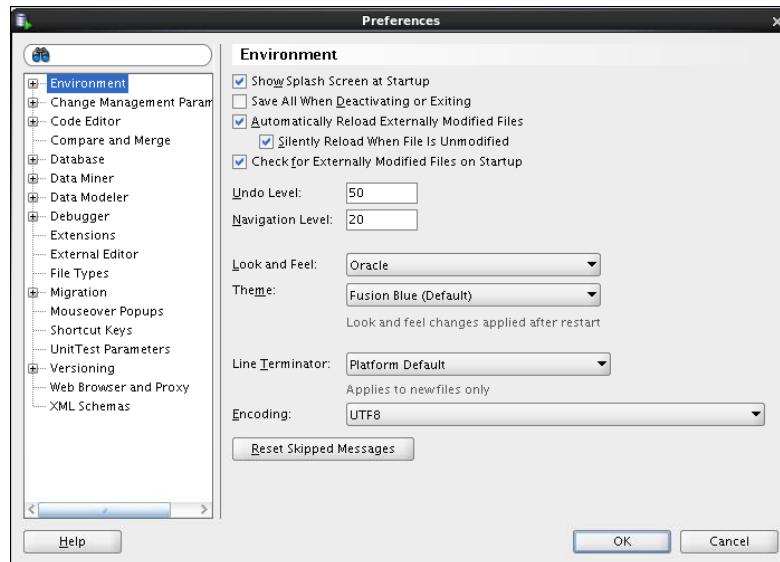
Um die Produktivität von SQL-Entwicklern zu erhöhen, wurde SQL Developer um Quicklinks zu gängigen Suchmaschinen und Diskussionsforen wie AskTom oder Google erweitert. Außerdem stehen Shortcut-Symbole zu häufig verwendeten Tools wie Notepad, Microsoft Word oder Dreamweaver zur Verfügung.

Sie können die vorhandene Liste um externe Tools ergänzen oder auch Shortcuts zu selten verwendeten Tools löschen. Gehen Sie dazu wie folgt vor:

1. Wählen Sie im Menü **Tools** die Option **External Tools**.
2. Um neue Tools hinzuzufügen, wählen Sie im Dialogfeld **External Tools** die Option **New**. Um Tools aus der Liste zu entfernen, wählen Sie **Delete**.

Voreinstellungen festlegen

- **Benutzeroberfläche und Umgebung von SQL Developer anpassen**
- **Im Menü "Tools" die Option "Preferences" wählen**



ORACLE

Copyright © 2013, Oracle und/oder verbundene Unternehmen. All rights reserved. Alle Rechte vorbehalten.

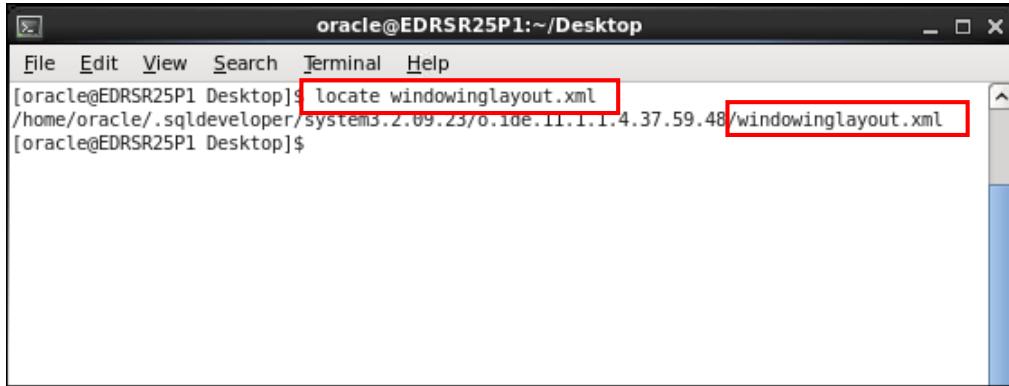
Sie können viele Aspekte der Benutzeroberfläche und der Umgebung von SQL Developer anpassen, indem Sie die SQL Developer-Voreinstellungen entsprechend Ihren Anforderungen ändern. Um SQL Developer-Voreinstellungen zu ändern, wählen Sie **Tools** und anschließend **Preferences**.

Die Voreinstellungen sind in folgende Kategorien eingeteilt:

- Environment
- Change Management Parameter
- Code Editor
- Compare and Merge
- Database
- Data Miner
- Data Modeler
- Debugger
- Extensions
- External Editor
- File Types
- Migration

- Mouseover Popups
- Shortcut Keys
- Unit Test Parameters
- Versioning
- Web Browser and Proxy
- XML Schemas

SQL Developer-Layout zurücksetzen



```
oracle@EDRSR25P1:~/Desktop
File Edit View Search Terminal Help
[oracle@EDRSR25P1 Desktop]$ locate windowinglayout.xml
/home/oracle/.sqldeveloper/system3.2.09.23.0.10e.11.1.1.4.37.59.48/windowinglayout.xml
[oracle@EDRSR25P1 Desktop]$
```

A screenshot of a terminal window titled "oracle@EDRSR25P1:~/Desktop". The window has a standard Linux-style title bar with menu options: File, Edit, View, Search, Terminal, and Help. The main pane displays the command "locate windowinglayout.xml" followed by its output: the path "/home/oracle/.sqldeveloper/system3.2.09.23.0.10e.11.1.1.4.37.59.48/windowinglayout.xml". The entire terminal window is enclosed in a red border.

ORACLE

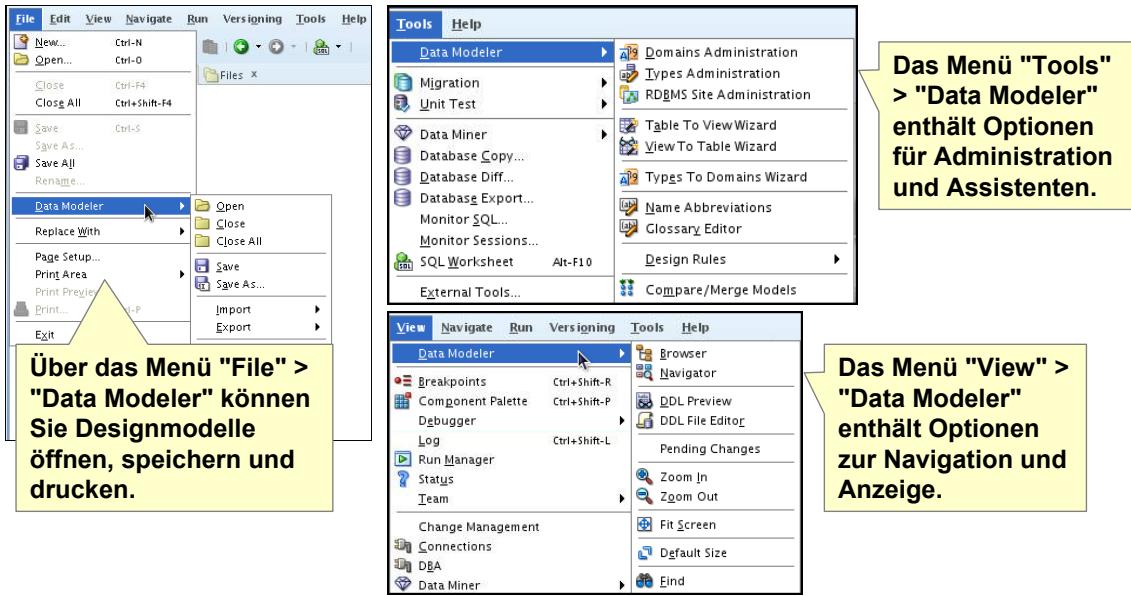
Copyright © 2013, Oracle und/oder verbundene Unternehmen. All rights reserved. Alle Rechte vorbehalten.

Wenn der Connections Navigator während der Arbeit mit SQL Developer ausgeblendet wird oder wenn Sie das Fenster **Log** nicht an seiner ursprünglichen Position verankern können, beheben Sie das Problem wie folgt:

1. Beenden Sie SQL Developer.
2. Öffnen Sie ein Terminalfenster, und verwenden Sie den Befehl `locate`, um den Speicherort von `windowinglayout.xml` zu ermitteln.
3. Navigieren Sie in das Verzeichnis, in dem sich `windowinglayout.xml` befindet, und löschen Sie diese Datei.
4. Starten Sie SQL Developer neu.

Data Modeler in SQL Developer

SQL Developer enthält eine integrierte Version von SQL Developer Data Modeler.



ORACLE

Copyright © 2013, Oracle und/oder verbundene Unternehmen. All rights reserved. Alle Rechte vorbehalten.

Mit der integrierten Version von SQL Developer Data Modeler können Sie:

- Datenbankdesigns erstellen, öffnen, importieren und speichern
- Data Modeler-Objekte erstellen, bearbeiten und löschen

Um Data Modeler in einem Bereich anzuzeigen, klicken Sie auf **Tools** und dann auf **Data Modeler**. Das Untermenü **Data Modeler** im Menü **Tools** enthält zusätzliche Befehle, etwa zur Festlegung von Designregeln und Voreinstellungen.

Zusammenfassung

In diesem Anhang haben Sie gelernt, mit SQL Developer folgende Aufgaben auszuführen:

- **Datenbankobjekte durchsuchen, erstellen und bearbeiten**
- **SQL-Anweisungen und -Skripte im SQL Worksheet ausführen**
- **Benutzerdefinierte Berichte erstellen und speichern**
- **Data Modeler-Optionen in SQL Developer durchsuchen**



Copyright © 2013, Oracle und/oder verbundene Unternehmen. All rights reserved. Alle Rechte vorbehalten.

Das kostenlose grafische Tool SQL Developer vereinfacht Aufgaben der Datenbankentwicklung. Mit SQL Developer können Sie Datenbankobjekte durchsuchen, erstellen und bearbeiten. Mit dem SQL Worksheet lassen sich SQL-Anweisungen und -Skripte ausführen. Darüber hinaus können Sie mit SQL Developer eigene spezielle Berichte erstellen und speichern, die Sie mehrfach wiederverwenden können.



ORACLE

Copyright © 2013, Oracle und/oder verbundene Unternehmen. All rights reserved. Alle Rechte vorbehalten.

Ziele

Nach Beendigung dieses Anhangs haben Sie folgende Ziele erreicht:

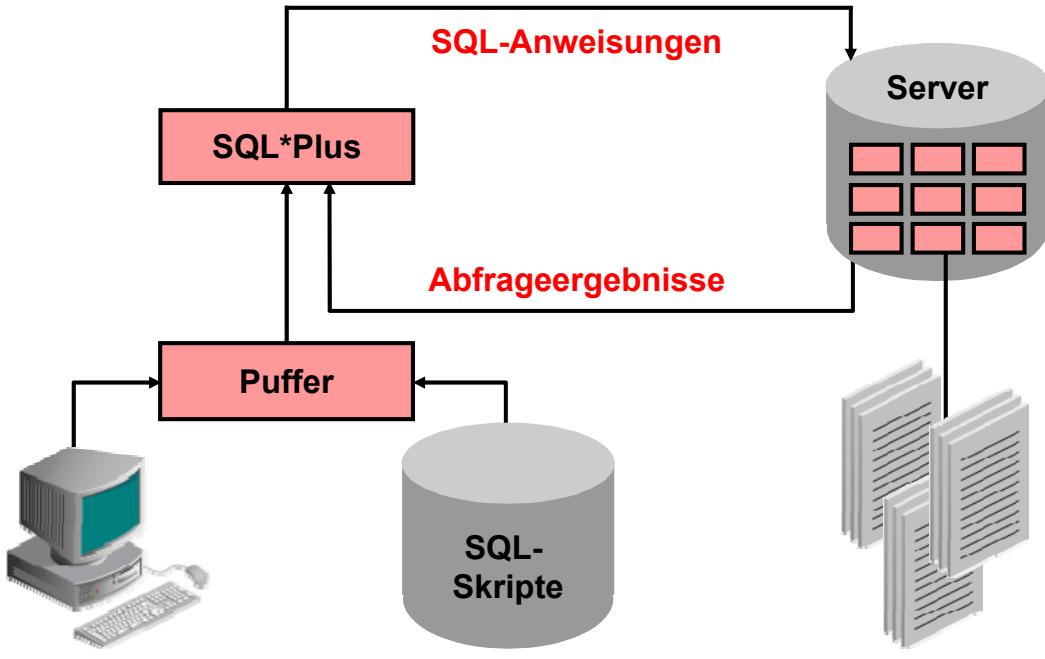
- **Bei SQL*Plus anmelden**
- **SQL-Befehle bearbeiten**
- **Ausgabe mithilfe von SQL*Plus-Befehlen formatieren**
- **Mit Skriptdateien interagieren**

ORACLE

Copyright © 2013, Oracle und/oder verbundene Unternehmen. All rights reserved. Alle Rechte vorbehalten.

Sie können SELECT-Anweisungen erstellen und mehrfach wiederverwenden. In diesem Anhang wird die Ausführung von SQL-Anweisungen mithilfe von SQL*Plus-Befehlen behandelt. Außerdem wird beschrieben, wie Sie Ausgaben mit SQL*Plus-Befehlen formatieren, SQL-Befehle bearbeiten und Skripte in SQL*Plus speichern.

SQL und SQL*Plus – Interaktion



ORACLE

Copyright © 2013, Oracle und/oder verbundene Unternehmen. All rights reserved. Alle Rechte vorbehalten.

SQL und SQL*Plus

SQL ist eine Befehlssprache, mit deren Hilfe beliebige Tools oder Anwendungen mit dem Oracle-Server kommunizieren können. Oracle SQL enthält zahlreiche Erweiterungen. Wenn Sie eine SQL-Anweisung eingeben, wird sie in einem Bereich des Speichers abgelegt, der als *SQL-Puffer* bezeichnet wird. Sie verbleibt dort, bis Sie eine neue SQL-Anweisung eingeben. SQL*Plus ist ein Oracle-Tool, das SQL-Anweisungen erkennt und zur Ausführung an den Oracle9i-Server weiterleitet. Es enthält eine eigene Befehlssprache.

SQL – Features

- Kann von allen Benutzern verwendet werden, auch von Benutzern ohne oder mit geringen Programmiererfahrungen
- Ist eine nicht prozedurale Sprache
- Reduziert den Zeitaufwand für Erstellung und Verwaltung von Systemen
- Ist an das Englische angelehnt

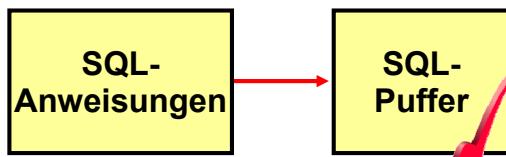
SQL*Plus – Features

- Akzeptiert die Ad-hoc-Eingabe von Anweisungen
- Akzeptiert die SQL-Eingabe aus Dateien
- Bietet einen Zeileneditor zum Ändern von SQL-Anweisungen
- Steuert die Umgebungseinstellungen
- Formatiert Abfrageergebnisse zu Basisberichten
- Greift auf lokale Datenbanken und Remote-Datenbanken zu

SQL-Anweisungen und SQL*Plus-Befehle – Vergleich

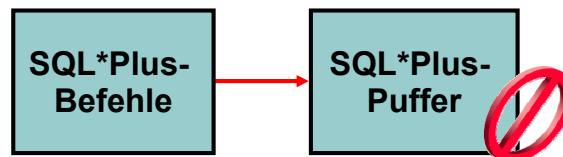
SQL

- **Eine Sprache**
- **ANSI-Standard**
- **Schlüsselwörter dürfen nicht abgekürzt werden.**
- **Anweisungen bearbeiten Daten und Tabellendefinitionen in der Datenbank.**



SQL*Plus

- **Eine Umgebung**
- **Oracle-Tool**
- **Schlüsselwörter können abgekürzt werden.**
- **Befehle erlauben keine Bearbeitung von Werten in der Datenbank.**



ORACLE

Copyright © 2013, Oracle und/oder verbundene Unternehmen. All rights reserved. Alle Rechte vorbehalten.

Die folgende Tabelle stellt SQL und SQL*Plus gegenüber:

SQL	SQL*Plus
Ist eine Sprache für die Kommunikation mit dem Oracle-Server beim Datenzugriff	Erkennt SQL-Anweisungen und sendet sie an den Server
Basiert auf Standard-SQL des American National Standards Institute (ANSI)	Ist die Oracle-eigene Schnittstelle zum Ausführen von SQL-Anweisungen
Bearbeitet Daten und Tabellendefinitionen in der Datenbank	Ermöglicht keine Bearbeitung von Werten in der Datenbank
Wird in einer oder mehreren Zeilen des SQL-Puffers eingegeben	Wird zeilenweise eingegeben und nicht im SQL-Puffer gespeichert
Hat kein Fortsetzungszeichen	Verwendet einen Bindestrich (-) als Fortsetzungszeichen, wenn der Befehl länger als eine Zeile ist
Kann nicht abgekürzt werden	Kann abgekürzt werden
Verwendet ein Abschlusszeichen, um Befehle unmittelbar auszuführen	Benötigt keine Abschlusszeichen. Befehle werden sofort ausgeführt
Verwendet Funktionen zur Ausführung von Formatierungen	Verwendet Befehle zum Formatieren von Daten

SQL*Plus – Überblick

- **Bei SQL*Plus anmelden**
- **Tabellenstruktur beschreiben**
- **SQL-Anweisungen bearbeiten**
- **SQL über SQL*Plus ausführen**
- **SQL-Anweisungen in Dateien speichern und an Dateien anhängen**
- **Gespeicherte Dateien ausführen**
- **Befehle aus der Datei zur Bearbeitung in den Puffer laden**

ORACLE

Copyright © 2013, Oracle und/oder verbundene Unternehmen. All rights reserved. Alle Rechte vorbehalten.

SQL*Plus

In der SQL*Plus-Umgebung können Sie folgende Aktionen ausführen:

- SQL-Anweisungen ausführen, um Daten aus der Datenbank abzurufen, zu ändern, hinzuzufügen und zu entfernen
- Abfrageergebnisse als Berichte formatieren, speichern und drucken sowie Berechnungen mit den Abfrageergebnissen ausführen
- Skriptdateien erstellen, um SQL-Anweisungen für die spätere Wiederverwendung zu speichern

SQL*Plus-Befehle können in folgende Hauptkategorien unterteilt werden:

Kategorie	Zweck
Umgebung	Allgemeines Verhalten von SQL-Anweisungen in der Session steuern
Formatierung	Abfrageergebnisse formatieren
Dateibearbeitung	Dateien speichern, laden und ausführen
Ausführung	SQL-Anweisungen vom SQL-Puffer an den Oracle-Server senden
Bearbeitung	SQL-Anweisungen im Puffer ändern
Interaktion	Variablen erstellen und an SQL-Anweisungen übergeben, Variablenwerte drucken und Meldungen auf dem Bildschirm ausgeben
Sonstiges	Mit der Datenbank verbinden, SQL*Plus-Umgebung bearbeiten und Spaltendefinitionen anzeigen

Bei SQL*Plus anmelden

```
oracle@EDRSR25P1:~/Desktop
[oracle@EDRSR25P1 Desktop]$ sqlplus
SQL*Plus: Release 12.1.0.0.2 Beta on Thu Sep 13 02:00:57 2012
Copyright (c) 1982, 2012, Oracle. All rights reserved.

Enter user-name: oral
Enter password: 1
Last Successful login time: Wed Sep 2012 23:16:13 +00:00

Connected to:
Oracle Database 12c Enterprise Edition Release 12.1.0.0.2 - 64bit Beta
With the Partitioning, OLAP, Data Mining and Real Application Testing options

SQL>
```

sqlplus [username[/password[@database]]]

```
oracle@EDRSR25P1:~/Desktop
[oracle@EDRSR25P1 Desktop]$ sqlplus oral/oral
SQL*Plus: Release 12.1.0.0.2 Beta on Thu Sep 13 02:29:51 2012
Copyright (c) 1982, 2012, Oracle. All rights reserved. 2
Last Successful login time: Thu Sep 2012 02:01:21 +00:00

Connected to:
Oracle Database 12c Enterprise Edition Release 12.1.0.0.2 - 64bit Beta
With the Partitioning, OLAP, Data Mining and Real Application Testing options

SQL>
```

ORACLE

Copyright © 2013, Oracle und/oder verbundene Unternehmen. All rights reserved. Alle Rechte vorbehalten.

Wie Sie SQL*Plus aufrufen, richtet sich nach dem Typ des Betriebssystems, auf dem Sie Oracle Database ausführen.

Um sich aus einer Linux-Umgebung anzumelden, gehen Sie wie folgt vor:

1. Klicken Sie mit der rechten Maustaste auf den Linux-Desktop, und wählen Sie **terminal**.
2. Geben Sie den auf der Folie gezeigten Befehl `sqlplus` ein.
3. Geben Sie Benutzername, Kennwort und Datenbanknamen ein.

Für die Syntax gilt:

- `username` Ihr Benutzername für die Datenbank
- `password` Ihr Kennwort für die Datenbank. (Das Kennwort ist hier bei der Eingabe sichtbar.)
- `@database` Die Verbindungszeichenfolge für die Datenbank

Hinweis: Um die Integrität des Kennwertes zu wahren, dürfen Sie es nicht in der Eingabeaufforderung des Betriebssystems eingeben. Geben Sie stattdessen nur Ihren Benutzernamen ein, und geben Sie das Kennwort in der Kennworteingabeaufforderung ein.

Tabellenstrukturen anzeigen

Struktur einer Tabelle mit dem SQL*Plus-Befehl **DESCRIBE** anzeigen:

```
DESC [RIBE] tablename
```

ORACLE

Copyright © 2013, Oracle und/oder verbundene Unternehmen. All rights reserved. Alle Rechte vorbehalten.

In SQL*Plus können Sie die Struktur einer Tabelle mit dem Befehl **DESCRIBE** anzeigen. Daraufhin werden die Spaltennamen und Datentypen sowie ein Hinweis darauf angezeigt, ob eine Spalte Daten enthalten muss.

Für die Syntax gilt:

tablename Der Name einer beliebigen vorhandenen Tabelle, einer View oder eines Synonyms, auf die bzw. das der Benutzer zugreifen kann

Die Tabelle DEPARTMENTS zeigen Sie mit folgendem Befehl an:

```
SQL> DESCRIBE DEPARTMENTS
      Name          Null    Type
----- -----
DEPARTMENT_ID      NOT NULL NUMBER(4)
DEPARTMENT_NAME    NOT NULL VARCHAR2(30)
MANAGER_ID          NUMBER(6)
LOCATION_ID         NUMBER(4)
```

Tabellenstrukturen anzeigen

```
DESCRIBE departments
```

Name	Null	Type
DEPARTMENT_ID	NOT NULL	NUMBER(4)
DEPARTMENT_NAME	NOT NULL	VARCHAR2(30)
MANAGER_ID		NUMBER(6)
LOCATION_ID		NUMBER(4)

ORACLE

Copyright © 2013, Oracle und/oder verbundene Unternehmen. All rights reserved. Alle Rechte vorbehalten.

Das Beispiel auf der Folie zeigt Informationen über die Struktur der Tabelle DEPARTMENTS. Für das Ergebnis gilt:

- Null: Gibt an, ob eine Spalte Daten enthalten muss. (NOT NULL zeigt an, dass eine Spalte Daten enthalten muss.)
- Type: Zeigt den Datentyp einer Spalte an

SQL*Plus – Bearbeitungsbefehle

- **A [PPEND] *text***
- **C [HANGE] / *old* / *new***
- **C [HANGE] / *text* /**
- **CL [EAR] BUFF [ER]**
- **DEL**
- **DEL *n***
- **DEL *m* *n***

ORACLE

Copyright © 2013, Oracle und/oder verbundene Unternehmen. All rights reserved. Alle Rechte vorbehalten.

SQL*Plus-Befehle werden Zeile für Zeile eingegeben und nicht im SQL-Puffer gespeichert.

Befehl	Beschreibung
A [PPEND] <i>text</i>	Fügt Text an das Ende der aktuellen Zeile an
C [HANGE] / <i>old</i> / <i>new</i>	Ändert in der aktuellen Zeile Text vom Typ <i>old</i> in <i>new</i>
C [HANGE] / <i>text</i> /	Löscht <i>text</i> aus der aktuellen Zeile
CL [EAR] BUFF [ER]	Löscht alle Zeilen aus dem SQL-Puffer
DEL	Löscht die aktuelle Zeile
DEL <i>n</i>	Löscht die Zeile <i>n</i>
DEL <i>m</i> <i>n</i>	Löscht die Zeilen <i>m</i> bis einschließlich <i>n</i>

Richtlinien

- Wenn Sie die EINGABETASTE drücken, bevor Sie einen Befehl abgeschlossen haben, zeigt SQL*Plus die Nummer der entsprechenden Befehlszeile an.
- Sie beenden den SQL-Puffer, indem Sie eines der Abschlusszeichen (Semikolon oder Schrägstrich) eingeben oder zweimal die EINGABETASTE drücken. Anschließend wird die SQL-Eingabeaufforderung angezeigt.

SQL*Plus – Bearbeitungsbefehle

- **I [NPUT]**
- **I [NPUT] *text***
- **L [IST]**
- **L [IST] *n***
- **L [IST] *m n***
- **R [UN]**
- ***n***
- ***n text***
- **0 *text***

ORACLE

Copyright © 2013, Oracle und/oder verbundene Unternehmen. All rights reserved. Alle Rechte vorbehalten.

Befehl	Beschreibung
I [NPUT]	Fügt eine ungebrenzte Anzahl an Zeilen ein
I [NPUT] <i>text</i>	Fügt eine aus <i>text</i> bestehende Zeile ein
L [IST]	Löscht alle Zeilen aus dem SQL-Puffer
L [IST] <i>n</i>	Listet eine Zeile (angegeben durch <i>n</i>)
L [IST] <i>m n</i>	Listet einen Bereich an Zeilen (<i>m</i> bis einschließlich <i>n</i>)
R [UN]	Zeigt die aktuelle SQL-Anweisung im Puffer an und führt sie aus
<i>n</i>	Gibt die aktuelle Zeile an
<i>n text</i>	Ersetzt line <i>n</i> durch <i>text</i>
0 <i>text</i>	Fügt eine Zeile vor die Zeile 1 ein

Hinweis: Sie können nur jeweils einen SQL*Plus-Befehl pro SQL-Eingabeaufforderung eingeben. SQL*Plus-Befehle werden nicht im SQL-Puffer gespeichert. Um einen SQL*Plus-Befehl in der nächsten Zeile fortzusetzen, geben Sie am Ende der ersten Zeile einen Bindestrich (-) ein.

LIST, n und APPEND

LIST

```
1  SELECT last_name
2* FROM   employees
```

1

```
1* SELECT last_name
```

A , job_id

```
1* SELECT last_name, job_id
```

LIST

```
1  SELECT last_name, job_id
2* FROM   employees
```

ORACLE

Copyright © 2013, Oracle und/oder verbundene Unternehmen. All rights reserved. Alle Rechte vorbehalten.

- Mit dem Befehl L[IST] zeigen Sie den Inhalt des SQL-Puffers an. Das Sternchen (*) neben der 2. Zeile im Puffer gibt an, dass die 2. Zeile die aktuelle Zeile ist. Alle eingegebenen Bearbeitungsbefehle gelten für die aktuelle Zeile.
- Sie können die aktuelle Zeile wechseln, indem Sie die Nummer (n) der Zeile eingeben, die Sie bearbeiten möchten. Die neue aktuelle Zeile wird angezeigt.
- Mit dem Befehl A[PPEND] fügen Sie der aktuellen Zeile Text hinzu. Die neu bearbeitete Zeile wird angezeigt. Den neuen Inhalt des Puffers prüfen Sie mit dem Befehl LIST.

Hinweis: Viele SQL*Plus-Befehle, einschließlich LIST und APPEND, können mit ihrem ersten Buchstaben abgekürzt werden. LIST lässt sich mit L abkürzen, APPEND mit A.

Befehl CHANGE

```
LIST
```

```
1* SELECT * from employees
```

```
c/employees/departments
```

```
1* SELECT * from departments
```

```
LIST
```

```
1* SELECT * from departments
```

ORACLE

Copyright © 2013, Oracle und/oder verbundene Unternehmen. All rights reserved. Alle Rechte vorbehalten.

- Mit `L[IST]` zeigen Sie den Inhalt des Puffers an.
- Mit dem Befehl `C[HANGE]` ändern Sie den Inhalt der aktuellen Zeile im SQL-Puffer. In diesem Fall ersetzen Sie die Tabelle `employees` durch die Tabelle `departments`. Die neue aktuelle Zeile wird angezeigt.
- Mit dem Befehl `L[IST]` prüfen Sie den neuen Inhalt des SQL-Puffers.

SQL*Plus – Dateibefehle

- **SAVE *filename***
- **GET *filename***
- **START *filename***
- **@ *filename***
- **EDIT *filename***
- **SPOOL *filename***
- **EXIT**

ORACLE

Copyright © 2013, Oracle und/oder verbundene Unternehmen. All rights reserved. Alle Rechte vorbehalten.

SQL-Anweisungen kommunizieren mit dem Oracle-Server. SQL*Plus-Befehle steuern die Umgebung, formatieren Abfrageergebnisse und verwalten Dateien. Sie können die in der folgenden Tabelle beschriebenen Befehle verwenden:

Befehl	Beschreibung
SAV[E] <i>filename</i> [.ext] [REP[LACE]APP[END]]	Speichert den aktuellen Inhalt des SQL-Puffers in eine Datei. Mit APPEND fügen Sie eine vorhandene Datei hinzu, mit REPLACE überschreiben Sie eine vorhandene Datei. Die Standarderweiterung ist .sql.
GET <i>filename</i> [.ext]	Speichert den Inhalt einer zuvor gespeicherten Datei in den SQL-Puffer. Die Standarderweiterung des Dateinamens ist .sql.
STA[RT] <i>filename</i> [.ext]	Führt eine zuvor gespeicherte Befehlsdatei aus
@ <i>filename</i>	Führt eine zuvor gespeicherte Befehlsdatei aus (wie START)
ED[IT]	Ruft den Editor auf und speichert den Pufferinhalt in die Datei afiedt.buf
ED[IT] [<i>filename</i> [.ext]]	Ruft den Editor auf, in dem der Inhalt einer gespeicherten Datei bearbeitet wird
SPO[OL] [<i>filename</i> [.ext]] OFF OUT	Speichert Abfrageergebnisse in eine Datei. OFF schließt die Spool-Datei. OUT schließt die Spool-Datei und sendet die Dateiergebnisse an den Drucker.
EXIT	Beendet SQL*Plus

Befehle **SAVE** und **START**

LIST

```
1  SELECT last_name, manager_id, department_id
2* FROM employees
```

SAVE my_query

Created file my_query

START my_query

LAST_NAME

MANAGER_ID DEPARTMENT_ID

King

90

Kochhar

100

90

...

107 rows selected.

ORACLE

Copyright © 2013, Oracle und/oder verbundene Unternehmen. All rights reserved. Alle Rechte vorbehalten.

SAVE

Mit dem Befehl **SAVE** speichern Sie den aktuellen Inhalt des Puffers in einer Datei. So können Sie häufig verwendete Skripte zur späteren Wiederverwendung speichern.

START

Mit dem Befehl **START** führen Sie ein Skript in SQL*Plus aus. Alternativ können Sie Skripte auch über das Symbol @ ausführen.

@my_query

Befehl SERVEROUTPUT

- **Mit dem Befehl SET SERVEROUT [PUT] steuern Sie, ob die Ausgabe von Stored Procedures oder PL/SQL-Blöcken in SQL*Plus angezeigt wird.**
- **Die Zeilenlängenbeschränkung für DBMS_OUTPUT wurde von 255 Byte auf 32767 Byte erhöht.**
- **Die Standardgröße ist nun unbegrenzt.**
- **Ist SERVEROUTPUT festgelegt, werden keine Ressourcen reserviert.**
- **Verwenden Sie UNLIMITED, da damit keine Performance-einbußen verbunden sind. Ausnahme: Sie möchten physischen Speicher einsparen.**

```
SET SERVEROUT[PUT] {ON | OFF} [SIZE {n | UNLIMITED}]
[FOR[MAT] {WRA[PPED] | WOR[D_WRAPPED] | TRU[NATED]}]
```



Copyright © 2013, Oracle und/oder verbundene Unternehmen. All rights reserved. Alle Rechte vorbehalten.

Die meisten PL/SQL-Programme führen Ein- und Ausgaben über SQL-Anweisungen durch, um Daten in Datenbanktabellen zu speichern oder diese Tabellen abzufragen. Alle anderen PL/SQL-Eingaben/Ausgaben erfolgen über APIs, die mit anderen Programmen interagieren. Beispiel: Das Package DBMS_OUTPUT verfügt über Prozeduren wie PUT_LINE. Um das Ergebnis außerhalb von PL/SQL anzuzeigen, ist zum Lesen und Anzeigen der an DBMS_OUTPUT übergebenen Daten ein anderes Programm (wie SQL*Plus) erforderlich.

SQL*Plus zeigt Daten von DBMS_OUTPUT nur an, wenn Sie zuvor den SQL*Plus-Befehl SET SERVEROUTPUT ON wie folgt abgesetzt haben:

```
SET SERVEROUTPUT ON
```

Hinweis

- SIZE legt die Anzahl der Ausgabebyte fest, die im Oracle Database-Server gepuffert werden können. Der Standardwert ist UNLIMITED. n darf nicht unter 2.000 oder über 1.000.000 liegen.
- Weitere Informationen zu SERVEROUTPUT finden Sie im Dokument *Oracle Database PL/SQL User's Guide and Reference 12c*.

SQL*Plus-Befehl SPOOL

```
SPO[OL] [file_name[.ext]] [CRE[APE] | REP[LACE] |  
APP[END]] | OFF | OUT]
```

Option	Beschreibung
file_name[.ext]	Schreibt die Ausgabe in die angegebene Datei
CRE[APE]	Erstellt eine neue Datei mit dem angegebenen Namen
REP[LACE]	Ersetzt den Inhalt einer vorhandenen Datei. Ist die Datei nicht vorhanden, wird sie durch REPLACE erstellt.
APP[END]	Fügt den Pufferinhalt am Ende der angegebenen Datei ein
OFF	Stoppt das Spool-Verfahren
OUT	Stoppt das Spool-Verfahren und sendet die Datei an den Standarddrucker des Rechners

ORACLE

Copyright © 2013, Oracle und/oder verbundene Unternehmen. All rights reserved. Alle Rechte vorbehalten.

Der Befehl SPOOL speichert die Abfrageergebnisse in einer Datei oder sendet die Datei optional an einen Drucker. Der Befehl SPOOL wurde erweitert, sodass Sie jetzt Daten an das Ende einer Datei anhängen oder eine vorhandene Datei ersetzen können. Bisher konnten Sie mit SPOOL lediglich Dateien erstellen (und ersetzen). REPLACE ist der Standardwert.

Sie können die über Befehle generierte Ausgabe mit SET TERMOUT OFF in ein Skript schreiben, ohne die Ausgabe auf dem Bildschirm anzuzeigen. SET TERMOUT OFF hat keine Auswirkungen auf die Ausgabe von Befehlen, die interaktiv ausgeführt werden.

Dateinamen, die Leerzeichen enthalten, müssen in Anführungszeichen gesetzt werden. Um mit SPOOL APPEND-Befehlen eine gültige HTML-Datei zu erstellen, müssen Sie mit PROMPT oder einem ähnlichen Befehl den Header und Footer der HTML-Seite erstellen. Der Befehl SPOOL APPEND parst keine HTML-Tags. Um die Parameter CREATE, APPEND und SAVE zu deaktivieren, stellen Sie SQLPLUSCOMPAT[IBILITY] auf 9.2 oder früher ein.

Befehl AUTOTRACE

- Zeigt nach der erfolgreichen Ausführung von SQL-DML-Anweisungen wie **SELECT**, **INSERT**, **UPDATE** oder **DELETE** einen Bericht an
- Der Bericht kann nun Ausführungsstatistiken und den Abfrageausführungspfad umfassen.

```
SET AUTOT[RACE] {ON | OFF | TRACE[ONLY]} [EXP[LAIN]]  
[STATISTICS]
```

```
SET AUTOTRACE ON  
-- The AUTOTRACE report includes both the optimizer  
-- execution path and the SQL statement execution  
-- statistics
```

ORACLE

Copyright © 2013, Oracle und/oder verbundene Unternehmen. All rights reserved. Alle Rechte vorbehalten.

EXPLAIN zeigt den Abfrageausführungspfad durch Ausführung von EXPLAIN PLAN an. STATISTICS zeigt Statistiken zu SQL-Anweisungen an. Die Formatierung des AUTOTRACE-Berichts kann je nach Version des Servers, bei dem Sie angemeldet sind, und Serverkonfiguration variieren. Das Package DBMS_XPLAN ermöglicht Ihnen die einfache Anzeige der Ausgabe des Befehls EXPLAIN PLAN in verschiedenen vordefinierten Formaten.

Hinweis

- Weitere Informationen zum Package und zu Unterprogrammen finden Sie im Dokument *Oracle Database PL/SQL Packages and Types Reference 12c*.
- Weitere Informationen zu EXPLAIN PLAN finden Sie im Dokument *Oracle Database SQL Reference 12c*.
- Weitere Informationen zu Ausführungsplänen und Statistiken finden Sie im *Oracle Database Performance Tuning Guide 12c*.

Zusammenfassung

In diesem Anhang haben Sie gelernt, SQL*Plus als Umgebung für folgende Aufgaben zu verwenden:

- **SQL-Anweisungen ausführen**
- **SQL-Anweisungen bearbeiten**
- **Ausgabe formatieren**
- **Mit Skriptdateien interagieren**



Copyright © 2013, Oracle und/oder verbundene Unternehmen. All rights reserved. Alle Rechte vorbehalten.

SQL*Plus ist eine Ausführungsumgebung, mit der Sie SQL-Befehle an den Datenbankserver senden sowie SQL-Befehle bearbeiten und speichern. Die Ausführung der Befehle erfolgt über die SQL-Eingabeaufforderung oder eine Skriptdatei.

D

REF-Cursor

ORACLE

Copyright © 2013, Oracle und/oder verbundene Unternehmen. All rights reserved. Alle Rechte vorbehalten.

Cursorvariablen

- **Cursorvariablen sind mit Zeigern in C und Pascal vergleichbar. Sie enthalten anstelle des eigentlichen Elements die Speicheradresse des Elements.**
- **In PL/SQL werden Zeiger als REF X deklariert, wobei REF die Abkürzung von REFERENCE ist und x für eine Klasse von Objekten steht.**
- **Eine Cursorvariable hat den Datentyp REF CURSOR.**
- **Cursor sind statisch, Cursorvariablen dagegen dynamisch.**
- **Cursorvariablen bieten eine größere Flexibilität.**

ORACLE

Copyright © 2013, Oracle und/oder verbundene Unternehmen. All rights reserved. Alle Rechte vorbehalten.

Cursorvariablen sind mit Zeigern in C und Pascal vergleichbar. Sie enthalten anstelle des eigentlichen Elements die Speicheradresse des Elements. Daher wird bei Deklarierung einer Cursorvariablen kein Element, sondern ein Zeiger erstellt. In PL/SQL haben Zeiger den Datentyp REF X, wobei REF die Abkürzung von REFERENCE ist und x für eine Klasse von Objekten steht. Eine Cursorvariable hat den Datentyp REF CURSOR.

Wie Cursor zeigen auch Cursorvariablen auf die aktuelle Zeile in der Ergebnismenge einer Multiple Row-Abfrage. Cursor unterscheiden sich jedoch von Cursorvariablen in gleicher Weise wie Konstanten von Variablen. Cursor sind statisch, Cursorvariablen dagegen dynamisch, da sie an keine bestimmte Abfrage gebunden sind. Sie können Cursorvariablen für jede mit dem Typ kompatible Abfrage öffnen und sind somit flexibler.

Cursorvariablen sind für jeden PL/SQL-Client verfügbar. Sie können eine Cursorvariable beispielsweise in einer PL/SQL-Hostumgebung wie einem OCI- oder Pro*C-Programm deklarieren und dann als Eingabehostvariable (Bind-Variable) an PL/SQL übergeben. Darüber hinaus können Tools zur Anwendungsentwicklung (z. B. Oracle Forms und Oracle Reports), die eine PL/SQL-Engine enthalten, Cursorvariablen vollständig auf der Clientseite verwenden. Der Oracle-Server verfügt ebenfalls über eine PL/SQL-Engine. Mithilfe von Remote-Prozeduraufrufen (RPCs) können Sie Cursorvariablen zwischen Anwendungen und Servern übergeben.

Cursorvariablen – Verwendung

- **Mithilfe von Cursorvariablen können Sie Ergebnismengen von Abfragen zwischen gespeicherten PL/SQL-Unterprogrammen und verschiedenen Clients übergeben.**
- **PL/SQL kann Zeiger auf Arbeitsbereiche, in denen die Ergebnismengen von Abfragen gespeichert sind, gemeinsam nutzen.**
- **Sie können den Wert einer Cursorvariablen nach Belieben von einem Gültigkeitsbereich an einen anderen übergeben.**
- **Zur Reduzierung des Netzwerkverkehrs können einzelne PL/SQL-Blöcke mehrere Hostcursorvariablen in einem Roundtrip öffnen oder schließen.**

ORACLE

Copyright © 2013, Oracle und/oder verbundene Unternehmen. All rights reserved. Alle Rechte vorbehalten.

Mithilfe von Cursorvariablen können Sie Abfrageergebnisse zwischen gespeicherten PL/SQL-Unterprogrammen und verschiedenen Clients übergeben. Weder PL/SQL noch die zugehörigen Clients sind Eigentümer der Ergebnismenge. Sie teilen sich lediglich einen Zeiger auf den Arbeitsbereich der Abfrage, in dem die Ergebnismenge gespeichert ist. Beispielsweise können ein OCI-Client, eine Oracle Forms-Anwendung und der Oracle-Server auf denselben Arbeitsbereich verweisen.

Auf den Arbeitsbereich einer Abfrage kann so lange zugegriffen werden, wie eine Cursorvariable auf ihn zeigt. Sie können daher den Wert einer Cursorvariablen nach Belieben aus einem Gültigkeitsbereich an einen anderen übergeben. Wenn Sie z. B. eine Hostcursorvariable an einen PL/SQL-Block übergeben, der in ein Pro*C-Programm eingebettet ist, können Sie nach Abschluss des Blocks weiterhin auf den Arbeitsbereich zugreifen, auf den die Cursorvariable zeigt.

Wenn auf der Clientseite eine PL/SQL-Engine vorhanden ist, gelten bei Aufrufen vom Client an den Server keine Einschränkungen. Sie können beispielsweise eine Cursorvariable auf der Clientseite deklarieren, sie auf der Serverseite öffnen und lesen und den Lesevorgang auf der Clientseite fortsetzen. Zur Reduzierung des Netzwerkverkehrs können einzelne PL/SQL-Blöcke auch mehrere Hostcursorvariablen in einem Roundtrip öffnen oder schließen.

Cursorvariablen adressieren die PGA (Program Global Area) nicht mit einem statischen Namen, sondern enthalten Verweise auf Cursorarbeitsbereiche in der PGA. Durch die Verwendung von Verweisen können Sie flexible Variablen einsetzen.

REF CURSOR-Typen definieren

REF CURSOR-Typ definieren:

```
Define a REF CURSOR type
TYPE ref_type_name IS REF CURSOR [RETURN return_type];
```

Cursorvariable mit diesem Typ deklarieren:

```
ref_cv ref_type_name;
```

Beispiel:

```
DECLARE
  TYPE DeptCurTyp IS REF CURSOR RETURN
    departments%ROWTYPE;
  dept_cv DeptCurTyp;
```

ORACLE

Copyright © 2013, Oracle und/oder verbundene Unternehmen. All rights reserved. Alle Rechte vorbehalten.

Um einen REF CURSOR zu definieren, führen Sie zwei Schritte aus. Zuerst definieren Sie einen REF CURSOR-Typ und deklarieren anschließend Cursorvariablen dieses Typs. Sie können REF CURSOR-Typen in beliebigen PL/SQL-Blöcken, -Unterprogrammen oder -Packages deklarieren.
Syntax:

```
TYPE ref_type_name IS REF CURSOR [RETURN return_type];
```

Dabei gilt:

- | | |
|---------------|--|
| ref_type_name | Ist die Typspezifikation, die bei folgenden Deklarationen von Cursorvariablen verwendet wird |
| return_type | Ist ein Record oder eine Zeile in einer Datenbanktabelle |

In diesem Beispiel geben Sie einen Rückgabetyp an, der einer Zeile in der Datenbanktabelle DEPARTMENT entspricht.

Es gibt starke (restriktive) und schwache (nicht restriktive) REF CURSOR-Typen. Wie im nächsten Beispiel ersichtlich, gibt eine starke REF CURSOR-Typdefinition einen Rückgabetyp an, eine schwache Definition dagegen nicht:

DECLARE

```
TYPE EmpCurTyp IS REF CURSOR RETURN employees%ROWTYPE; -- strong
TYPE GenericCurTyp IS REF CURSOR; -- weak
```

Starke REF CURSOR-Typen sind weniger fehleranfällig, da der PL/SQL-Compiler vorgibt, dass Sie Cursorvariablen mit starken Typdefinitionen nur Abfragen zuordnen können, die mit diesem Typ kompatibel sind. Schwache REF CURSOR-Typen sind jedoch flexibler, da Sie Cursorvariablen mit schwachen Typdefinitionen beliebigen Abfragen zuordnen können.

Cursorvariablen deklarieren

Nachdem Sie einen REF CURSOR-Typ deklariert haben, können Sie Cursorvariablen dieses Typs in allen PL/SQL-Blöcken oder -Unterprogrammen deklarieren. Im folgenden Beispiel deklarieren Sie die Cursorvariable DEPT_CV:

```
DECLARE
    TYPE DeptCurTyp IS REF CURSOR RETURN departments%ROWTYPE;
    dept_cv DeptCurTyp; -- declare cursor variable
```

Hinweis: Sie können Cursorvariablen nicht in einem Package deklarieren. Im Gegensatz zu Variablen, die in einem Package integriert sind, haben Cursorvariablen keinen persistenten Status. Wie bereits erwähnt, wird bei der Deklarierung einer Cursorvariablen kein Element, sondern ein Zeiger erstellt. Cursorvariablen können nicht in der Datenbank gespeichert werden. Sie richten sich nach den üblichen Richtlinien für Gültigkeitsbereiche und Instanziierung.

In der Klausel RETURN einer REF CURSOR-Typdefinition können Sie wie folgt mit %ROWTYPE einen Record-Typ angeben, der für eine Zeile steht, die von einer Cursorvariablen mit starker (nicht schwacher) Typdefinition zurückgegeben wird:

```
DECLARE
    TYPE TmpCurTyp IS REF CURSOR RETURN employees%ROWTYPE;
    tmp_cv TmpCurTyp; -- declare cursor variable
    TYPE EmpCurTyp IS REF CURSOR RETURN tmp_cv%ROWTYPE;
    emp_cv EmpCurTyp; -- declare cursor variable
```

Analog dazu können Sie %TYPE verwenden, um wie im folgenden Beispiel den Datentyp einer Record-Variablen anzugeben:

```
DECLARE
    dept_rec departments%ROWTYPE; -- declare record variable
    TYPE DeptCurTyp IS REF CURSOR RETURN dept_rec%TYPE;
    dept_cv DeptCurTyp; -- declare cursor variable
```

Im letzten Beispiel geben Sie in der Klausel RETURN einen benutzerdefinierten Typ RECORD an:

```
DECLARE
    TYPE EmpRecTyp IS RECORD (
        empno NUMBER(4),
        ename VARCHAR2(10),
        sal    NUMBER(7,2));
    TYPE EmpCurTyp IS REF CURSOR RETURN EmpRecTyp;
    emp_cv EmpCurTyp; -- declare cursor variable
```

Cursorvariablen als Parameter

Sie können Cursorvariablen als formale Parameter von Funktionen und Prozeduren deklarieren. Im folgenden Beispiel definieren Sie den REF CURSOR-Typ EmpCurTyp. Anschließend deklarieren Sie eine Cursorvariable dieses Typs als formalen Parameter einer Prozedur:

```
DECLARE
```

```
    TYPE EmpCurTyp IS REF CURSOR RETURN emp%ROWTYPE;  
    PROCEDURE open_emp_cv (emp_cv IN OUT EmpCurTyp) IS ...
```

Anweisungen vom Typ OPEN-FOR, FETCH und CLOSE

- Die Anweisung **OPEN-FOR** ordnet einer Multiple Row-Abfrage eine Cursorvariable zu, führt die Abfrage aus, identifiziert die Ergebnismenge und positioniert den Cursor so, dass er auf die erste Zeile in der Ergebnismenge zeigt.
- Die Anweisung **FETCH** gibt eine Zeile aus der Ergebnismenge einer Multiple Row-Abfrage zurück, weist die Werte der SELECT-Listenelemente den entsprechenden Variablen oder Feldern in der Klausel **INTO** zu, erhöht den von **%ROWCOUNT** verwalteten Zähler und rückt den Cursor in die nächste Zeile vor.
- Die Anweisung **CLOSE** deaktiviert eine Cursorvariable.

ORACLE

Copyright © 2013, Oracle und/oder verbundene Unternehmen. All rights reserved. Alle Rechte vorbehalten.

Dynamische Multiple Row-Abfragen verarbeiten Sie mit drei Anweisungen: OPEN-FOR, FETCH und CLOSE. Zunächst "öffnen" (OPEN) Sie eine Cursorvariable "für" (FOR) eine Multiple Row-Abfrage. Anschließend "lesen" (FETCH) Sie die Zeilen nacheinander aus der Ergebnismenge. Wenn alle Zeilen verarbeitet sind, "schließen" (CLOSE) Sie die Cursorvariable.

Cursorvariablen öffnen

Die Anweisung OPEN-FOR ordnet einer Multiple Row-Abfrage eine Cursorvariable zu, führt die Abfrage aus, identifiziert die Ergebnismenge, positioniert den Cursor so, dass er auf die erste Zeile in der Ergebnismenge zeigt und stellt den von %ROWCOUNT verwalteten Zähler der verarbeiteten Zeilen auf Null zurück. Im Gegensatz zum statischen Format von OPEN-FOR enthält das dynamische Format eine optionale USING-Klausel. Zur Laufzeit ersetzen Bind-Argumente in der Klausel USING entsprechende Platzhalter in der dynamischen Anweisung SELECT. Syntax:

```
OPEN {cursor_variable | :host_cursor_variable} FOR
dynamic_string
[USING bind_argument[, bind_argument]...];
```

Dabei gilt: CURSOR_VARIABLE ist eine Cursorvariable mit schwacher Typdefinition (ohne Rückgabetyp), HOST_CURSOR_VARIABLE eine in einer PL/SQL-Hostumgebung wie einem OCI-Programm deklarierte Cursorvariable und dynamic_string ein Zeichenfolgenausdruck, der eine Multiple Row-Abfrage repräsentiert.

Die Syntax im folgenden Beispiel deklariert eine Cursorvariable, die anschließend einer dynamischen SELECT-Anweisung zugeordnet wird, die Zeilen aus der Tabelle EMPLOYEES zurückgibt:

```

DECLARE
    TYPE EmpCurTyp IS REF CURSOR;      -- define weak REF CURSOR          type
    emp_cv   EmpCurTyp;   -- declare cursor variable
    my_ename VARCHAR2(15);
    my_sal   NUMBER := 1000;
BEGIN
    OPEN emp_cv FOR -- open cursor variable
        'SELECT last_name, salary FROM employees WHERE salary >
        :s'
        USING my_sal;
    ...
END;

```

Bind-Argumente in der Abfrage werden nur ausgewertet, wenn die Cursorvariable geöffnet ist. Um also Zeilen mithilfe verschiedener Bind-Werte aus dem Cursor zu lesen, müssen Sie die Cursorvariable erneut öffnen, nachdem die Bind-Argumente auf neue Werte festgelegt wurden.

Daten aus einer Cursorvariablen abrufen

Die Anweisung `FETCH` gibt eine Zeile aus der Ergebnismenge einer Multiple Row-Abfrage zurück, weist die Werte der SELECT-Listenelemente den entsprechenden Variablen oder Feldern in der Klausel `INTO` zu, erhöht den von `%ROWCOUNT` verwalteten Zähler und rückt den Cursor in die nächste Zeile vor. Syntax:

```

FETCH {cursor_variable | :host_cursor_variable}
    INTO {define_variable[, define_variable]... | record};

```

Zur Fortführung des Beispiels lesen Sie Zeilen aus der Cursorvariablen `emp_cv` in die Variablen `MY_ENAME` und `MY_SAL` ein:

```

LOOP
    FETCH emp_cv INTO my_ename, my_sal; -- fetch next row
    EXIT WHEN emp_cv%NOTFOUND; -- exit loop when last row is      fetched
    -- process row
END LOOP;

```

Zu jedem Spaltenwert, der bei der Abfrage mit der Cursorvariable zurückgegeben wird, muss eine entsprechende, typkompatible Variable oder ein entsprechendes Feld in der Klausel `INTO` vorhanden sein. Für verschiedene `FETCH`-Anweisungen mit derselben Cursorvariablen können Sie jeweils eine andere `INTO`-Klausel verwenden. Jeder Fetch-Vorgang ruft eine andere Zeile aus der Ergebnismenge ab. Wenn Sie versuchen, aus einer geschlossenen oder noch nie geöffneten Cursorvariablen zu lesen, löst PL/SQL die vordefinierte Exception `INVALID_CURSOR` aus.

Cursorvariablen schließen

Die Anweisung CLOSE deaktiviert eine Cursorvariable. Die zugehörige Ergebnismenge ist dann undefiniert. Syntax:

```
CLOSE {cursor_variable | :host_cursor_variable};
```

In diesem Beispiel soll nach Verarbeitung der letzten Zeile die Cursorvariable emp_cv geschlossen werden:

```
LOOP
  FETCH emp_cv INTO my_ename, my_sal;
  EXIT WHEN emp_cv%NOTFOUND;
  -- process row
END LOOP;
CLOSE emp_cv;  -- close cursor variable
```

Wenn Sie versuchen, eine bereits geschlossene oder noch nie geöffnete Cursorvariable zu schließen, löst PL/SQL INVALID_CURSOR aus.

Fetch-Vorgänge – Beispiel

```
DECLARE
    TYPE EmpCurTyp IS REF CURSOR;
    emp_cv    EmpCurTyp;
    emp_rec   employees%ROWTYPE;
    sql_stmt  VARCHAR2(200);
    my_job    VARCHAR2(10) := 'ST_CLERK';
BEGIN
    sql_stmt := 'SELECT * FROM employees
                 WHERE job_id = :j';
    OPEN emp_cv FOR sql_stmt USING my_job;
    LOOP
        FETCH emp_cv INTO emp_rec;
        EXIT WHEN emp_cv%NOTFOUND;
        -- process record
    END LOOP;
    CLOSE emp_cv;
END;
/
```

ORACLE

Copyright © 2013, Oracle und/oder verbundene Unternehmen. All rights reserved. Alle Rechte vorbehalten.

Wie im Beispiel auf der Folie gezeigt, können Sie Zeilen aus der Ergebnismenge einer dynamischen Multiple Row-Abfrage in einen Record einlesen. Zunächst müssen Sie den REF CURSOR-Typ EmpCurTyp definieren. Anschließend definieren Sie eine Cursorvariable emp_cv vom Typ EmpcurTyp. Im ausführbaren Bereich des PL/SQL-Blocks ordnet die Anweisung OPEN-FOR die Cursorvariable emp_cv der Multiple Row-Abfrage sql_stmt zu. Die Anweisung FETCH gibt eine Zeile aus der Ergebnismenge einer Multiple Row-Abfrage zurück und ordnet die Werte der SELECT-Listenelemente EMP_REC in der Klausel INTO zu. Nach Verarbeitung der letzten Zeile wird die Cursorvariable emp_cv geschlossen.

Häufig verwendete SQL-Befehle

ORACLE

Copyright © 2013, Oracle und/oder verbundene Unternehmen. All rights reserved. Alle Rechte vorbehalten.

Ziele

Nach Beendigung dieses Anhangs haben Sie folgende Ziele erreicht:

- **Einfache SELECT-Anweisungen ausführen**
- **Tabellen mithilfe von DDL-Anweisungen erstellen, ändern und löschen**
- **Zeilen aus einzelnen oder mehreren Tabellen mithilfe von DML-Anweisungen einfügen, aktualisieren und löschen**
- **Savepoints mithilfe von Anweisungen zur Transaktionskontrolle festschreiben, zurücksetzen und erstellen**
- **Join-Vorgänge für einzelne oder mehrere Tabellen durchführen**



Copyright © 2013, Oracle und/oder verbundene Unternehmen. All rights reserved. Alle Rechte vorbehalten.

In dieser Lektion erfahren Sie, wie Sie mithilfe von SELECT-Anweisungen Daten aus einzelnen oder mehreren Tabellen abrufen, die Struktur von Datenobjekten mit DDL-Anweisungen ändern, Daten in vorhandenen Schemaobjekten mit DML-Anweisungen bearbeiten und die über DML-Anweisungen vorgenommenen Änderungen verwalten sowie Daten aus mehreren Tabellen mithilfe von Joins und der Syntax für SQL:1999 anzeigen.

Einfache SELECT-Anweisungen

- **Mit SELECT-Anweisungen können Sie:**
 - die anzuzeigenden Spalten bestimmen
 - Daten aus einzelnen oder mehreren Tabellen, Objekttabellen, Views, Objekt-Views oder Materialized Views abrufen
- **SELECT-Anweisungen werden auch als Abfragen bezeichnet, da sie eine Datenbank abfragen.**
- **Syntax:**

```
SELECT { * | [DISTINCT] column|expression [alias],... }
      FROM table;
```

ORACLE

Copyright © 2013, Oracle und/oder verbundene Unternehmen. All rights reserved. Alle Rechte vorbehalten.

Eine SELECT-Anweisung muss in ihrer einfachsten Form folgende Elemente enthalten:

- Eine SELECT-Klausel, die bestimmt, welche Spalten angezeigt werden.
- Eine FROM-Klausel zur Angabe der Tabelle mit den in der Klausel SELECT aufgeführten Spalten

Für die Syntax gilt:

SELECT	Ist eine Liste aus einzelnen oder mehreren Spalten
*	Wählt alle Spalten
DISTINCT	Unterdrückt doppelte Werte
column / expression	Wählt die angegebene Spalte oder den Ausdruck
alias	Gibt den gewählten Spalten eine andere Überschrift
FROM table	Gibt an, welche Tabelle die Spalten enthält

Hinweis: Im gesamten Kurs werden die Begriffe *Schlüsselwort*, *Klausel* und *Anweisung* folgendermaßen verwendet:

- Ein *Schlüsselwort* bezieht sich auf ein einzelnes SQL-Element. Beispiele für Schlüsselwörter: SELECT und FROM
- Eine *Klausel* ist ein Teil einer SQL-Anweisung. Beispiel: SELECT employee_id, last_name
- Eine *Anweisung* ist eine Kombination aus zwei oder mehr Klauseln. Beispiel: SELECT * FROM employees

SELECT-Anweisungen

- Alle Spalten wählen:

```
SELECT *
FROM job_history;
```

	EMPLOYEE_ID	START_DATE	END_DATE	JOB_ID	DEPARTMENT_ID
1	102	13-JAN-01	24-JUL-06	IT_PROG	60
2	101	21-SEP-97	27-OCT-01	AC_ACCOUNT	110
3	101	28-OCT-01	15-MAR-05	AC_MGR	110
4	201	17-FEB-04	19-DEC-07	MK_REP	20
5	114	24-MAR-06	31-DEC-07	ST_CLERK	50
6	122	01-JAN-07	31-DEC-07	ST_CLERK	50
7	200	17-SEP-95	17-JUN-01	AD_ASST	90
8	176	24-MAR-06	31-DEC-06	SA REP	80
9	176	01-JAN-07	31-DEC-07	SA_MAN	80
10	200	01-JUL-02	31-DEC-06	AC_ACCOUNT	90

- Bestimmte Spalten wählen:

```
SELECT manager_id, job_id
FROM employees;
```

	MANAGER_ID	JOB_ID
1	(null)	AD_PRES
2		100 AD_VP
3		100 AD_VP
4		102 IT_PROG
5		103 IT_PROG
6		103 IT_PROG
7		100 ST_MAN
8		124 ST_CLERK
9		124 ST_CLERK
10		124 ST_CLERK

...

ORACLE

Copyright © 2013, Oracle und/oder verbundene Unternehmen. All rights reserved. Alle Rechte vorbehalten.

Um alle Datenspalten einer Tabelle anzuzeigen, können Sie nach dem Schlüsselwort SELECT ein Sternchen (*) angeben oder die Namen aller Spalten auflisten. Im ersten Beispiel auf der Folie werden alle Zeilen der Tabelle job_history angezeigt. Bestimmte Spalten der Tabelle zeigen Sie an, indem Sie die entsprechenden Spaltennamen durch Kommas getrennt angeben. Im zweiten Beispiel auf der Folie werden die Spalten manager_id und job_id aus der Tabelle employees angezeigt.

Geben Sie in der Klausel SELECT die Spalten in der Reihenfolge an, in der sie in der Ausgabe angezeigt werden sollen. Beispiel: Die folgende SQL-Anweisung zeigt die Spalte location_id vor der Spalte department_id an:

```
SELECT location_id, department_id FROM departments;
```

Hinweis: Um Anweisungen in SQL Developer auszuführen, können Sie die SQL-Anweisung in ein SQL Worksheet eingeben und auf das Symbol **Run Statement** klicken oder F9 drücken. Die Ausgabe wird in der Registerkarte **Results** angezeigt, wie auf der Folie dargestellt.

WHERE-Klauseln

- **Mit der optionalen Klausel WHERE:**
 - Zeilen in einer Abfrage filtern
 - Teilmenge von Zeilen erstellen
- **Syntax:**

```
SELECT * FROM table  
[WHERE condition];
```

- **Beispiel:**

```
SELECT location_id from departments  
WHERE department_name = 'Marketing';
```

ORACLE

Copyright © 2013, Oracle und/oder verbundene Unternehmen. All rights reserved. Alle Rechte vorbehalten.

Die Klausel WHERE gibt eine Bedingung zur Zeilenfilterung an und generiert eine Teilmenge aus den Zeilen in der Tabelle. Bedingungen bestehen aus einer Kombination aus einzelnen oder mehreren Ausdrücken und logischen (booleschen) Operatoren. Sie geben als Ergebnis den Wert TRUE, FALSE oder NULL zurück. Im Beispiel auf der Folie wird die location_id der Marketingabteilung abgerufen.

Mithilfe von WHERE-Klauseln können Sie auch Daten aus der Datenbank aktualisieren oder löschen.

Beispiel:

```
UPDATE departments  
SET department_name = 'Administration'  
WHERE department_id = 20;  
und  
DELETE from departments  
WHERE department_id =20;
```

ORDER BY-Klauseln

- Die optionale Klausel ORDER BY regelt die Reihenfolge der Zeilen.
- Syntax:

```
SELECT * FROM table  
[WHERE condition]  
[ORDER BY {<column> | <position>} [ASC|DESC] [, ...] ];
```

- Beispiel:

```
SELECT last_name, department_id, salary  
FROM employees  
ORDER BY department_id ASC, salary DESC;
```

ORACLE

Copyright © 2013, Oracle und/oder verbundene Unternehmen. All rights reserved. Alle Rechte vorbehalten.

Mit der Klausel ORDER BY legen Sie fest, in welcher Reihenfolge die Zeilen angezeigt werden sollen. Zeilen können auf- oder absteigend sortiert werden. Die standardmäßige Sortierfolge für Zeilen ist aufsteigend.

Im Beispiel auf der Folie werden Zeilen aus der Tabelle employees abgerufen und erst aufsteigend nach department_id und dann absteigend nach salary sortiert.

GROUP BY-Klauseln

- **Mit der optionalen Klausel GROUP BY fassen Sie Spalten mit übereinstimmenden Werten zu Untergruppen zusammen.**
- **In keiner Gruppe gibt es zwei Zeilen mit identischem Wert für die Spalte(n), nach der oder denen die Gruppierung erfolgt.**
- **Syntax:**

```
SELECT <column1, column2, ... column_n>
  FROM table
  [WHERE condition]
  [GROUP BY <column> [, ...] ]
  [ORDER BY <column> [, ...] ] ;
```

- **Beispiel:**

```
SELECT department_id, MIN(salary), MAX (salary)
  FROM employees
 GROUP BY department_id ;
```

ORACLE

Copyright © 2013, Oracle und/oder verbundene Unternehmen. All rights reserved. Alle Rechte vorbehalten.

Mithilfe der Klausel GROUP BY fassen Sie gewählte Zeilen anhand des Wertes für expr (s) in den einzelnen Zeilen zu Gruppen zusammen. Die Klausel gruppiert Zeilen, garantiert jedoch nicht die Reihenfolge der Ergebnismenge. Um die Reihenfolge der Gruppen festzulegen, geben Sie die Klausel ORDER BY an.

Alle SELECT-Listenelemente, die nicht Bestandteil von Aggregationsfunktionen sind, müssen in die Elementliste GROUP BY aufgenommen werden. Hierzu gehören sowohl Spalten als auch Ausdrücke. Die Datenbank gibt für jede Gruppe eine einzelne Zeile mit Aggregatinformationen zurück.

Im Beispiel auf der Folie wird für jede Abteilung aus der Tabelle employees das jeweils niedrigste und höchste Gehalt zurückgegeben.

Data Definition Language (DDL)

- **Mit DDL-Anweisungen können Sie Schemaobjekte definieren und löschen bzw. die Struktur von Schemaobjekten ändern.**
- **Häufige DDL-Anweisungen:**
 - **CREATE TABLE, ALTER TABLE und DROP TABLE**
 - **GRANT, REVOKE**
 - **TRUNCATE**

ORACLE

Copyright © 2013, Oracle und/oder verbundene Unternehmen. All rights reserved. Alle Rechte vorbehalten.

Mithilfe von DDL-(Data Definition Language-)Anweisungen können Sie die Attribute eines Objekts ändern, ohne gleichzeitig auch die Anwendungen ändern zu müssen, die auf das Objekt zugreifen. Daneben lässt sich mit DDL-Anweisungen auch die Struktur von Objekten ändern, während Benutzer in der Datenbank arbeiten. Häufige Verwendungsbereiche von DML-Anweisungen:

- Schemaobjekte und andere Datenbankstrukturen erstellen, ändern und löschen, einschließlich der Datenbank selbst und ihrer Benutzer
- Alle Daten in Schemaobjekten löschen, ohne die Struktur dieser Objekte zu entfernen
- Berechtigungen und Rollen erteilen und entziehen

Oracle Database schreibt die aktuelle Transaktion vor und nach jeder DDL-Anweisung implizit fest.

CREATE TABLE-Anweisungen

- **Mit der Anweisung CREATE TABLE erstellen Sie Tabellen in der Datenbank.**
- **Syntax:**

```
CREATE TABLE tablename (
  {column-definition | Table-level constraint}
  [ , {column-definition | Table-level constraint} ] * )
```

- **Beispiel:**

```
CREATE TABLE teach_dept (
  department_id NUMBER(3) PRIMARY KEY,
  department_name VARCHAR2(10));
```

ORACLE

Copyright © 2013, Oracle und/oder verbundene Unternehmen. All rights reserved. Alle Rechte vorbehalten.

Mit der Anweisung CREATE TABLE erstellen Sie Tabellen in der Datenbank. Voraussetzung für die Erstellung von Tabellen ist, dass Sie über die Berechtigung CREATE TABLE und einen Speicherbereich verfügen, in dem Sie Objekte erstellen können.

Die Eigentümer der Tabelle und der Datenbank erhalten nach der Erstellung einer Tabelle automatisch folgende Berechtigungen für die Tabelle:

- INSERT
- SELECT
- REFERENCES
- ALTER
- UPDATE

Die Eigentümer der Tabelle und der Datenbank können anderen Benutzern die vorstehenden Berechtigungen erteilen.

ALTER TABLE-Anweisungen

- **Mit der Anweisung ALTER TABLE ändern Sie die Definition einer vorhandenen Tabelle in der Datenbank.**
- **1. Beispiel:**

```
ALTER TABLE teach_dept  
ADD location_id NUMBER NOT NULL;
```

- **2. Beispiel:**

```
ALTER TABLE teach_dept  
MODIFY department_name VARCHAR2(30) NOT NULL;
```

ORACLE

Copyright © 2013, Oracle und/oder verbundene Unternehmen. All rights reserved. Alle Rechte vorbehalten.

Mithilfe der Anweisung ALTER TABLE können Sie Änderungen an einer vorhandenen Tabelle vornehmen.

Sie können:

- Spalten zu Tabellen hinzufügen
- Constraints zu Tabellen hinzufügen
- vorhandene Spaltendefinitionen ändern
- Spalten aus Tabellen löschen
- vorhandene Constraints aus Tabellen löschen
- die Breite der Spalten VARCHAR und CHAR vergrößern
- Tabellen in den schreibgeschützten Status setzen

Im 1. Beispiel auf der Folie wird die Tabelle teach_dept um die neue Spalte location_id erweitert.

Im 2. Beispiel wird die vorhandene Spalte department_name von VARCHAR2 (10) in VARCHAR2 (30) geändert und das Constraint NOT NULL hinzugefügt.

DROP TABLE-Anweisungen

- **Mit der Anweisung `DROP TABLE` löschen Sie die Tabelle mit sämtlichen darin enthaltenen Daten aus der Datenbank.**
- **Beispiel:**

```
DROP TABLE teach_dept;
```

- **`DROP TABLE` mit der Klausel `PURGE` löscht die Tabelle und gibt den damit verbundenen Speicherplatz frei.**
- **Bei Verwendung der Klausel `CASCADE CONSTRAINTS` werden alle Constraints zur referentiellen Integrität aus der Tabelle gelöscht.**

```
DROP TABLE teach_dept CASCADE CONSTRAINTS;
```

ORACLE

Copyright © 2013, Oracle und/oder verbundene Unternehmen. All rights reserved. Alle Rechte vorbehalten.

Mithilfe der Anweisung `DROP TABLE` können Sie eine Tabelle einschließlich ihres Inhalts aus der Datenbank löschen und in den Papierkorb verschieben. Beim Löschen einer Tabelle werden die abhängigen Objekte invalidiert und Objektberechtigungen für die Tabelle entfernt.

Um den zugewiesenen Speicherplatz für die Tabelle wieder für den Tablespace freizugeben, verwenden Sie die Anweisung `DROP TABLE` mit der Klausel `PURGE`. Sie können `DROP TABLE`-Anweisungen mit der Klausel `PURGE` nicht zurücksetzen und auch die Tabelle nicht wiederherstellen, wenn sie mit der Klausel `PURGE` gelöscht wurde.

Mit der Klausel `CASCADE CONSTRAINTS` können Sie die Referenz auf den Primärschlüssel und die eindeutigen Schlüssel in der gelöschten Tabelle aufheben.

GRANT-Anweisungen

- Die Anweisung **GRANT** erteilt Benutzern Berechtigungen zur Ausführung folgender Vorgänge:
 - Daten einfügen oder löschen
 - Fremdschlüsselreferenz auf die benannte Tabelle oder eine Teilmenge von Spalten aus einer Tabelle erstellen
 - Daten, Views oder eine Teilmenge von Spalten aus einer Tabelle wählen
 - Trigger für Tabellen erstellen
 - Angegebene Funktionen oder Prozeduren ausführen
- Beispiel:

```
GRANT SELECT any table to PUBLIC;
```

ORACLE®

Copyright © 2013, Oracle und/oder verbundene Unternehmen. All rights reserved. Alle Rechte vorbehalten.

Mithilfe der Anweisung GRANT können Sie:

- bestimmten Benutzern oder Rollen (bzw. allen Benutzern) Berechtigungen zur Ausführung von Aktionen an Datenbankobjekten erteilen
- Benutzern, PUBLIC oder anderen Rollen eine Rolle erteilen

Vergewissern Sie sich vor dem Absetzen einer GRANT-Anweisung, dass für die Autorisierungseigenschaft derby.database.sql der Wert True eingestellt ist. Mit dieser Eigenschaft wird der SQL-Autorisierungsmodus aktiviert. Sie können Berechtigungen für ein Objekt erteilen, wenn Sie der Eigentümer der Datenbank sind.

Um allen Benutzern Berechtigungen zu erteilen, verwenden Sie das Schlüsselwort PUBLIC. Ist PUBLIC angegeben, gelten die Berechtigungen oder Rollen für alle aktuellen und künftigen Benutzer.

Typen von Berechtigungen

- **Folgende Berechtigungen mit der Anweisung GRANT zuweisen:**
 - **ALL PRIVILEGES**
 - **DELETE**
 - **INSERT**
 - **REFERENCES**
 - **SELECT**
 - **UPDATE**

ORACLE

Copyright © 2013, Oracle und/oder verbundene Unternehmen. All rights reserved. Alle Rechte vorbehalten.

Oracle Database bietet verschiedene Berechtigungstypen, um Benutzern oder Rollen Berechtigungen zu erteilen:

- Mit dem Berechtigungstyp **ALL PRIVILEGES** erteilen Sie Benutzern oder Rollen sämtliche Berechtigungen für die angegebene Tabelle.
- Mit dem Berechtigungstyp **DELETE** erteilen Sie die Berechtigung, Zeilen aus der angegebenen Tabelle zu löschen.
- Mit dem Berechtigungstyp **INSERT** erteilen Sie die Berechtigung, Zeilen in die angegebene Tabelle einzufügen.
- Mit dem Berechtigungstyp **REFERENCES** erteilen Sie die Berechtigung zur Erstellung einer Fremdschlüsselreferenz auf die angegebene Tabelle.
- Mit dem Berechtigungstyp **SELECT** erteilen Sie die Berechtigung zur Ausführung von **SELECT**-Anweisungen für eine Tabelle oder View.
- Mit dem Berechtigungstyp **UPDATE** erteilen Sie die Berechtigung zur Verwendung von **UPDATE**-Anweisungen für die angegebene Tabelle.

REVOKE-Anweisungen

- **Mit REVOKE-Anweisungen entziehen Sie Benutzern Berechtigungen zur Ausführung von Aktionen an Datenbankobjekten.**
- **Einem Benutzer eine Systemberechtigung entziehen:**

```
REVOKE DROP ANY TABLE  
FROM hr;
```
- **Einem Benutzer eine Rolle entziehen:**

```
REVOKE dw_manager  
FROM sh;
```

ORACLE

Copyright © 2013, Oracle und/oder verbundene Unternehmen. All rights reserved. Alle Rechte vorbehalten.

Mit der Anweisung REVOKE entziehen Sie einem bestimmten Benutzer (bzw. mehreren Benutzern) oder einer Rolle die Berechtigungen zur Ausführung von Aktionen an Datenbankobjekten. Folgende Vorgänge sind möglich:

- Benutzern, PUBLIC oder anderen Rolle eine Rolle entziehen
- Berechtigungen für ein Objekt entziehen, sofern Sie der Eigentümer des Objekts oder der Datenbank sind

Hinweis: Um eine Rolle oder eine Systemberechtigung entziehen zu können, benötigen Sie die Berechtigung mit der ADMIN OPTION.

TRUNCATE TABLE-Anweisungen

- **Mithilfe von TRUNCATE TABLE-Anweisungen löschen Sie alle Zeilen einer Tabelle.**
- **Beispiel:**

```
TRUNCATE TABLE employees_demo;
```

- **Oracle Database führt standardmäßig folgende Aufgaben durch:**
 - Durch die gelöschten Zeilen belegten Speicherplatz freigeben
 - Speicherparameter NEXT auf die Größe des letzten Extents einstellen, das durch den Leerungsprozess aus dem Segment entfernt wurde

ORACLE

Copyright © 2013, Oracle und/oder verbundene Unternehmen. All rights reserved. Alle Rechte vorbehalten.

Mit der Anweisung TRUNCATE TABLE löschen Sie alle Zeilen aus einer angegebenen Tabelle. Zeilen mit der Anweisung TRUNCATE TABLE zu entfernen, kann effizienter sein als die Tabelle zu löschen und neu zu erstellen. Beim Löschen und erneuten Erstellen einer Tabelle:

- werden die abhängigen Objekte der Tabelle invalidiert
- müssen Sie:
 - Objektberechtigungen erneut vergeben
 - Indizes, Integritäts-Constraints und Trigger neu erstellen
 - Speicherparameter erneut angeben

Bei Verwendung der Anweisung TRUNCATE TABLE können Sie auf diese Arbeitsschritte verzichten.

Hinweis: TRUNCATE TABLE-Anweisungen können nicht zurückgesetzt werden.

Data Manipulation Language (DML)

- **Mit DML-Anweisungen fragen Sie Daten in vorhandenen Schemaobjekten ab oder bearbeiten diese Daten.**
- **Eine DML-Anweisung wird ausgeführt, wenn Sie:**
 - mit der Anweisung **INSERT** neue Zeilen zu einer Tabelle hinzufügen
 - mit der Anweisung **UPDATE** vorhandene Zeilen in einer Tabelle ändern
 - mit der Anweisung **DELETE** vorhandene Zeilen aus einer Tabelle löschen
- **Eine *Transaktion* besteht aus einer Zusammenstellung von DML-Anweisungen, die eine logische Arbeitseinheit bilden.**

ORACLE

Copyright © 2013, Oracle und/oder verbundene Unternehmen. All rights reserved. Alle Rechte vorbehalten.

Mithilfe von DML-(Data Manipulation Language-)Anweisungen können Sie den Inhalt vorhandener Schemaobjekte abfragen oder ändern. Häufige Verwendungsbereiche von DML-Anweisungen:

- Neue Datenzeilen zu einer Tabelle oder View hinzufügen (durch Angabe einer Liste von Spaltenwerten oder durch Auswahl und Bearbeitung vorhandener Daten mithilfe einer Unterabfrage)
- Spaltenwerte in den vorhandenen Zeilen einer Tabelle oder View ändern
- Zeilen aus Tabellen oder Views entfernen

Eine Zusammenstellung von DML-Anweisungen, die eine logische Arbeitseinheit bilden, wird als Transaktion bezeichnet. Im Gegensatz zu DDL-Anweisungen schreiben DML-Anweisungen die aktuelle Transaktion nicht implizit fest.

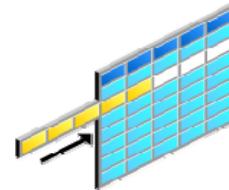
INSERT-Anweisungen

- **Mithilfe von INSERT-Anweisungen fügen Sie neue Zeilen zu einer Tabelle hinzu.**
- **Syntax:**

```
INSERT INTO table [(column [, column...])]  
VALUES      (value [, value...]);
```

- **Beispiel:**

```
INSERT INTO departments  
VALUES      (200, 'Development', 104, 1400);  
1 rows inserted.
```



ORACLE

Copyright © 2013, Oracle und/oder verbundene Unternehmen. All rights reserved. Alle Rechte vorbehalten.

Neue Tabellenzeilen fügen Sie mit der Anweisung `INSERT` hinzu. Achten Sie darauf, neue Zeilen mit Werten für alle Spalten einzufügen und die Werte in der standardmäßigen Reihenfolge der Spalten in der Tabelle aufzulisten. Optional können Sie die Spalten auch in der Anweisung `INSERT` auflisten.

Beispiel:

```
INSERT INTO job_history (employee_id, start_date, end_date,  
                      job_id)  
VALUES (120, '25-JUL-06', '12-FEB-08', 'AC_ACCOUNT');
```

Mit der auf der Folie gezeigten Syntax können Sie jeweils eine einzelne Zeile einfügen. Das Schlüsselwort `VALUES` weist die Werte von Ausdrücken den entsprechenden Spalten in der Spaltenliste zu.

UPDATE-Anweisungen – Syntax

- **Mit der Anweisung UPDATE ändern Sie die vorhandenen Zeilen in einer Tabelle.**
- **Falls erforderlich, können mehrere Zeilen gleichzeitig aktualisiert werden.**

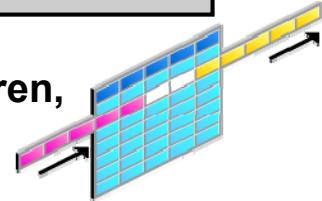
```
UPDATE      table
SET         column = value [, column = value, ...]
[WHERE      condition];
```

- **Beispiel:**

```
UPDATE      copy_emp
SET         employee_id = 110;
```

22 rows updated

Um einen Spaltenwert mit NULL zu aktualisieren, geben Sie SET *column_name*= NULL an.



ORACLE

Copyright © 2013, Oracle und/oder verbundene Unternehmen. All rights reserved. Alle Rechte vorbehalten.

Mit der Anweisung UPDATE ändern Sie die vorhandenen Werte in einer Tabelle. Um den Aktualisierungsvorgang zu prüfen, zeigen Sie über eine Tabellenabfrage die aktualisierten Zeilen an. Um nur bestimmte Zeilen zu ändern, geben Sie die Klausel WHERE an.

Beispiel:

```
UPDATE employees
SET     salary = 17500
WHERE   employee_id = 102;
```

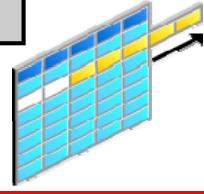
In der Regel identifizieren Sie die zu aktualisierende Zeile in der Klausel WHERE mithilfe der Primärschlüsselspalte. Beispiel: Sie möchten eine bestimmte Zeile in der Tabelle employees aktualisieren. Um die Zeile anzugeben, verwenden Sie employee_id anstelle von employee_name, da eventuell mehrere Mitarbeiter denselben Namen haben können.

Hinweis: Das Schlüsselwort condition besteht in der Regel aus Spaltennamen, Ausdrücken, Konstanten, Unterabfragen und Vergleichsoperatoren.

DELETE-Anweisungen

- **Mit der Anweisung DELETE löschen Sie vorhandene Zeilen aus einer Tabelle.**
 - **Syntax:**
- ```
DELETE [FROM] table
[WHERE condition];
```
- **Um bestimmte Zeilen aus einer Tabelle zu löschen, geben Sie die Anweisung DELETE mit der Klausel WHERE an.**

```
DELETE FROM departments
WHERE department_name = 'Finance';
1 rows deleted
```



ORACLE

Copyright © 2013, Oracle und/oder verbundene Unternehmen. All rights reserved. Alle Rechte vorbehalten.

Mit der Anweisung DELETE entfernen Sie vorhandene Zeilen aus einer Tabelle. Um bestimmte Zeilen auf Basis einer vorgegebenen Bedingung zu löschen, geben Sie die Klausel WHERE an. Die zu löschenen Zeilen werden über das Schlüsselwort condition bestimmt, das Spaltennamen, Ausdrücke, Konstanten, Unterabfragen und Vergleichsoperatoren enthalten kann.

Im ersten Beispiel auf der Folie wird die Finanzabteilung aus der Tabelle departments gelöscht. Um den Löschvorgang zu prüfen, fragen Sie die Tabelle mit der Anweisung SELECT ab.

```
SELECT *
FROM departments
WHERE department_name = 'Finance';
```

Wenn Sie die Klausel WHERE weglassen, werden alle Zeilen aus der Tabelle gelöscht. Beispiel:

```
DELETE FROM copy_emp;
```

Im vorstehenden Beispiel werden alle Zeilen aus der Tabelle copy\_emp gelöscht.

## Anweisungen zur Transaktionskontrolle

- **Anweisungen zur Transaktionskontrolle dienen zur Verwaltung der über DML-Anweisungen vorgenommenen Änderungen.**
- **Die DML-Anweisungen werden zu Transaktionen zusammengefasst.**
- **Anweisungen zur Transaktionskontrolle:**
  - **COMMIT**
  - **ROLLBACK**
  - **SAVEPOINT**



Copyright © 2013, Oracle und/oder verbundene Unternehmen. All rights reserved. Alle Rechte vorbehalten.

Eine Transaktion besteht aus einer Folge von SQL-Anweisungen, die von Oracle Database als einzelne Einheit behandelt werden. Anweisungen zur Transaktionskontrolle werden in einer Datenbank verwendet, um die über DML-Anweisungen vorgenommenen Änderungen zu verwalten und diese Anweisungen zu Transaktionen zusammenzufassen.

Jeder Transaktion wird eine eindeutige `transaction_id` zugewiesen. Die in der Transaktion enthaltenen SQL-Anweisungen werden entweder allesamt festgeschrieben (und in der Datenbank angewendet) oder allesamt zurückgesetzt (und in der Datenbank rückgängig gemacht).

## COMMIT-Anweisungen

- **Mit COMMIT-Anweisungen können Sie:**
  - die während der aktuellen Transaktion vorgenommenen Änderungen endgültig speichern
  - alle Savepoints in der Transaktion löschen
  - Transaktionssperren aufheben
- **Beispiel:**

```
INSERT INTO departments
VALUES (201, 'Engineering', 106, 1400);
COMMIT;

1 rows inserted.
committed.
```

ORACLE

Copyright © 2013, Oracle und/oder verbundene Unternehmen. All rights reserved. Alle Rechte vorbehalten.

Wenn Sie die aktuelle Transaktion mit der Anweisung COMMIT beenden, werden alle noch nicht gespeicherten Datenänderungen dauerhaft festgeschrieben. Dabei werden alle Zeilen- und Tabellensperren aufgehoben sowie alle seit dem letzten Commit- oder Rollback-Vorgang eventuell festgelegten Savepoints gelöscht. Die mit der Anweisung COMMIT vorgenommenen Änderungen sind für alle Benutzer sichtbar.

Oracle empfiehlt, jede Transaktion in Ihren Anwendungsprogrammen (einschließlich der letzten Transaktion) explizit mit COMMIT oder ROLLBACK zu beenden, bevor Sie sich bei Oracle Database abmelden. Wenn Sie die Transaktion nicht explizit festschreiben und das Programm anschließend nicht ordnungsgemäß beendet werden kann, wird andernfalls die letzte nicht festgeschriebene Transaktion automatisch zurückgesetzt.

**Hinweis:** Oracle Database setzt vor und nach DDL-Anweisungen implizite COMMIT-Anweisungen ab.

## ROLLBACK-Anweisungen

- **Mit der Anweisung ROLLBACK machen Sie die während der aktuellen Transaktion an der Datenbank vorgenommenen Änderungen rückgängig.**
- **Um den Teil der Transaktion nach dem Savepoint rückgängig zu machen, verwenden Sie die Klausel TO SAVEPOINT.**
- **Beispiel:**

```

UPDATE employees
SET salary = 7000
WHERE last_name = 'Ernst';
SAVEPOINT Ernst_sal;

UPDATE employees
SET salary = 12000
WHERE last_name = 'Mourgos';

ROLLBACK TO SAVEPOINT Ernst_sal;

```

**ORACLE**

Copyright © 2013, Oracle und/oder verbundene Unternehmen. All rights reserved. Alle Rechte vorbehalten.

Mit der Anweisung ROLLBACK werden die in der aktuellen Transaktion durchgeführten Aktionen zurückgesetzt. Um die aktuelle Transaktion zurückzusetzen, sind keine Berechtigungen erforderlich.

Mit ROLLBACK und der Klausel TO SAVEPOINT führen Sie folgende Vorgänge durch:

- Nur den Teil der Transaktion zurücksetzen, der nach dem Savepoint liegt
- Alle nach diesem Savepoint erstellten Savepoints löschen. Der benannte Savepoint bleibt erhalten, sodass Sie mehrere Rollbacks bis zu diesem Savepoint vornehmen können.

Mit ROLLBACK ohne Angabe der Klausel TO SAVEPOINT führen Sie folgende Vorgänge durch:

- Transaktion beenden
- Alle während der aktuellen Transaktion vorgenommenen Änderungen rückgängig machen
- Alle Savepoints in der Transaktion löschen

## SAVEPOINT-Anweisungen

- **Mit der Anweisung `SAVEPOINT` benennen und markieren Sie den aktuellen Verarbeitungspunkt einer Transaktion.**
- **Geben Sie für jeden Savepoint einen Namen an.**
- **Vergeben Sie eindeutige Namen für Savepoints innerhalb einer Transaktion, um Überschreibungen zu vermeiden.**
- **Syntax:**

```
SAVEPOINT savepoint;
```

ORACLE

Copyright © 2013, Oracle und/oder verbundene Unternehmen. All rights reserved. Alle Rechte vorbehalten.

Mit der Anweisung `SAVEPOINT` kennzeichnen Sie einen Punkt in der Transaktion, bis zu dem Sie spätere Rollbacks vornehmen können. Jeder Savepoint muss über einen eindeutigen Namen verfügen. Wenn Sie im Anschluss einen zweiten Savepoint mit demselben Namen erstellen, wird der frühere Savepoint gelöscht.

Nach der Erstellung eines Savepoints können Sie mit der Verarbeitung fortfahren, Ihre Arbeit festschreiben, die gesamte Transaktion zurückschreiben oder alle Aktionen nach dem Savepoint zurücksetzen.

Bei einem einfachen Rollback- oder Commit-Vorgang werden alle Savepoints gelöscht. Bei einem Rollback bis zu einem Savepoint werden alle späteren Savepoints gelöscht. Der Savepoint, der für den Rollback-Vorgang verwendet wurde, bleibt erhalten.

Wenn Savepoint-Namen innerhalb einer Transaktion mehrfach verwendet werden, verschiebt Oracle Database den Savepoint von der bisherigen an die aktuelle Position in der Transaktion und setzt damit den älteren Savepoint außer Kraft.

# Joins

Mithilfe von Joins fragen Sie Daten aus mehreren Tabellen ab:

```
SELECT table1.column, table2.column
FROM table1, table2
WHERE table1.column1 = table2.column2;
```

- **Join-Bedingungen werden in der Klausel WHERE angegeben.**
- **Setzen Sie den Tabellennamen vor den Spaltennamen, wenn derselbe Spaltenname in mehreren Tabellen vorkommt.**

ORACLE

Copyright © 2013, Oracle und/oder verbundene Unternehmen. All rights reserved. Alle Rechte vorbehalten.

Wenn Sie Daten aus mehreren Tabellen einer Datenbank benötigen, werden *Join*-Bedingungen verwendet. Zeilen in einer Tabelle können mit Zeilen einer anderen Tabelle über gemeinsame Werte in den entsprechenden Spalten verknüpft werden. (In der Regel geschieht dies über Primär- und Fremdschlüssel Spalten.)

Um Daten aus zwei oder mehr verknüpften Tabellen anzuzeigen, geben Sie in der Klausel WHERE eine einfache Join-Bedingung an.

Für die Syntax gilt:

|                  |                                                                                             |
|------------------|---------------------------------------------------------------------------------------------|
| table1.column    | Gibt die Tabelle und die Spalte an, aus der Daten abgerufen werden                          |
| table1.column1 = | Ist die Bedingung, nach der Tabellen verknüpft (oder zueinander in Relation gesetzt) werden |
| table2.column2   |                                                                                             |

## Richtlinien

- Setzen Sie bei der Erstellung von SELECT-Anweisungen, die Tabellen verknüpfen, den Tabellennamen vor den Spaltennamen, um den Code besser verständlich zu machen und den Datenbankzugriff zu optimieren.
- Wenn derselbe Spaltenname in mehreren Tabellen verwendet wird, müssen Sie dem Spaltenamen den Tabellennamen voranstellen.
- Um  $n$  Tabellen miteinander zu verknüpfen, benötigen Sie mindestens  $n-1$  Join-Bedingungen. Beispiel: Um vier Tabellen zu verknüpfen, benötigen Sie mindestens drei Joins. Diese Regel gilt möglicherweise nicht, wenn Ihre Tabelle über einen verketteten Primärschlüssel verfügt, da in diesem Fall mehrere Spalten benötigt werden, um jede Zeile eindeutig zu identifizieren.

## Typen von Joins

- **Natural Join**
- **Equi Join**
- **Non-Equi Join**
- **Outer Join**
- **Self Join**
- **Cross Join**

ORACLE

Copyright © 2013, Oracle und/oder verbundene Unternehmen. All rights reserved. Alle Rechte vorbehalten.

Mit der Join-Syntax von Oracle können Sie Tabellen verknüpfen.

**Hinweis:** In Datenbankreleases bis Oracle9*i* galt für Joins eine Oracle-spezifische Syntax. Die SQL:1999-konforme Join-Syntax bietet gegenüber der Oracle-spezifischen Join-Syntax keine Performancevorteile.

## Mehrdeutige Spaltennamen eindeutig kennzeichnen

- **Spaltennamen, die in mehreren Tabellen vorkommen, müssen durch das Tabellenpräfix eindeutig gekennzeichnet werden.**
- **Tabellenpräfixe verbessern die Performance.**
- **Verwenden Sie als Präfix Tabellenaliasnamen anstelle von vollständigen Tabellennamen.**
- **Tabellenaliasnamen verkürzen die Namen von Tabellen:**
  - Kürzerer SQL-Code, weniger Speicherbedarf
- **Spaltenaliasnamen dienen der Unterscheidung von Spalten, die identische Namen haben, aber in verschiedenen Tabellen stehen.**

**ORACLE**

Copyright © 2013, Oracle und/oder verbundene Unternehmen. All rights reserved. Alle Rechte vorbehalten.

Wenn Sie zwei oder mehr Tabellen miteinander verknüpfen, kennzeichnen Sie die Spaltennamen durch den jeweiligen Tabellennamen, um Mehrdeutigkeit zu vermeiden. Ohne das Tabellenpräfix könnte sich der Spaltenname DEPARTMENT\_ID in der Liste SELECT sowohl auf die entsprechende Spalte in der Tabelle DEPARTMENTS als auch in der Tabelle EMPLOYEES beziehen. Daher müssen Sie das Tabellenpräfix hinzufügen, um die Abfrage auszuführen. Wenn ein Spaltenname lediglich in einer Tabelle vorkommt, besteht keine Notwendigkeit, die Spalte durch Angabe des Tabellennamens eindeutig zu kennzeichnen. Tabellenpräfixe verbessern jedoch die Performance, weil sie dem Oracle-Server genau angeben, wo sich die Spalten befinden.

Die eindeutige Kennzeichnung von Spaltennamen durch Tabellennamen kann insbesondere bei langen Tabellennamen sehr zeitaufwändig sein. Sie können daher anstelle von Tabellennamen *Tabellenaliasnamen* verwenden. Analog zu Spaltenaliasnamen, die einer Spalte einen anderen Namen zuweisen, legen Tabellenaliasnamen einen anderen Namen für eine Tabelle fest. Tabellenaliasnamen dienen dazu, den SQL-Code kürzer und damit weniger speicherintensiv zu halten.

Der Tabellename wird komplett angegeben. Danach folgt ein Leerzeichen und dann der Tabellenalias. Beispiel: Der Tabelle EMPLOYEES kann der Alias e zugewiesen werden, der Tabelle DEPARTMENTS der Alias d.

## Richtlinien

- Tabellenaliasnamen können bis zu 30 Zeichen lang sein, kürzere Aliasnamen sind allerdings besser.
- Wenn für einen bestimmten Tabellennamen in der Klausel `FROM` ein Tabellenalias verwendet wird, muss dieser Tabellenalias in der gesamten Anweisung `SELECT` anstelle des Tabellennamens verwendet werden.
- Wählen Sie möglichst aussagekräftige Tabellenaliasnamen.
- Ein Tabellenalias gilt nur für die aktuelle Anweisung `SELECT`.

## Natural Joins

- Die Klausel **NATURAL JOIN** basiert auf allen Spalten, die in beiden Tabellen denselben Namen haben.
- Die Klausel wählt Zeilen aus Tabellen, deren Spalten identische Namen und Datenwerte aufweisen.
- Beispiel:

```
SELECT country_id, location_id, country_name, city
FROM countries NATURAL JOIN locations;
```

| COUNTRY_ID | LOCATION_ID | COUNTRY_NAME             | CITY                |
|------------|-------------|--------------------------|---------------------|
| 1 US       | 1400        | United States of America | Southlake           |
| 2 US       | 1500        | United States of America | South San Francisco |
| 3 US       | 1700        | United States of America | Seattle             |
| 4 CA       | 1800        | Canada                   | Toronto             |
| 5 UK       | 2500        | United Kingdom           | Oxford              |

ORACLE

Copyright © 2013, Oracle und/oder verbundene Unternehmen. All rights reserved. Alle Rechte vorbehalten.

Sie können Tabellen auf Basis von Spalten mit gleichem Namen und gleichem Datentyp automatisch verknüpfen, indem Sie die Schlüsselwörter **NATURAL JOIN** angeben.

**Hinweis:** Joins sind nur für Spalten möglich, die in beiden Tabellen denselben Namen und denselben Datentyp aufweisen. Wenn die Spalten denselben Namen haben, ihre Datentypen sich jedoch unterscheiden, verursacht die Syntax **NATURAL JOIN** einen Fehler.

Im Beispiel auf der Folie werden die Tabellen COUNTRIES und LOCATIONS über die Spalte COUNTRY\_ID verknüpft, da dies der einzige Spaltenname ist, der in beiden Tabellen vorkommt. Wären weitere übereinstimmende Spalten vorhanden, würde der Join alle diese Spalten berücksichtigen.

## Equi Joins

**EMPLOYEES**

|    | EMPLOYEE_ID | DEPARTMENT_ID |
|----|-------------|---------------|
| 1  | 200         | 10            |
| 2  | 201         | 20            |
| 3  | 202         | 20            |
| 4  | 205         | 110           |
| 5  | 206         | 110           |
| 6  | 100         | 90            |
| 7  | 101         | 90            |
| 8  | 102         | 90            |
| 9  | 103         | 60            |
| 10 | 104         | 60            |

**DEPARTMENTS**

|   | DEPARTMENT_ID | DEPARTMENT_NAME |
|---|---------------|-----------------|
| 1 | 10            | Administration  |
| 2 | 20            | Marketing       |
| 3 | 50            | Shipping        |
| 4 | 60            | IT              |
| 5 | 80            | Sales           |
| 6 | 90            | Executive       |
| 7 | 110           | Accounting      |
| 8 | 190           | Contracting     |

Primärschlüssel

Fremdschlüssel

ORACLE

Copyright © 2013, Oracle und/oder verbundene Unternehmen. All rights reserved. Alle Rechte vorbehalten.

Ein **Equi Join** ist ein Join mit einer Join-Bedingung, die einen Gleichheitsoperator enthält. Equi Joins kombinieren Zeilen, die äquivalente Werte in den angegebenen Spalten enthalten. Um die Abteilung eines Mitarbeiters zu bestimmen, vergleichen Sie die Werte in der Spalte DEPARTMENT\_ID der Tabelle EMPLOYEES mit den Werten für DEPARTMENT\_ID in der Tabelle DEPARTMENTS. Die Beziehung zwischen den Tabellen EMPLOYEES und DEPARTMENTS ist ein **Equi Join**, das heißt, die Werte in der Spalte DEPARTMENT\_ID müssen in beiden Tabellen gleich sein. Häufig sind bei dieser Art von Join Primärschlüssel- und Fremdschlüsselkomponenten beteiligt.

**Hinweis:** Equi Joins werden auch als *einfache Joins* bezeichnet.

## Datensätze mithilfe von Equi Joins abrufen

```
SELECT e.employee_id, e.last_name, e.department_id,
 d.department_id, d.location_id
 FROM employees e JOIN departments d
 WHERE e.department_id = d.department_id;
```

| EMPLOYEE_ID | LAST_NAME     | DEPARTMENT_ID | DEPARTMENT_ID_1 | LOCATION_ID |
|-------------|---------------|---------------|-----------------|-------------|
| 1           | 200 Whalen    | 10            | 10              | 1700        |
| 2           | 201 Hartstein | 20            | 20              | 1800        |
| 3           | 202 Fay       | 20            | 20              | 1800        |
| 4           | 144 Vargas    | 50            | 50              | 1500        |
| 5           | 143 Matos     | 50            | 50              | 1500        |
| 6           | 142 Davies    | 50            | 50              | 1500        |
| 7           | 141 Rajs      | 50            | 50              | 1500        |
| 8           | 124 Mourgos   | 50            | 50              | 1500        |
| 9           | 103 Hunold    | 60            | 60              | 1400        |
| 10          | 104 Ernst     | 60            | 60              | 1400        |
| 11          | 107 Lorentz   | 60            | 60              | 1400        |
| ...         |               |               |                 |             |

ORACLE

Copyright © 2013, Oracle und/oder verbundene Unternehmen. All rights reserved. Alle Rechte vorbehalten.

Im Beispiel auf der Folie gilt:

- Die Klausel **SELECT** gibt die abzurufenden Spaltennamen an:
  - Nachname, Personalnummer und Abteilungsnummer des Mitarbeiters aus den entsprechenden Spalten der Tabelle EMPLOYEES
  - Abteilungs-ID und Standort-ID aus den entsprechenden Spalten der Tabelle DEPARTMENTS
- Die Klausel **FROM** gibt die beiden Tabellen an, auf die die Datenbank zugreifen muss:
  - EMPLOYEES
  - DEPARTMENTS
- Die Klausel **WHERE** gibt an, wie die Tabellen verknüpft werden sollen:
   
e.department\_id = d.department\_id

Da die Spalte DEPARTMENT\_ID in beiden Tabellen enthalten ist, muss dem Spaltennamen der Tabellenalias vorangestellt werden, um Mehrdeutigkeit zu vermeiden. Eine Kennzeichnung anderer Spalten, die nicht in beiden Tabellen vorkommen, durch einen Tabellenalias ist nicht erforderlich, empfiehlt sich jedoch aus Performancegründen.

**Hinweis:** Wenn Sie die Abfrage mit dem Symbol **Execute Statement** ausführen, hängt SQL Developer zur Unterscheidung der beiden DEPARTMENT\_IDS "\_1" an.

## Suchbedingungen mit den Operatoren AND und WHERE hinzufügen

```
SELECT d.department_id, d.department_name, l.city
FROM departments d JOIN locations l
ON d.location_id = l.location_id
AND d.department_id IN (20, 50);
```

A table with three columns: DEPARTMENT\_ID, DEPARTMENT\_NAME, and CITY. It contains two rows:

|   | DEPARTMENT_ID | DEPARTMENT_NAME | CITY                |
|---|---------------|-----------------|---------------------|
| 1 | 20            | Marketing       | Toronto             |
| 2 | 50            | Shipping        | South San Francisco |

```
SELECT d.department_id, d.department_name, l.city
FROM departments d JOIN locations l
ON d.location_id = l.location_id
WHERE d.department_id IN (20, 50);
```

ORACLE

Copyright © 2013, Oracle und/oder verbundene Unternehmen. All rights reserved. Alle Rechte vorbehalten.

Neben dem Join können Sie auch Kriterien für die Klausel WHERE angeben, um die Zeilen aus einer oder mehreren Tabellen des Joins einzuschränken. Im Beispiel auf der Folie werden die Tabellen DEPARTMENTS und LOCATIONS verknüpft und darüber hinaus nur die Abteilungen mit der ID 20 oder 50 angezeigt. Um zusätzliche Bedingungen für die Klausel ON aufzunehmen, können Sie AND-Klauseln hinzufügen. Alternativ können Sie zusätzliche Bedingungen über eine WHERE-Klausel anwenden.

Beide Abfragen führen zum selben Ergebnis.

| LAST_NAME | DEPARTMENT_ID | DEPARTMENT_NAME |
|-----------|---------------|-----------------|
| Matos     | 50            | Shipping        |

## Datensätze mithilfe von Non-Equi Joins abrufen

```
SELECT e.last_name, e.salary, j.grade_level
FROM employees e JOIN job_grades j
ON e.salary
 BETWEEN j.lowest_sal AND j.highest_sal;
```

|   | LAST_NAME | SALARY | GRADE_LEVEL |
|---|-----------|--------|-------------|
| 1 | Vargas    | 2500   | A           |
| 2 | Matos     | 2600   | A           |
| 3 | Davies    | 3100   | B           |
| 4 | Rajs      | 3500   | B           |
| 5 | Lorentz   | 4200   | B           |
| 6 | Whalen    | 4400   | B           |
| 7 | Fay       | 6000   | C           |

...

ORACLE

Copyright © 2013, Oracle und/oder verbundene Unternehmen. All rights reserved. Alle Rechte vorbehalten.

Im Beispiel auf der Folie wird ein Non-Equi Join erstellt, um die Gehaltsstufe eines Mitarbeiters auszuwerten. Das Gehalt muss *zwischen* dem Mindest- und Höchstwert für die einzelnen Gehaltsbereiche liegen.

Jeder Mitarbeiter wird bei der Ausführung dieser Abfrage genau einmal angezeigt. Dafür gibt es zwei Gründe:

- Keine der Zeilen in der Tabelle `job_grades` enthält Gehaltsstufen, die sich überschneiden. Das bedeutet, der Gehaltswert für einen Mitarbeiter kann nur zwischen dem Mindest- und Höchstwert aus einer der Zeilen der Tabelle `job_grades` liegen.
- Alle Mitarbeitergehälter liegen innerhalb der durch die Tabelle `job_grades` festgelegten Grenzwerte. Kein Mitarbeiter verdient also weniger als das niedrigste Gehalt aus der Spalte `LOWEST_SAL` oder mehr als das höchste Gehalt aus der Spalte `HIGHEST_SAL`.

**Hinweis:** Es können auch andere Bedingungen verwendet werden, z. B. `<=` und `>=`. `BETWEEN` ist jedoch die einfachste Möglichkeit. Geben Sie zuerst den Mindest- und dann den Höchstwert an, wenn Sie den Operator `BETWEEN` verwenden. Der Oracle-Server übersetzt den Operator `BETWEEN` in ein `AND`-Bedingungspaar. Daher bietet `BETWEEN` keine Performancevorteile und dient nur der logischen Einfachheit.

Die Tabellenaliasnamen wurden im Beispiel auf der Folie aus Performancegründen verwendet, nicht wegen möglicher Mehrdeutigkeit.

## Datensätze mithilfe von USING-Klauseln abrufen

- Um nur eine Spalte zu verwenden, wenn mehrere übereinstimmende Spalten existieren, verwenden Sie die Klausel **USING**.
- Sie können diese Klausel nicht mit einem **NATURAL Join** angeben.
- Kennzeichnen Sie den Spaltennamen nicht mit einem Tabellennamen oder -alias.
- Beispiel:

```
SELECT country_id, country_name, location_id, city
FROM countries JOIN locations
USING (country_id) ;
```

|   | COUNTRY_ID | COUNTRY_NAME             | LOCATION_ID | CITY                |
|---|------------|--------------------------|-------------|---------------------|
| 1 | US         | United States of America | 1400        | Southlake           |
| 2 | US         | United States of America | 1500        | South San Francisco |
| 3 | US         | United States of America | 1700        | Seattle             |
| 4 | CA         | Canada                   | 1800        | Toronto             |
| 5 | UK         | United Kingdom           | 2500        | Oxford              |

ORACLE

Copyright © 2013, Oracle und/oder verbundene Unternehmen. All rights reserved. Alle Rechte vorbehalten.

Im Beispiel auf der Folie werden die Spalten COUNTRY\_ID aus den Tabellen COUNTRIES und LOCATIONS verknüpft. Damit wird die LOCATION\_ID für den Standort angezeigt, an dem ein Mitarbeiter arbeitet.

## Datensätze mithilfe von ON-Klauseln abrufen

- Die Join-Bedingung für Natural Joins ist im Prinzip ein Equi Join aller Spalten mit gleichem Namen.
- Mit der Klausel **ON** können Sie beliebige Bedingungen oder die zu verknüpfenden Spalten angeben.
- Die Klausel **ON** macht den Code verständlicher.

```
SELECT e.employee_id, e.last_name, j.department_id,
 FROM employees e JOIN job_history j
 ON (e.employee_id = j.employee_id);
```

|   | EMPLOYEE_ID | LAST_NAME | DEPARTMENT_ID |
|---|-------------|-----------|---------------|
| 1 | 101         | Kochhar   | 110           |
| 2 | 101         | Kochhar   | 110           |
| 3 | 102         | De Haan   | 60            |
| 4 | 176         | Taylor    | 80            |
| 5 | 176         | Taylor    | 80            |
| 6 | 200         | Whalen    | 90            |
| 7 | 200         | Whalen    | 90            |
| 8 | 201         | Hartstein | 20            |

ORACLE

Copyright © 2013, Oracle und/oder verbundene Unternehmen. All rights reserved. Alle Rechte vorbehalten.

Wenn Sie eine Join-Bedingung mit der Klausel **ON** angeben, können Sie Join-Bedingungen in der Klausel **WHERE** getrennt von Such- oder Filterkriterien angeben.

In diesem Beispiel werden die Spalten **EMPLOYEE\_ID** aus den Tabellen **EMPLOYEES** und **JOB\_HISTORY** mit der Klausel **ON** verknüpft. Immer wenn eine Personalnummer in der Tabelle **EMPLOYEES** einer Personalnummer in der Tabelle **JOB\_HISTORY** entspricht, wird die Zeile zurückgegeben. Der Tabellenalias ist erforderlich, um die übereinstimmenden Spaltennamen zu kennzeichnen.

Sie können mit der Klausel **ON** auch Spalten mit unterschiedlichen Namen verknüpfen. Die Klammersetzung um die verknüpften Spalten im Beispiel auf der Folie, **(e.employee\_id = j.employee\_id)**, ist optional. Damit funktioniert auch **ON e.employee\_id = j.employee\_id**.

**Hinweis:** Wenn Sie die Abfrage mit dem Symbol **Execute Statement** ausführen, hängt SQL Developer zur Unterscheidung der beiden **employee\_ids** "1" an.

## Left Outer Joins

- Ein Join zwischen zwei Tabellen, der alle übereinstimmenden Zeilen sowie die nicht übereinstimmenden Zeilen aus der linken Tabelle zurückgibt, wird als **LEFT OUTER JOIN** bezeichnet.
- Beispiel:

```
SELECT c.country_id, c.country_name, l.location_id, l.city
FROM countries c LEFT OUTER JOIN locations l
ON (c.country_id = l.country_id) ;
```

|   | COUNTRY_ID | COUNTRY_NAME             | LOCATION_ID | CITY                |
|---|------------|--------------------------|-------------|---------------------|
| 1 | CA         | Canada                   | 1800        | Toronto             |
| 2 | DE         | Germany                  | (null)      | (null)              |
| 3 | UK         | United Kingdom           | 2500        | Oxford              |
| 4 | US         | United States of America | 1400        | Southlake           |
| 5 | US         | United States of America | 1500        | South San Francisco |
| 6 | US         | United States of America | 1700        | Seattle             |



Copyright © 2013, Oracle und/oder verbundene Unternehmen. All rights reserved. Alle Rechte vorbehalten.

Diese Abfrage ruft alle Zeilen aus der Tabelle COUNTRIES (linke Tabelle) ab, auch wenn keine Übereinstimmung in der Tabelle LOCATIONS gegeben ist.

## Right Outer Joins

- Ein Join zwischen zwei Tabellen, der alle übereinstimmenden Zeilen sowie die nicht übereinstimmenden Zeilen aus der rechten Tabelle zurückgibt, wird als **RIGHT OUTER JOIN** bezeichnet.
- Beispiel:

```
SELECT e.last_name, d.department_id, d.department_name
FROM employees e RIGHT OUTER JOIN departments d
ON (e.department_id = d.department_id) ;
```

|     | LAST_NAME | DEPARTMENT_ID | DEPARTMENT_NAME |
|-----|-----------|---------------|-----------------|
| 1   | Whalen    | 10            | Administration  |
| 2   | Hartstein | 20            | Marketing       |
| 3   | Fay       | 20            | Marketing       |
| 4   | Davies    | 50            | Shipping        |
| ... |           |               |                 |

|            |                 |
|------------|-----------------|
| 18 Higgins | 110 Accounting  |
| 19 Gietz   | 110 Accounting  |
| 20 (null)  | 190 Contracting |

ORACLE®

Copyright © 2013, Oracle und/oder verbundene Unternehmen. All rights reserved. Alle Rechte vorbehalten.

Diese Abfrage ruft alle Zeilen aus der Tabelle DEPARTMENTS (rechte Tabelle) ab, auch wenn es keine Übereinstimmung dazu in der Tabelle EMPLOYEES gibt.

## Full Outer Joins

- Ein Join zwischen zwei Tabellen, der neben allen übereinstimmenden Zeilen auch die nicht übereinstimmenden Zeilen aus beiden Tabellen zurückgibt, wird als **FULL OUTER JOIN** bezeichnet.
- Beispiel:

```
SELECT e.last_name, d.department_id, d.manager_id,
 d.department_name
 FROM employees e FULL OUTER JOIN departments d
 WHERE (e.manager_id = d.manager_id) ;
```

|   | LAST_NAME | DEPARTMENT_ID | MANAGER_ID | DEPARTMENT_NAME |
|---|-----------|---------------|------------|-----------------|
| 1 | King      | (null)        | (null)     | (null)          |
| 2 | Kochhar   | 90            | 100        | Executive       |
| 3 | De Haan   | 90            | 100        | Executive       |
| 4 | Hunold    | (null)        | (null)     | (null)          |

...

|    |         |        |        |                |
|----|---------|--------|--------|----------------|
| 19 | Higgins | (null) | (null) | (null)         |
| 20 | Gietz   | 110    | 205    | Accounting     |
| 21 | (null)  | 190    | (null) | Contracting    |
| 22 | (null)  | 10     | 200    | Administration |

ORACLE

Copyright © 2013, Oracle und/oder verbundene Unternehmen. All rights reserved. Alle Rechte vorbehalten.

Diese Abfrage ruft alle Zeilen aus der Tabelle EMPLOYEES ab, auch wenn es keine Übereinstimmung dazu in der Tabelle DEPARTMENTS gibt. Außerdem werden alle Zeilen aus der Tabelle DEPARTMENTS abgerufen, unabhängig davon, ob Übereinstimmungen in der Tabelle EMPLOYEES vorhanden sind.

## Self-Joins – Beispiel

```
SELECT worker.last_name || ' works for '
 || manager.last_name
 FROM employees worker JOIN employees manager
 WHERE worker.manager_id = manager.employee_id
 ORDER BY worker.last_name;
```

| WORKER_LAST_NAME  "WORKSFOR'  MANAGER_LAST_NAME |
|-------------------------------------------------|
| 1 Abel works for Zlotkey                        |
| 2 Davies works for Mourgos                      |
| 3 De Haan works for King                        |
| 4 Ernst works for Hunold                        |
| 5 Fay works for Hartstein                       |
| 6 Gietz works for Higgins                       |
| 7 Grant works for Zlotkey                       |
| 8 Hartstein works for King                      |
| 9 Higgins works for Kochhar                     |

\*\*\*

ORACLE

Copyright © 2013, Oracle und/oder verbundene Unternehmen. All rights reserved. Alle Rechte vorbehalten.

Gelegentlich müssen Sie eine Tabelle mit sich selbst verknüpfen. Um für jeden Mitarbeiter den Namen des zuständigen Managers zu ermitteln, verknüpfen Sie die Tabelle EMPLOYEES mit sich selbst und führen damit einen Self Join durch. Im Beispiel auf der Folie wird die Tabelle EMPLOYEES mit sich selbst verknüpft. Um in der Klausel FROM zwei Tabellen zu simulieren, werden für dieselbe Tabelle (EMPLOYEES) zwei Aliasnamen verwendet: worker und manager.

In diesem Beispiel enthält die Klausel WHERE den Join, der wie folgt interpretiert wird: "wobei die Manager-ID für einen Mitarbeiter der Personalnummer des Managers entspricht".

## Cross Joins

- Ein CROSS JOIN ist ein JOIN-Vorgang, der das kartesische Produkt aus zwei Tabellen zurückgibt.
- Beispiel:

```
SELECT department_name, city
FROM department CROSS JOIN location;
```

| DEPARTMENT_NAME  | CITY                |
|------------------|---------------------|
| 1 Administration | Oxford              |
| 2 Administration | Seattle             |
| 3 Administration | South San Francisco |
| 4 Administration | Southlake           |
| 5 Administration | Toronto             |
| 6 Marketing      | Oxford              |
| 7 Marketing      | Seattle             |
| 8 Marketing      | South San Francisco |
| 9 Marketing      | Southlake           |
| 10 Marketing     | Toronto             |

...

ORACLE

Copyright © 2013, Oracle und/oder verbundene Unternehmen. All rights reserved. Alle Rechte vorbehalten.

Die Syntax für einen CROSS JOIN gibt das Kreuzprodukt an (auch als kartesisches Produkt bezeichnet). Ein Cross Join gibt das Kreuzprodukt aus zwei Relationen zurück und entspricht im Prinzip einer durch Kommas getrennten Oracle Database-Notation.

Zwischen den beiden Tabellen im CROSS JOIN geben Sie keine WHERE-Bedingungen an.

## Zusammenfassung

In diesem Anhang haben Sie Folgendes gelernt:

- Mithilfe von **SELECT**-Anweisungen Zeilen aus einzelnen oder mehreren Tabellen abrufen
- Mithilfe von **DDL**-Anweisungen die Struktur von Objekten ändern
- Mithilfe von **DML**-Anweisungen Daten in vorhandenen Schemaobjekten abfragen oder bearbeiten
- Mithilfe von Anweisungen zur Transaktionskontrolle die über **DML**-Anweisungen vorgenommenen Änderungen verwalten
- Mithilfe von Joins Daten aus mehreren Tabellen anzeigen



Copyright © 2013, Oracle und/oder verbundene Unternehmen. All rights reserved. Alle Rechte vorbehalten.

In diesem Anhang wurden häufig verwendete SQL-Anweisungen und -Befehlen behandelt, darunter DDL-, DML- und Transaktionskontrollanweisungen sowie Joins.

# PL/SQL-Code verwalten

ORACLE

Copyright © 2013, Oracle und/oder verbundene Unternehmen. All rights reserved. Alle Rechte vorbehalten.

## Ziele

**Nach Beendigung dieses Anhangs haben Sie folgende Ziele erreicht:**

- **Conditional Compilation beschreiben und verwenden**
- **PL/SQL-Quellcode mit dynamischer Obfuscation und dem Utility `wrap` verbergen**



Copyright © 2013, Oracle und/oder verbundene Unternehmen. All rights reserved. Alle Rechte vorbehalten.

Dieser Anhang vermittelt eine Einführung zu Conditional Compilation sowie zum Verschlüsseln (oder Wrappen) von PL/SQL-Code.

# Agenda

- **Conditional Compilation**
- **PL/SQL-Code verschlüsseln**

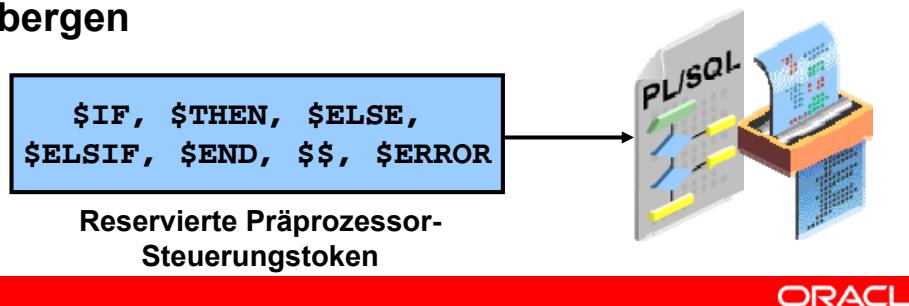
ORACLE

Copyright © 2013, Oracle und/oder verbundene Unternehmen. All rights reserved. Alle Rechte vorbehalten.

## Conditional Compilation

**Ermöglicht die Anpassung der Funktionalität in einer PL/SQL-Anwendung ohne Entfernen von Quellcode:**

- **Neueste Funktionalität mit dem aktuellen Datenbankrelease nutzen oder die neuen Features deaktivieren, um die Anwendung für ein älteres Release der Datenbank auszuführen**
- **Debugging- oder Tracing-Funktionalität in der Entwicklungsumgebung aktivieren und bei der Ausführung der Anwendung an einer Produktionssite verbergen**



Copyright © 2013, Oracle und/oder verbundene Unternehmen. All rights reserved. Alle Rechte vorbehalten.

Conditional Compilation ermöglicht die selektive Einbeziehung von Code auf der Grundlage von Bedingungswerten, die während der Kompilierung ausgewertet werden. So können Sie mit Conditional Compilation beispielsweise festlegen, welche PL/SQL-Features in den Versionen einer PL/SQL-Anwendung für verschiedene Datenbankreleases verwendet werden sollen. Während in der Anwendung für ein neues Datenbankrelease die neuesten PL/SQL-Features ausgeführt werden können, ist über Bedingungen geregelt, dass die gleiche Anwendung auch mit einem früheren Datenbankrelease kompatibel ist. Conditional Compilation ist auch hilfreich, wenn Debuggingprozeduren in einer Entwicklungsumgebung ausgeführt werden, die entsprechenden Routinen in der Produktionsumgebung aber deaktiviert sein sollen.

### Vorteile von Conditional Compilation

- Unterstützung mehrerer Versionen desselben Programms in nur einem Quellcode
- Einfache Wartung und leichtes Debugging von Code
- Problemlose Migration von Code zu einem anderen Datenbankrelease

## Conditional Compilation – Funktionsweise



**ORACLE**

Copyright © 2013, Oracle und/oder verbundene Unternehmen. All rights reserved. Alle Rechte vorbehalten.

Sie können Conditional Compilation verwenden, indem Sie Anweisungen in Ihre PL/SQL-Quellprogramme einbetten. Wenn das PL/SQL-Programm zur Kompilierung weitergeleitet wird, wertet ein Präprozessor die Anweisungen aus und wählt die Teile des Programms, die kompiliert werden sollen. Der entsprechende Quellcode für das Programm wird dann zur Kompilierung an den Compiler übergeben.

In Abfrageanweisungen werden mit dem Token `$$` Abfragen über die Kompilierungsumgebung ausgeführt, etwa zum Wert des Initialisierungsparameters `PLSQL_CCFLAGS` oder `PLSQL_OPTIMIZE_LEVEL` im PL/SQL-Compiler für die Einheit, die kompiliert werden soll. Diese Anweisung kann gemeinsam mit der bedingten Auswahlanweisung verwendet werden, um die Programmteile für die Kompilierung zu wählen.

Mit Auswahlanweisungen können Sie Abfrageanweisungen oder statische Packagekonstanten testen. Dazu verwenden Sie das Konstrukt `$IF`, um die einzelnen Codeabschnitte zu prüfen und bei Erfüllung der Bedingung zu kompilieren.

Fehleranweisungen geben mithilfe des Tokens `$ERROR` einen Kompilierungsfehler aus, wenn im Rahmen von Conditional Compilation eine unerwartete Bedingung auftritt.

Das Package `DBMS_DB_VERSION` gibt Konstanten für Datenbankversion und Release an, die für Conditional Compilation verwendet werden können.

Das Package `DBMS_PREPROCESSOR` stellt Unterprogramme für den Zugriff auf den nachverarbeiteten Quelltext bereit, der gemäß den Conditional Compilation-Anweisungen in einer PL/SQL-Einheit gewählt wurde.

# Auswahlanweisungen

```
$IF <Boolean-expression> $THEN Text
$ELSIF <Boolean-expression> $THEN Text
. . .
$ELSE Text
$END
```

```
DECLARE
CURSOR cur IS SELECT employee_id FROM
employees WHERE
$IF myapp_tax_package.new_tax_code $THEN
 salary > 20000;
$ELSE
 salary > 50000;
$END
BEGIN
 OPEN cur;
. . .
END;
```

**ORACLE**

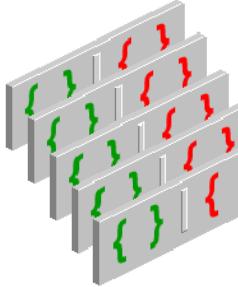
Copyright © 2013, Oracle und/oder verbundene Unternehmen. All rights reserved. Alle Rechte vorbehalten.

Die bedingte Auswahlanweisung entspricht optisch und funktional dem PL/SQL-Mechanismus IF-THEN-ELSE. Stößt der Präprozessor auf ein \$THEN, prüft er, ob der Text zwischen \$IF und \$THEN ein statischer Ausdruck ist. Wenn dies der Fall ist und die Auswertung TRUE ergibt, wird der PL/SQL-Programmtext zwischen \$THEN und \$ELSE (oder \$ELSIF) für die Kompilierung gewählt.

Bei der Formulierung der Auswahlbedingung (des Ausdrucks zwischen \$IF und \$THEN) können Sie auf Konstanten, die in einem anderen Package definiert sind, und/oder auf Abfrageanweisungen verweisen.

Im Beispiel auf der Folie trifft die Auswahlanweisung auf der Grundlage des Wertes von MYAPP\_TAX\_PACKAGE.NEW\_TAX\_CODE eine Wahl zwischen zwei Versionen des Cursors (cur). Lautet der Wert TRUE, werden Mitarbeiter gewählt, für die salary > 20000 gilt. Andernfalls werden Mitarbeiter gewählt, für die salary > 50000 gilt.

## Vordefinierte und benutzerdefinierte Abfrageanweisungen



```

PLSQL_CCFLAGS
PLSQL_CODE_TYPE
PLSQL_OPTIMIZE_LEVEL
PLSQL_WARNINGS
NLS_LENGTH_SEMANTICS
PLSQL_LINE
PLSQL_UNIT

```

### Vordefinierte Abfrageanweisungen

```
PLSQL_CCFLAGS = 'plsql_ccflags:true,debug:true,debug:0';
```

### Benutzerdefinierte Abfrageanweisungen

**ORACLE**

Copyright © 2013, Oracle und/oder verbundene Unternehmen. All rights reserved. Alle Rechte vorbehalten.

Abfrageanweisungen können vordefiniert oder benutzerdefiniert sein. Der Versuch, eine Abfrageanweisung aufzulösen, erfolgt mit Conditional Compilation folgendermaßen:

1. Die ID wird als Abfrageanweisung in Form von `$$id` für den Suchschlüssel verwendet.
  2. Der Algorithmus wird in zwei Durchgängen wie folgt verarbeitet:
    - a. Die Zeichenfolge im Initialisierungsparameter `PLSQL_CCFLAGS` wird von rechts nach links nach ID nach einem entsprechenden Namen durchsucht. (Die Groß-/Kleinschreibung wird nicht beachtet.) Wird eine ID gefunden, ist die Verarbeitung abgeschlossen.
    - b. Die vordefinierten Abfrageanweisungen werden durchsucht. Wird eine ID gefunden, ist die Verarbeitung abgeschlossen.
  3. Wenn `$$ID` nicht in einen Wert aufgelöst werden kann, wird die Warnmeldung `PLW-6003` ausgegeben, sofern der Quelltext nicht gewrappt wurde. Das Literal `NULL` wird als Ersatzwert für undefinierte Abfrageanweisungen verwendet.
- Hinweis:** Bei gewrappedem PL/SQL-Code ist die Warnmeldung deaktiviert, sodass die undefinierte Abfrageanweisung nicht angegeben wird.

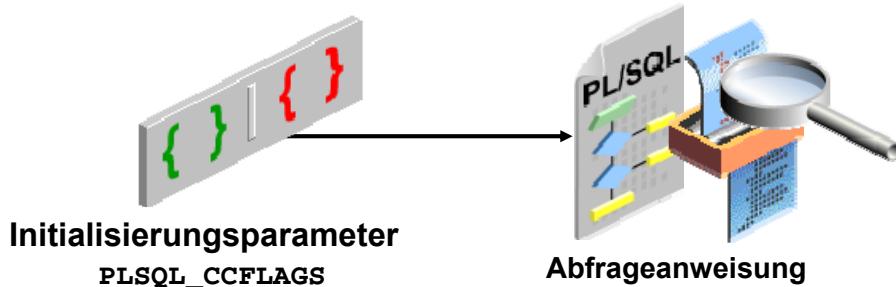
Im Beispiel auf der Folie verfügt `$$debug` über den Wert `0` und `$$plsql_ccflags` über den Wert `TRUE`. Beachten Sie, dass `$$plsql_ccflags` innerhalb des Wertes für den Compiler-Parameter `PLSQL_CCFLAGS` in den benutzerdefinierten Wert `plsql_ccflags` aufgelöst wird. Das liegt daran, dass eine benutzerdefinierte Anweisung eine vordefinierte Anweisung außer Kraft setzt.

## Parameter PLSQL\_CCFLAGS und Abfrageanweisungen

**Mit dem Parameter PLSQL\_CCFLAGS steuern Sie Conditional Compilation für jede PL/SQL-Library-Einheit gesondert.**

```
PLSQL_CCFLAGS = '<v1>:<c1>,<v2>:<c2>,...,<vn>:<cn>'
```

```
ALTER SESSION SET
PLSQL_CCFLAGS = 'plsql_ccflags:true, debug:true, debug:0';
```



ORACLE

Copyright © 2013, Oracle und/oder verbundene Unternehmen. All rights reserved. Alle Rechte vorbehalten.

In Oracle Database 10g Release 2 wurde mit PLSQL\_CCFLAGS ein neuer Oracle-Initialisierungsparameter für Conditional Compilation eingeführt. Mit diesem dynamischen Parameter können Sie Name/Wert-Paare einrichten. Die Namen (sogenannte Flag-Namen) können dann in Abfrageanweisungen referenziert werden. PLSQL\_CCFLAGS stellt einen Mechanismus bereit, mit dem PL/SQL-Programmierer Conditional Compilation für jede PL/SQL-Library-Einheit gesondert steuern können.

### Werte

- **vi:** Hat die Form eines uneingeschränkten PL/SQL-Bezeichners ohne Anführungszeichen und kann ein reserviertes Wort oder Schlüsselwort sein. Im Text muss die Groß- und Kleinschreibung nicht beachtet werden. Jeder Wert wird als Flag oder Flag-Name bezeichnet. Jeder vi kann mehrmals in der Zeichenfolge vorkommen. Bei jedem Auftreten kann ein anderer Flag-Wert verwendet werden, und die Flag-Werte können verschiedene Typen aufweisen.
- **ci:** Kann einer der folgenden Typen sein:
  - ein boolesches PL/SQL-Literal
  - ein PLS\_INTEGER-Literal
  - das Literal NULL (Standard). Im Text muss die Groß- und Kleinschreibung nicht beachtet werden. Jeder Wert wird als Flag-Wert bezeichnet und entspricht einem Flag-Namen.

## Einstellung für den Initialisierungsparameter PLSQL\_CCFLAGS anzeigen

```
SELECT name, type, plsql_ccflags
FROM user_plsql_object_settings
```

Results:

| NAME                  | TYPE         | PLSQL_CCFLAGS                           |
|-----------------------|--------------|-----------------------------------------|
| 1 DEPT_PKG            | PACKAGE      | (null)                                  |
| 2 DEPT_PKG            | PACKAGE BODY | (null)                                  |
| 3 TAXES_PKG           | PACKAGE      | (null)                                  |
| 4 TAXES_PKG           | PACKAGE BODY | (null)                                  |
| 5 EMP_PKG             | PACKAGE      | (null)                                  |
| 6 EMP_PKG             | PACKAGE BODY | (null)                                  |
| 7 SECURE_DML          | PROCEDURE    | (null)                                  |
| 8 SECURE_EMPLOYEES    | TRIGGER      | (null)                                  |
| 9 ADD_JOB_HISTORY     | PROCEDURE    | plsql_ccflags:true, debug:true, debug:0 |
| 10 UPDATE_JOB_HISTORY | TRIGGER      | (null)                                  |

...

Copyright © 2013, Oracle und/oder verbundene Unternehmen. All rights reserved. Alle Rechte vorbehalten.

Die Einstellungen für ein PL/SQL-Objekt zeigen Sie mit den Data Dictionary Views USER | ALL | DBA\_PLSQL\_OBJECT\_SETTINGS an.

Sie können für PLSQL\_CCFLAGS jeden zulässigen Wert definieren. Oracle empfiehlt jedoch, diesen Parameter zur Steuerung von Conditional Compilation für Debugging- oder Tracing-Code zu verwenden.

Die Flag-Namen können auf jeden beliebigen Bezeichner eingestellt werden, einschließlich reserverter Wörter und Schlüsselwörter. Bei den Werten muss es sich um die Literale TRUE, FALSE oder NULL bzw. ein PLSQL\_INTEGER-Literal handeln. Bei den Flag-Namen und Werten muss die Groß-/Kleinschreibung nicht beachtet werden. Der Parameter PLSQL\_CCFLAGS ist ein PL/SQL-Compiler-Parameter (wie andere Compiler-Parameter) und wird mit der PL/SQL-Programmeinheit gespeichert. Wird also das PL/SQL-Programm später mit der Klausel REUSE SETTINGS rekompiliert (Beispiel: ALTER PACKAGE ...REUSE SETTINGS), wird derselbe Wert für PLSQL\_CCFLAGS für die Rekompilierung verwendet. Da der Parameter PLSQL\_CCFLAGS für jede PL/SQL-Einheit auf einen anderen Wert eingestellt werden kann, stellt er eine praktische Methode dar, um Conditional Compilation auf Basis der einzelnen Programmeinheit zu steuern.

## Parameter PLSQL\_CCFLAGS und Abfrageanweisungen – Beispiel

```
ALTER SESSION SET PLSQL_CCFLAGS = 'Tracing:true';
CREATE OR REPLACE PROCEDURE P IS
BEGIN
 $IF $$tracing $THEN
 DBMS_OUTPUT.PUT_LINE ('TRACING');
 $END
END P;
```

```
ALTER SESSION SET succeeded.
PROCEDURE P Compiled.
```

```
SELECT name, plsql_ccflags
FROM USER_PLSQL_OBJECT_SETTINGS
WHERE name = 'P';
```

Results:

|   | NAME | PLSQL_CCFLAGS |
|---|------|---------------|
| 1 | P    | Tracing:true  |

ORACLE

Copyright © 2013, Oracle und/oder verbundene Unternehmen. All rights reserved. Alle Rechte vorbehalten.

Im Beispiel auf der Folie wird der Parameter eingestellt und danach die Prozedur erstellt. Die Einstellung wird mit jeder PL/SQL-Einheit gespeichert.

## Benutzerdefinierte Fehler mithilfe von Conditional Compilation-Fehleranweisungen auslösen

```
$ERROR varchar2_static_expression $END

ALTER SESSION SET Plsql_CCFlags = ' Trace_Level:3 '
/
CREATE PROCEDURE P IS
BEGIN
 $IF $$Trace_Level = 0 $THEN ...
 $ELSIF $$Trace_Level = 1 $THEN ...
 $ELSIF $$Trace_Level = 2 $THEN ...
 $Else $error 'Bad: '||$$Trace_Level $END -- error
 -- directive
 $END -- selection directive ends
END P;

SHOW ERRORS
Errors for PROCEDURE P:
LINE/COL ERROR

6/9 PLS-00179: $ERROR: Bad: 3
```

**ORACLE**

Copyright © 2013, Oracle und/oder verbundene Unternehmen. All rights reserved. Alle Rechte vorbehalten.

Die Fehleranweisung \$ERROR löst einen benutzerdefinierten Fehler aus und weist folgende Form auf:

```
$ERROR varchar2_static_expression $END
```

**Hinweis:** varchar2\_static\_expression muss ein statischer Ausdruck vom Typ VARCHAR2 sein.

# Statische Ausdrücke und Conditional Compilation

- **Statische Ausdrücke vom Typ Boolean:**
  - TRUE, FALSE, NULL, IS NULL, IS NOT NULL
  - >, <, >=, <=, =, <>, NOT, AND, OR
- **Statische Ausdrücke vom Typ PLS\_INTEGER:**
  - -2147483648 bis 2147483647, NULL
- **Beispiele für statische Ausdrücke vom Typ VARCHAR2:**
  - ||, NULL, TO\_CHAR
- **Statische Konstanten:**

```
static_constant CONSTANT datatype := static_expression;
```

**ORACLE**

Copyright © 2013, Oracle und/oder verbundene Unternehmen. All rights reserved. Alle Rechte vorbehalten.

Wie bereits beschrieben, verarbeitet ein Präprozessor Bedingungsanweisungen, bevor die eigentliche Kompilierung beginnt. Daher sind in Conditional Compilation-Anweisungen nur Ausdrücke zugelassen, die bei der Kompilierung vollständig ausgewertet werden können. Alle Ausdrücke, die Variablen oder Funktionen referenzieren, für die PL/SQL ausgeführt werden muss, sind bei der Kompilierung nicht verfügbar und können nicht ausgewertet werden.

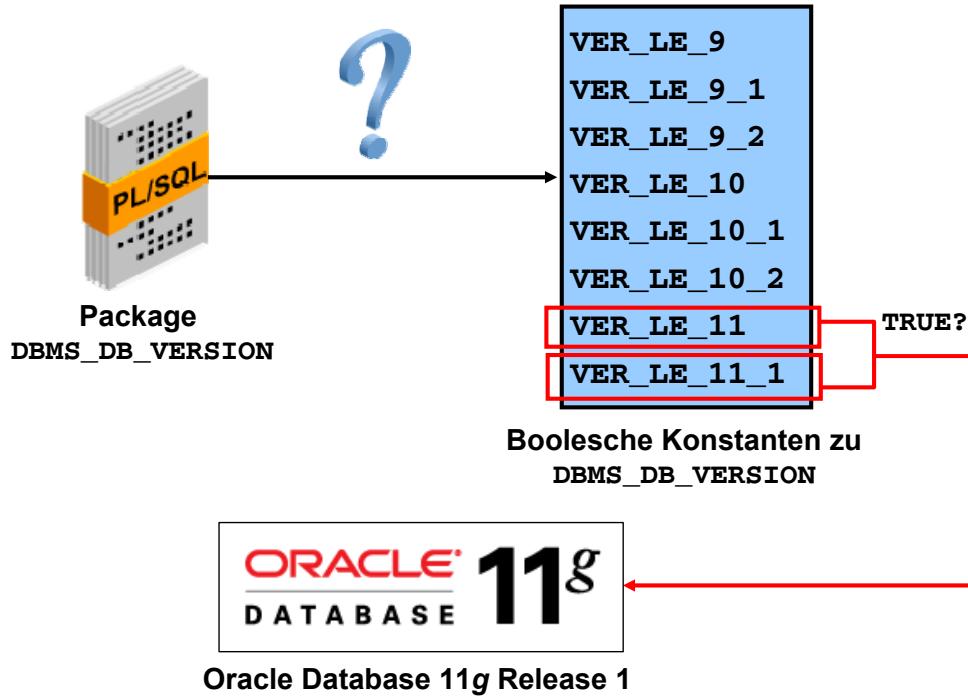
PL/SQL-Ausdrücke, die in Conditional Compilation-Anweisungen zulässig sind, werden als "statische Ausdrücke" bezeichnet. Statische Ausdrücke sind sorgfältig definiert, um zu gewährleisten, dass bei der automatischen Rekompilierung einer Einheit ohne Änderungen an den zugrunde liegenden Werten die Ausdrücke identisch ausgewertet werden und derselbe Quellcode kompiliert wird.

Im Allgemeinen werden statische Ausdrücke aus drei Quellen erstellt:

- Abfrageanweisungen mit der Markierung \$\$
- In PL/SQL-Packages wie DBMS\_DB\_VERSION definierte Konstanten. Diese Werte können mit den üblichen PL/SQL-Vorgängen kombiniert und verglichen werden.
- Literale wie TRUE, FALSE, 'CA', 123 und NULL

Statische Ausdrücke können auch Vorgänge enthalten, die Vergleiche, logische boolesche Vorgänge (wie OR und AND) oder Verkettungen von statischen Zeichenausdrücken umfassen.

## Package DBMS\_DB\_VERSION – boolesche Konstanten



Copyright © 2013, Oracle und/oder verbundene Unternehmen. All rights reserved. Alle Rechte vorbehalten.

In Oracle Database 10g Release 2 wurde das Package `DBMS_DB_VERSION` eingeführt. Die Angaben zur Oracle-Datenbankversion und den Releasenummern aus diesem Package sind hilfreich, um eine einfache Auswahl für Conditional Compilation zu treffen.

Die Konstanten repräsentieren eine boolesche Bedingung, die bei Auswertung kleiner oder gleich der Version und dem Release sein muss (soweit vorhanden).

### Beispiel

`VER_LE_11` zeigt an, dass die Datenbankversion  $\leq 11$  ist. Die Werte der Konstanten lauten entweder `TRUE` oder `FALSE`. Beispiel: In einer Oracle-Datenbank der Version 11g Release 1 ergeben `VER_LE_11` und `VER_LE_11_1` den Wert `TRUE` und alle anderen Konstanten den Wert `FALSE`.

## Package DBMS\_DB\_VERSION – Konstanten

| Name        | Beschreibung                   |
|-------------|--------------------------------|
| VER_LE_9    | Version <= 9                   |
| VER_LE_9_1  | Version <= 9 und Release <= 1  |
| VER_LE_9_2  | Version <= 9 und Release <= 2  |
| VER_LE_10   | Version <= 10                  |
| VER_LE_10_1 | Version <= 10 und Release <= 1 |
| VER_LE_10_2 | Version <= 10 und Release <= 2 |
| VER_LE_11   | Version <= 11                  |
| VER_LE_11_1 | Version <= 11 und Release <= 1 |

**ORACLE**

Copyright © 2013, Oracle und/oder verbundene Unternehmen. All rights reserved. Alle Rechte vorbehalten.

Package für Oracle Database 11g Release 1:

```

PACKAGE DBMS_DB_VERSION IS
 VERSION CONSTANT PLS_INTEGER := 11; -- RDBMS version
 -- number
 RELEASE CONSTANT PLS_INTEGER := 1; -- RDBMS release
 -- number
 ver_le_9_1 CONSTANT BOOLEAN := FALSE;
 ver_le_9_2 CONSTANT BOOLEAN := FALSE;
 ver_le_9 CONSTANT BOOLEAN := FALSE;
 ver_le_10_1 CONSTANT BOOLEAN := FALSE;
 ver_le_10_2 CONSTANT BOOLEAN := FALSE;
 ver_le_10 CONSTANT BOOLEAN := FALSE;
 ver_le_11_1 CONSTANT BOOLEAN := TRUE;
 ver_le_11 CONSTANT BOOLEAN := TRUE;
END DBMS_DB_VERSION;
```

Das Package DBMS\_DB\_VERSION enthält unterschiedliche Konstanten für die verschiedenen Releases der Oracle-Datenbank. Für Oracle Database 11g Release 1 verwendet DBMS\_DB\_VERSION die auf der Folie gezeigten Konstanten.

## Conditional Compilation und Datenbankversionen – Beispiel

```

ALTER SESSION SET PLSQL_CCFLAGS = 'my_debug:FALSE, my_tracing:FALSE';
CREATE PACKAGE my_pkg AS
 SUBTYPE my_real IS
 -- Check the database version, if >= 10g, use BINARY_DOUBLE data type,
 -- else use NUMBER data type
 $IF DBMS_DB_VERSION.VERSION < 10 $THEN NUMBER;
 $ELSE BINARY_DOUBLE;
 $END
 my_pi my_real; my_e my_real;
 END my_pkg;
/
CREATE PACKAGE BODY my_pkg AS
BEGIN
 $IF DBMS_DB_VERSION.VERSION < 10 $THEN
 my_pi := 3.14016408289008292431940027343666863227;
 my_e := 2.71828182845904523536028747135266249775;
 $ELSE
 my_pi := 3.14016408289008292431940027343666863227d;
 my_e := 2.71828182845904523536028747135266249775d;
 $END
END my_pkg;
/

```

**ORACLE**

Copyright © 2013, Oracle und/oder verbundene Unternehmen. All rights reserved. Alle Rechte vorbehalten.

Dieses Beispiel zeigt auch die Verwendung des Parameters `PLSQL_CCFLAGS`. Zuerst stellen Sie das Parameter-Flag `PLSQL_CCFLAGS` ein, um Debuggingcode und Tracing-Informationen anzuzeigen.

Im Beispiel auf dieser und der nächsten Folie wird Conditional Compilation verwendet, um Code für Datenbankversionen anzugeben. Mithilfe von Conditional Compilation wird ermittelt, ob der Datentyp `BINARY_DOUBLE` in der Datenbank bei Berechnungen für PL/SQL-Einheiten verwendet werden kann. Dies ist nur in Oracle Database 10g oder höher möglich. Wenn Sie mit Oracle Database 10g arbeiten, lautet der Datentyp für `my_real BINARY_DOUBLE`; andernfalls hat `my_real` den Datentyp `NUMBER`.

In der Spezifikation des neuen Packages (`my_pkg`) wird Conditional Compilation eingesetzt, um die Datenbankversion zu prüfen. In der Bodydefinition des Packages werden mit Conditional Compilation die Werte für `my_pi` und `my_e` für künftige Berechnungen auf Grundlage der Datenbankversion eingestellt.

Das Ergebnis des Beispielcodes auf der Folie lautet:

```

PACKAGE my_pkg compiled
PACKAGE BODY my_pkg compiled
session SET altered.

```

## Conditional Compilation und Datenbankversionen – Beispiel

```

CREATE OR REPLACE PROCEDURE circle_area(p_radius my_pkg.my_real) IS
 v_my_area my_pkg.my_real;
 v_my_datatype VARCHAR2(30);
BEGIN
 v_my_area := my_pkg.my_pi * p_radius * p_radius;
 DBMS_OUTPUT.PUT_LINE('Radius: ' || TO_CHAR(p_radius)
 || ' Area: ' || TO_CHAR(v_my_area));
 $IF $$my_debug $THEN -- if my_debug is TRUE, run some debugging code
 SELECT DATA_TYPE INTO v_my_datatype FROM USER_ARGUMENTS
 WHERE OBJECT_NAME = 'CIRCLE_AREA' AND ARGUMENT_NAME = 'P_RADIUS';
 DBMS_OUTPUT.PUT_LINE('Datatype of the RADIUS argument is: ' ||
 v_my_datatype);
 $END
END; /

```

PROCEDURE circle\_area compiled

CALL circle\_area(50); -- Using Oracle Database 11g Release 2

CALL circle\_area(50) succeeded.  
Radius: 5.0E+001 Area: 7.8504102072252062E+003

**ORACLE**

Copyright © 2013, Oracle und/oder verbundene Unternehmen. All rights reserved. Alle Rechte vorbehalten.

Im Beispiel auf der Folie wird eine neue Prozedur `circle_area` definiert. Diese Prozedur berechnet die Fläche eines Kreises auf Grundlage der Werte in den Variablen des auf der vorherigen Seite definierten Packages `my_pkg`. Die Prozedur enthält einen informalen Parameter: `radius`.

Die Prozedur deklariert zwei Variablen:

- `my_area` mit demselben Datentyp wie `my_real` in `my_pkg`
- `my_datatype` mit dem Datentyp `VARCHAR2(30)`

Im Body der Prozedur wird `my_area` gleichgesetzt mit dem in `my_pkg` eingestellten Wert von `my_pi` multipliziert mit dem Wert, der als Radius an die Prozedur übergeben wird. In einer Meldung werden der Radius und die Kreisfläche angegeben, wie im zweiten Codebeispiel auf der Folie gezeigt.

**Hinweis:** Wenn Sie für `my_debug` den Wert `TRUE` einstellen möchten, können Sie diese Änderung nur für die Prozedur `circle_area` vornehmen, indem Sie die Klausel `REUSE SETTINGS` wie folgt angeben:

```
ALTER PROCEDURE circle_area COMPILE PLSQL_CCFLAGS = 'my_debug:TRUE'
REUSE SETTINGS;
```

# Quelltext mit DBMS\_PREPROCESSOR-Prozeduren ausgeben oder abrufen

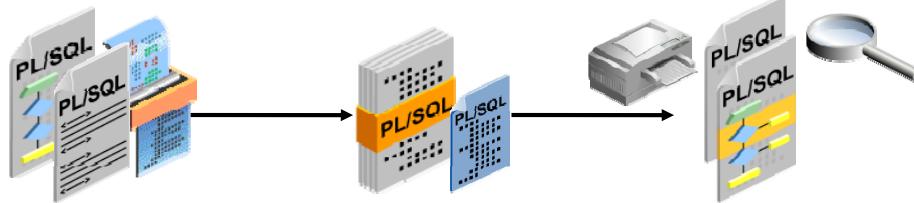
**CALL**

```
DBMS_PREPROCESSOR.PRINT_POST_PROCESSED_SOURCE('PACKAGE'
, 'ORA61', 'MY_PKG');
```

```
CALL DBMS_PREPROCESSOR.PRINT_POST_PROCESSED_SOURCE('PACKAGE', succeeded.
PACKAGE my_pkg AS
SUBTYPE my_real IS
 -- Check the database version, if >= 10g, use BINARY_DOUBLE data type,
 -- else use NUMBER data type

 BINARY_DOUBLE;

 my_pi my_real; my_e my_real;
END my_pkg;
```



ORACLE

Copyright © 2013, Oracle und/oder verbundene Unternehmen. All rights reserved. Alle Rechte vorbehalten.

DBMS\_PREPROCESSOR-Unterprogramme können nach Verarbeitung der Conditional Compilation-Anweisungen den nachverarbeiteten Quelltext einer PL/SQL-Einheit ausgeben oder abrufen. Dieser nachverarbeitete Text ist die eigentliche Quelle, die für die Kompilierung einer gültigen PL/SQL-Einheit verwendet wird. Das Beispiel auf der Folie zeigt die Ausgabe der nachverarbeiteten Form von my\_pkg mit der Prozedur PRINT\_POST\_PROCESSED\_SOURCE.

Wird my\_pkg in Oracle Database 10g oder höher mit dem Account HR kompiliert, ergibt sich daraus die auf der Folie gezeigte Ausgabe.

Mit PRINT\_POST\_PROCESSED\_SOURCE wird nicht gewählter Text entfernt. Die nicht im nachverarbeiteten Text enthaltenen Codezeilen werden entfernt. Die Prozedur PRINT\_POST\_PROCESSED\_SOURCE verfügt über folgende Argumente: Objekttyp, Schemaname (Teilnehmeraccount ORA61) und Objektname.

**Hinweis:** Weitere Informationen zum Package DBMS\_PREPROCESSOR finden Sie im Dokument *Oracle Database PL/SQL Packages and Types Reference*.

# Agenda

- Conditional Compilation
- **PL/SQL-Code verschlüsseln**

ORACLE

Copyright © 2013, Oracle und/oder verbundene Unternehmen. All rights reserved. Alle Rechte vorbehalten.

## Obfuscation

- **Obfuscation (oder Wrapping) einer PL/SQL-Einheit bezeichnet den Prozess, mit dem der PL/SQL-Quellcode verborgen wird.**
- **Wrapping kann mit dem Utility `wrap` sowie mit DBMS\_DDL-Unterprogrammen erfolgen.**
- **Das Utility `wrap` wird über die Befehlszeile ausgeführt. Es verarbeitet SQL-Eingabedateien, etwa SQL\*Plus-Installationsskripte.**
- **Die DBMS\_DDL-Unterprogramme wrappen eine einzelne PL/SQL-Einheit (wie einen einzelnen Befehl `CREATE PROCEDURE`), die dynamisch generiert wurde.**

ORACLE

Copyright © 2013, Oracle und/oder verbundene Unternehmen. All rights reserved. Alle Rechte vorbehalten.

### Hinweis:

Weitere Informationen über Obfuscation finden Sie im Dokument *Oracle Database PL/SQL Language Reference*.

## Obfuscation – Vorteile

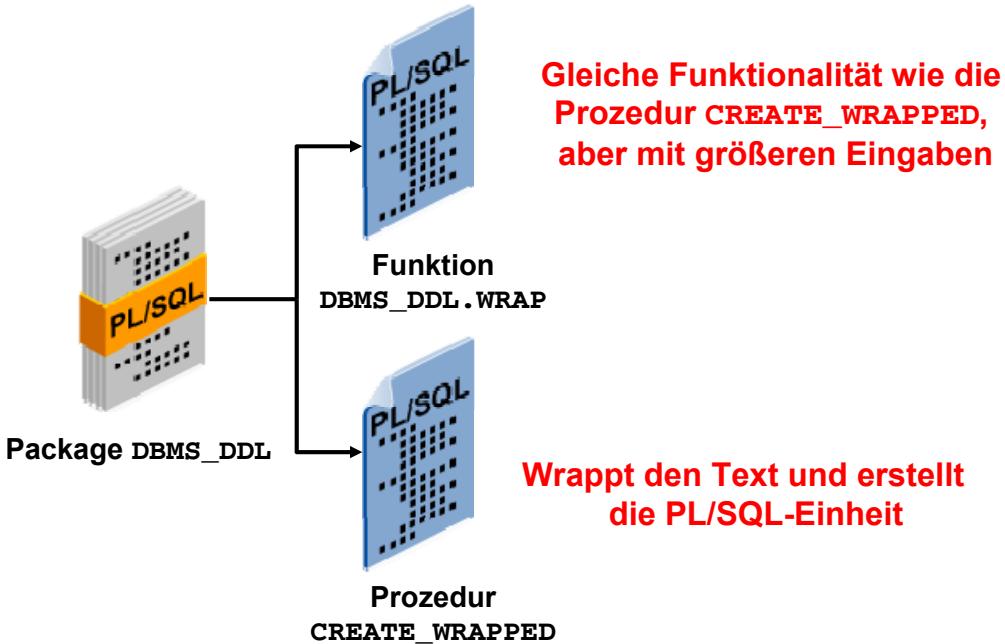
- Ihr Quellcode ist für andere nicht einsehbar.
- Der Quellcode wird nicht in den Data Dictionary Views `USER_SOURCE`, `ALL_SOURCE` oder `DBA_SOURCE` angezeigt.
- SQL\*Plus kann die mit Obfuscation verschlüsselten Quelldateien verarbeiten.
- Die Utilitys `Import` und `Export` unterstützen gewrappte Dateien.



ORACLE

Copyright © 2013, Oracle und/oder verbundene Unternehmen. All rights reserved. Alle Rechte vorbehalten.

## Dynamische Obfuscation – Neuerungen gegenüber Oracle 10g



**ORACLE**

Copyright © 2013, Oracle und/oder verbundene Unternehmen. All rights reserved. Alle Rechte vorbehalten.

### Prozedur CREATE\_WWRAPPED

Funktionsweise:

1. Die Prozedur nimmt als Eingabe eine einzelne Anweisung vom Typ CREATE OR REPLACE zur Erstellung einer PL/SQL-Packagespezifikation, eines Packagebodys, einer Funktion, Prozedur, Typspezifikation oder eines Typbodys an.
2. Die Prozedur generiert eine CREATE OR REPLACE-Anweisung, in der der PL/SQL-Quelltext verschlüsselt wurde.
3. Die Prozedur führt die generierte Anweisung aus.

### Funktion WRAP

Funktionsweise:

1. Die Prozedur nimmt als Eingabe eine Anweisung vom Typ CREATE OR REPLACE zur Erstellung einer PL/SQL-Packagespezifikation, eines Packagebodys, einer Funktion, Prozedur, Typspezifikation oder eines Typbodys an.
2. Die Prozedur gibt eine CREATE OR REPLACE-Anweisung zurück, in der die PL/SQL-Einheit verschlüsselt wurde.

## Nicht verschlüsselter PL/SQL-Code – Beispiel

```
SET SERVEROUTPUT ON
BEGIN -- The ALL_SOURCE view family shows source code
 EXECUTE IMMEDIATE '
 CREATE OR REPLACE PROCEDURE P1 IS
 BEGIN
 DBMS_OUTPUT.PUT_LINE (''I am not wrapped'');
 END P1;
 ';
END;
/
CALL p1();
```

```
anonymous block completed
CALL p1() succeeded.
I'm not wrapped
```

```
SELECT text FROM user_source
WHERE name = 'P1' ORDER BY line;
```

| TEXT                                             |
|--------------------------------------------------|
| 1 PROCEDURE P1 IS                                |
| 2   BEGIN                                        |
| 3     DBMS_OUTPUT.PUT_LINE ('I am not wrapped'); |
| 4   END P1;                                      |

ORACLE

Copyright © 2013, Oracle und/oder verbundene Unternehmen. All rights reserved. Alle Rechte vorbehalten.

Im ersten Beispiel auf der Folie wird die Anweisung EXECUTE IMMEDIATE verwendet, um die Prozedur P1 zu erstellen. Der Code in der erstellten Prozedur ist nicht gewrappt. Der Code ist somit nicht verborgen, wenn Sie den Prozedurcode mit einer View aus der View-Familie ALL\_SOURCE anzeigen (siehe Folie).

## Verschlüsselter PL/SQL-Code – Beispiel

```
BEGIN -- ALL_SOURCE view family obfuscates source code
DBMS_DDL.CREATE_WRAPPED (
 CREATE OR REPLACE PROCEDURE P1 IS
 BEGIN
 DBMS_OUTPUT.PUT_LINE (''I am wrapped now'');
 END P1;
);
END;
/
CALL p1();
```

anonymous block completed  
p1 ) succeeded.

```
SELECT text FROM user_source
WHERE name = 'P1' ORDER BY line;
```

| TEXT                 |
|----------------------|
| -----                |
| PROCEDURE P1 wrapped |
| a000000              |
| b2                   |
| abcd                 |

...

**ORACLE**

Copyright © 2013, Oracle und/oder verbundene Unternehmen. All rights reserved. Alle Rechte vorbehalten.

Im Beispiel auf der Folie wird die Packageprozedur DBMS\_DDL.CREATE\_WRAPPED verwendet, um die Prozedur P1 zu erstellen. Der Code ist verschlüsselt, wenn Sie den Prozedurcode mit einer View aus der View-Familie ALL\_SOURCE anzeigen (siehe nächste Seite). Wie in der Ausgabe für den Befehl auf der Folie zu sehen, ist der Quellcode in Views vom Typ \*\_SOURCE gewrappt bzw. verborgen, sodass andere die Codedetails nicht anzeigen können.

## Dynamische Obfuscation – Beispiel

```

SET SERVEROUTPUT ON

DECLARE
 c_code CONSTANT VARCHAR2(32767) :=
 ' CREATE OR REPLACE PROCEDURE new_proc AS
 v_VDATE DATE;
 BEGIN
 v_VDATE := SYSDATE;
 DBMS_OUTPUT.PUT_LINE(v_VDATE) ;
 END; ' ;
BEGIN
 DBMS_DDL.CREATE_WRAPPED (c_CODE);
END;
/

```

**ORACLE**

Copyright © 2013, Oracle und/oder verbundene Unternehmen. All rights reserved. Alle Rechte vorbehalten.

Das Beispiel auf der Folie zeigt die Erstellung der dynamisch verschlüsselten Prozedur NEW\_PROC. Um sicherzustellen, dass der Code für NEW\_PROC verschlüsselt ist, können Sie die Dictionary Views DBA|ALL|USER\_SOURCE abfragen:

```

SELECT text FROM user_source
WHERE name = 'NEW_PROC';

```

|                                       |
|---------------------------------------|
| TEXT                                  |
| -----                                 |
| PROCEDURE new_proc wrapped<br>a000000 |

|                                                                                                                                                                        |
|------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| 7 71 9e hBWMpGeSsd58b4jCP3/0d04rof0wg5nnm7+fMr2ywFyFDGLQ1haXriu4dCuPCWnnx1J0Ulxp pvc8nsr7Seq/riQvHrsXAQovdoh0K6ZvM1Kbskr+KLK957KzHQYwLK4k6rJLC55EyJ7qJB/2 RDmm3j79Uw== |
|------------------------------------------------------------------------------------------------------------------------------------------------------------------------|

## PL/SQL Wrapper

- **PL/SQL Wrapper ist ein Standalone-Utility, das PL/SQL-Quellcode in portierbaren Objektcode konvertiert und dadurch die interne Funktionalität von Anwendungen verbirgt.**
- **Das Wrapping-Verfahren weist folgende Features auf:**
  - Plattformunabhängigkeit
  - Dynamisches Laden
  - Dynamisches Binding
  - Abhängigkeitsprüfung
  - Normaler Im- und Export beim Aufruf

ORACLE

Copyright © 2013, Oracle und/oder verbundene Unternehmen. All rights reserved. Alle Rechte vorbehalten.

Mit PL/SQL Wrapper können Sie PL/SQL-Anwendungen bereitstellen, ohne Ihren eigenen Quellcode aufzudecken, der möglicherweise proprietäre Algorithmen und Datenstrukturen enthält. Der Wrapper konvertiert den lesbaren Quellcode in unlesbaren Code. Er verbirgt die interne Funktionalität der Anwendung und verhindert so deren Missbrauch.

Mit PL/SQL Wrapper konvertierter Code (etwa gespeicherte PL/SQL-Programme) bietet mehrere Features:

- Er ist plattformunabhängig. Sie müssen daher nicht mehrere Versionen derselben Komplilierungseinheit bereitstellen.
- Er lässt dynamisches Laden zu. Die Benutzer müssen daher zum Hinzufügen neuer Features keinen Neustart vornehmen.
- Er lässt dynamisches Binding zu. Somit werden externe Referenzen beim Laden aufgelöst.
- Er bietet eine strenge Abhängigkeitsprüfung. Invalidierte Programmeinheiten werden beim Aufruf automatisch rekompiliert.
- Er unterstützt normalen Im- und Export. Dadurch kann das Import/Export-Utility konvertierte Dateien verarbeiten.

# PL/SQL Wrapper ausführen

```
WRAP INAME=input_file_name [ONAME=output_file_name]
```

- Keine Leerzeichen vor und nach Gleichheitszeichen
- Das Argument **INAME** ist obligatorisch.
- Die Standarderweiterung der Eingabedatei ist **.sql**, sofern keine andere Erweiterung mit dem Namen angegeben wird.
- Das Argument **ONAME** ist optional.
- Die Standarderweiterung der Ausgabedatei ist **.plb**, sofern keine andere Erweiterung mit dem Argument **ONAME** angegeben wird.

## Beispiele

```
WRAP INAME=demo_04_hello.sql
WRAP INAME=demo_04_hello
WRAP INAME=demo_04_hello.sql ONAME=demo_04_hello.plb
```

**ORACLE**

Copyright © 2013, Oracle und/oder verbundene Unternehmen. All rights reserved. Alle Rechte vorbehalten.

Das Utility PL/SQL Wrapper ist ein ausführbares Betriebssystemprogramm mit der Bezeichnung WRAP.

**Hinweis:** Der Code stammt vollständig aus der Version 11g.

Um den Wrapper auszuführen, geben Sie an einer Eingabeaufforderung Ihres Betriebssystems den folgenden Befehl ein:

```
WRAP INAME=input_file_name [ONAME=output_file_name]
```

Alle Beispiele auf der Folie verwenden die Datei `demo_04_hello.sql` als Eingabe und erstellen die Ausgabedatei `demo_04_hello.plb`.

Führen Sie nach der Erstellung der konvertierten Datei die PLB-Datei aus SQL\*Plus aus (analog zu SQL-Skriptdateien). Dadurch wird die konvertierte Version des Quellcodes kompiliert und gespeichert.

**Hinweis:**

- Nur das Argument **INAME** ist obligatorisch. Wird **ONAME** nicht angegeben, erhält die Ausgabedatei denselben Namen wie die Eingabedatei mit der Erweiterung **.plb**.
- Die Eingabedatei kann eine beliebige Erweiterung haben. Der Standardwert ist **.sql**.
- Die Beachtung der Groß-/Kleinschreibung bei den Werten für **INAME** und **ONAME** richtet sich nach dem Betriebssystem.
- Im Allgemeinen ist die Ausgabedatei erheblich größer als die Eingabedatei.
- Geben Sie bei den Argumenten und Werten für **INAME** und **ONAME** keine Leerzeichen vor oder hinter den Gleichheitszeichen ein.

## Wrapping-Ergebnisse

```
-- Original PL/SQL source code in input file:

CREATE PACKAGE banking IS
 min_bal := 100;
 no_funds EXCEPTION;
 ...
END banking;
/
```

```
-- Wrapped code in output file:

CREATE PACKAGE banking
 wrapped
012abc463e ...
/
```

ORACLE

Copyright © 2013, Oracle und/oder verbundene Unternehmen. All rights reserved. Alle Rechte vorbehalten.

Konvertierte Objekttypen, Packages oder Unterprogramme weisen das folgende Format auf:  
Header, gefolgt vom Wort `wrapped` und danach der verschlüsselte Body.

Die Eingabedatei kann eine beliebige Kombination von SQL-Anweisungen enthalten. PL/SQL  
Wrapper konvertiert jedoch nur folgende CREATE-Anweisungen:

- CREATE [OR REPLACE] TYPE
- CREATE [OR REPLACE] TYPE BODY
- CREATE [OR REPLACE] PACKAGE
- CREATE [OR REPLACE] PACKAGE BODY
- CREATE [OR REPLACE] FUNCTION
- CREATE [OR REPLACE] PROCEDURE

Alle anderen SQL-Anweisungen vom Typ CREATE werden unverändert in die Ausgabedatei  
übertragen.

## Wrapping-Richtlinien

- Sie müssen nur den Packagebody konvertieren, nicht die Packagespezifikation.
- Der Wrapper kann Syntaxfehler erkennen, jedoch keine semantischen Fehler.
- Die Ausgabedatei darf nicht bearbeitet werden. Behalten Sie den ursprünglichen Quellcode bei, und konvertieren Sie ihn erneut, falls nötig.
- Um sicherzustellen, dass alle wichtigen Bestandteile Ihres Quellcodes verschlüsselt sind, zeigen Sie die konvertierte Datei vor der Verteilung in einem Texteditor an.

**ORACLE**

Copyright © 2013, Oracle und/oder verbundene Unternehmen. All rights reserved. Alle Rechte vorbehalten.

Beachten Sie folgende Richtlinien:

- Konvertieren Sie bei Packages oder Objekttypen nur den Body, nicht die Spezifikation. Auf diese Weise stellen Sie anderen Entwicklern die benötigten Angaben zur Verwendung des Packages zur Verfügung, ohne dessen Implementierung offenzulegen.
- PL/SQL Wrapper erkennt gegebenenfalls vorhandene Syntaxfehler in Ihrer Eingabedatei und meldet sie. Der Wrapper kann allerdings keine semantischen Fehler erkennen, weil er keine externen Referenzen auflöst. Beispielsweise meldet der Wrapper keinen Fehler, wenn die Tabelle oder View `amp` nicht vorhanden ist:

```
CREATE PROCEDURE raise_salary (emp_id INTEGER, amount NUMBER) AS
BEGIN
 UPDATE amp -- should be emp
 SET sal = sal + amount WHERE empno = emp_id;
END;
```

Der PL/SQL-Compiler löst dagegen externe Referenzen auf. Daher werden semantische Fehler bei der Kompilierung der konvertierten Ausgabedatei (PLB-Datei) gemeldet.

- Die Ausgabedatei darf nicht bearbeitet werden, da ihr Inhalt nicht lesbar ist. Um konvertierte Objekte zu ändern, müssen Sie den ursprünglichen Quellcode bearbeiten und erneut konvertieren.

## Package DBMS\_DDL und Utility wrap – Vergleich

| Funktionalität                                   | Package DBMS_DDL | Utility wrap |
|--------------------------------------------------|------------------|--------------|
| Code-Obfuscation                                 | Ja               | Ja           |
| Dynamische Obfuscation                           | Ja               | Nein         |
| Gleichzeitige Verschlüsselung mehrerer Programme | Nein             | Ja           |

**ORACLE®**

Copyright © 2013, Oracle und/oder verbundene Unternehmen. All rights reserved. Alle Rechte vorbehalten.

Das Utility wrap und das Package DBMS\_DDL erfüllen jeweils eigene Verwendungszwecke:

### Utility wrap

Das Utility wrap ist nützlich, um mit einer Ausführung mehrere Programme zu verschlüsseln. Im Prinzip kann damit eine komplette Anwendung gewrappt werden. Das Utility wrap lässt sich jedoch nicht für die Verschlüsselung von dynamisch generiertem Code zur Laufzeit einsetzen. Das Utility wrap verarbeitet eine SQL-Eingabedatei und verschlüsselt nur die PL/SQL-Einheiten in der Datei. Beispiel:

- Packagespezifikation und Packagebody
- Funktion und Prozedur
- Typspezifikation und Typbody

Das Utility wrap verschlüsselt keinen PL/SQL-Inhalt in:

- anonymen Blöcken
- Triggern
- Nicht-PL/SQL-Code

### Package DBMS\_DDL

Das Package DBMS\_DDL dient zur Verschlüsselung einer dynamisch generierten Programmseinheit innerhalb einer anderen Programmeinheit. Mit den Methoden des Packages DBMS\_DDL ist es nicht möglich, mehrere Programmeinheiten in einer Ausführung zu verschlüsseln. Jede Ausführung dieser Methoden akzeptiert nur jeweils eine CREATE OR REPLACE-Anweisung als Argument.

## Zusammenfassung

**In diesem Anhang haben Sie Folgendes gelernt:**

- **Conditional Compilation beschreiben und verwenden**
- **PL/SQL-Quellcode mit dynamischer Obfuscation und dem Utility `wrap` verbergen**

**ORACLE**

Copyright © 2013, Oracle und/oder verbundene Unternehmen. All rights reserved. Alle Rechte vorbehalten.

# **PL/SQL – Wiederholung**

**ORACLE**

Copyright © 2013, Oracle und/oder verbundene Unternehmen. All rights reserved. Alle Rechte vorbehalten.

## Ziele

Nach Beendigung dieses Anhangs haben Sie folgende Ziele erreicht:

- Blockstruktur für anonyme PL/SQL-Blöcke prüfen
- PL/SQL-Variablen deklarieren
- PL/SQL-Records und -Tabellen erstellen
- Daten einfügen, aktualisieren und löschen
- IF-, THEN- und ELSIF-Anweisungen verwenden
- Basis-, FOR- und WHILE-Schleifen verwenden
- Explizite Cursor mit Parametern deklarieren und verwenden
- Cursor FOR-Schleifen sowie FOR UPDATE- und WHERE CURRENT OF-Klauseln verwenden
- Vordefinierte und benutzerdefinierte Exceptions abfangen



Copyright © 2013, Oracle und/oder verbundene Unternehmen. All rights reserved. Alle Rechte vorbehalten.

Sie haben implizite Cursor kennengelernt, die von PL/SQL automatisch erstellt werden, wenn Sie die SQL-Anweisung SELECT oder eine DML-Anweisung ausführen. In dieser Lektion werden explizite Cursor behandelt. Sie lernen den Unterschied zwischen impliziten und expliziten Cursorn kennen. Darüber hinaus wird erläutert, wie Sie einfache Cursor und Cursor mit Parametern deklarieren und kontrollieren.

# Blockstruktur für anonyme PL/SQL-Blöcke

- **DECLARE** (optional)
  - PL/SQL-Objekte für die Verwendung innerhalb des jeweiligen Blockes deklarieren
- **BEGIN** (erforderlich)
  - Ausführbare Anweisungen deklarieren
- **EXCEPTION** (optional)
  - Bei Fehlern oder Exceptions durchzuführende Aktionen definieren
- **END;** (erforderlich)

**ORACLE**

Copyright © 2013, Oracle und/oder verbundene Unternehmen. All rights reserved. Alle Rechte vorbehalten.

## Anonyme Blöcke

Anonyme Blöcke haben keine Namen. Sie werden an dem Punkt in einer Anwendung deklariert, an dem sie ausgeführt werden sollen, und für die Ausführung zur Laufzeit an die PL/SQL-Engine übergeben.

- Der Bereich zwischen den Schlüsselwörtern **DECLARE** und **BEGIN** wird als deklarativer Bereich bezeichnet. Im deklarativen Bereich werden PL/SQL-Objekte wie Variablen, Konstanten, Cursor und benutzerdefinierte Exceptions definiert, die innerhalb des Blockes referenziert werden sollen. Das Schlüsselwort **DECLARE** ist optional, wenn Sie keine PL/SQL-Objekte deklarieren.
- Die Schlüsselwörter **BEGIN** und **END** sind obligatorisch. Sie umgeben den Bodyteil mit den auszuführenden Aktionen. Dieser Bereich wird als ausführbarer Bereich des Blockes bezeichnet.
- Der Bereich zwischen **EXCEPTION** und **END** ist der Exception-Bereich. In diesem Bereich werden Fehlerbedingungen abgefangen. Sie definieren darin Aktionen, die durchgeführt werden sollen, wenn eine angegebene Bedingung eintritt. Der Exception-Bereich ist optional.

Die Schlüsselwörter **DECLARE**, **BEGIN** und **EXCEPTION** werden nicht durch Strichpunkt, sondern durch **END** abgeschlossen. Für alle anderen PL/SQL-Anweisungen sind jedoch Strichpunkte erforderlich.

## PL/SQL-Variablen deklarieren

- **Syntax:**

```
identifier [CONSTANT] datatype [NOT NULL]
[:= | DEFAULT expr];
```

- **Beispiele:**

```
Declare
 v_hiredate DATE;
 v_deptno NUMBER(2) NOT NULL := 10;
 v_location VARCHAR2(13) := 'Atlanta';
 c_comm CONSTANT NUMBER := 1400;
 v_count BINARY_INTEGER := 0;
 v_valid BOOLEAN NOT NULL := TRUE;
```

**ORACLE**

Copyright © 2013, Oracle und/oder verbundene Unternehmen. All rights reserved. Alle Rechte vorbehalten.

Alle PL/SQL-Bezeichner müssen im deklarativen Bereich deklariert sein, damit sie im PL/SQL-Block referenziert werden können. Sie haben die Möglichkeit, einer Variablen einen Ausgangswert zuzuweisen. Variablen können jedoch auch ohne zugewiesenen Wert deklariert werden. Wenn Sie andere Variablen in einer Deklaration referenzieren, stellen Sie sicher, dass diese in einer vorhergehenden Anweisung bereits separat deklariert wurden.

Für die Syntax gilt:

*Identifier* ist der Name der Variablen.

*CONSTANT* legt die Variable als Konstante fest, sodass ihr Wert nicht geändert werden kann. Konstanten müssen initialisiert werden.

*Datatype* ist ein Datentyp wie skalar, zusammengesetzt, Referenz oder LOB. (In diesem Kurs werden nur skalare und zusammengesetzte Datentypen behandelt.)

*NOT NULL* schränkt die Variable so ein, dass sie einen Wert enthalten muss. *NOT NULL*-Variablen müssen initialisiert werden.

*expr* steht für einen PL/SQL-Ausdruck und kann ein Literal, eine andere Variable oder ein Ausdruck mit Operatoren und Funktionen sein.

## Variablen mit dem Attribut %TYPE deklarieren – Beispiele

```
...
 v_ename employees.last_name%TYPE;
 v_balance NUMBER(7,2);
 v_min_balance v_balance%TYPE := 10;
...
```

ORACLE

Copyright © 2013, Oracle und/oder verbundene Unternehmen. All rights reserved. Alle Rechte vorbehalten.

### Variablen mit dem Attribut %TYPE deklarieren

Deklarieren Sie Variablen, die den Namen eines Mitarbeiters aufnehmen.

```
...
 v_ename employees.last_name%TYPE;
...
```

Deklarieren Sie Variablen, um für ein Konto den Kontostand sowie ein Mindestguthaben von 10 als Ausgangswert zu speichern.

```
...
 v_balance NUMBER(7,2);
 v_min_balance v_balance%TYPE := 10;
...
```

Das Constraint NOT NULL für Spalten gilt nicht für Variablen, die mit %TYPE deklariert wurden. Daher können Sie einer Variablen, die Sie mit dem Attribut %TYPE deklariert haben und die eine als NOT NULL definierten Datenbankspalte verwendet, den Wert NULL zuordnen.

## PL/SQL-Records erstellen

**Variablen deklarieren, um Namen, Tätigkeit und Gehalt eines neuen Mitarbeiters zu speichern**

```
...
TYPE emp_record_type IS RECORD
 (ename VARCHAR2(25),
 job VARCHAR2(10),
 sal NUMBER(8,2));
emp_record emp_record_type;
...
```

ORACLE

Copyright © 2013, Oracle und/oder verbundene Unternehmen. All rights reserved. Alle Rechte vorbehalten.

Felddeklarationen sind mit Variablen Deklarationen vergleichbar. Jedes Feld besitzt einen eindeutigen Namen und einen spezifischen Datentyp. Im Gegensatz zu skalaren Variablen gibt es für PL/SQL-Records keine vordefinierten Datentypen. Daher müssen Sie zuerst den Datentyp erstellen und danach einen Bezeichner mit diesem Datentyp deklarieren.

Das folgende Beispiel zeigt, wie Sie mit dem Attribut %TYPE einen Felddatentyp angeben:

```
DECLARE
 TYPE emp_record_type IS RECORD
 (empid NUMBER(6) NOT NULL := 100,
 ename employees.last_name%TYPE,
 job employees.job_id%TYPE);
 emp_record emp_record_type;
...
```

**Hinweis:** Um zu verhindern, dass diesem Feld Nullwerte zugewiesen werden, können Sie Felddeklarationen das Constraint NOT NULL hinzufügen. Denken Sie daran, dass die mit NOT NULL deklarierten Felder initialisiert werden müssen.

## Attribut %ROWTYPE – Beispiele

- Variable deklarieren, die die Informationen zur Abteilung aus der Tabelle DEPARTMENTS übernimmt

```
dept_record departments%ROWTYPE;
```

- Variable deklarieren, die die Informationen zum Mitarbeiter aus der Tabelle EMPLOYEES übernimmt

```
emp_record employees%ROWTYPE;
```

**ORACLE**

Copyright © 2013, Oracle und/oder verbundene Unternehmen. All rights reserved. Alle Rechte vorbehalten.

### Beispiele

In der ersten Deklaration auf der Folie wird ein Record erstellt, der Feldnamen und Felddatentypen aus einer Zeile der Tabelle DEPARTMENTS übernimmt. Die Felder sind DEPARTMENT\_ID, DEPARTMENT\_NAME, MANAGER\_ID und LOCATION\_ID.

In der zweiten Deklaration auf der Folie wird ein Record erstellt, der die Feldnamen und Felddatentypen aus einer Zeile der Tabelle EMPLOYEES übernimmt. Die Felder sind EMPLOYEE\_ID, FIRST\_NAME, LAST\_NAME, EMAIL, PHONE\_NUMBER, HIRE\_DATE, JOB\_ID, SALARY, COMMISSION\_PCT, MANAGER\_ID und DEPARTMENT\_ID.

Im folgenden Beispiel wählen Sie Spaltenwerte für den Record job\_record.

```
· DECLARE
 job_record jobs%ROWTYPE;
 ...
BEGIN
 SELECT * INTO job_record
 FROM jobs
 WHERE ...
```

# PL/SQL-Tabellen erstellen

```

DECLARE
 TYPE ename_table_type IS TABLE OF
 employees.last_name%TYPE
 INDEX BY BINARY_INTEGER;
 TYPE hiredate_table_type IS TABLE OF DATE
 INDEX BY BINARY_INTEGER;
 ename_table ename_table_type;
 hiredate_table hiredate_table_type;
BEGIN
 ename_table(1) := 'CAMERON';
 hiredate_table(8) := SYSDATE + 7;
 IF ename_table.EXISTS(1) THEN
 INSERT INTO ...
 ...
END;

```

**ORACLE**

Copyright © 2013, Oracle und/oder verbundene Unternehmen. All rights reserved. Alle Rechte vorbehalten.

Für PL/SQL-Tabellen gibt es keine vordefinierten Datentypen wie für skalare Variablen. Daher müssen Sie zuerst den Datentyp erstellen und danach einen Bezeichner mit diesem Datentyp deklarieren.

## PL/SQL-Tabellen referenzieren

### Syntax

pl/sql\_table\_name(primary\_key\_value)

In dieser Syntax hat primary\_key\_value den Datentyp BINARY\_INTEGER.

Referenzieren Sie die dritte Zeile in der PL/SQL-Tabelle ENAME\_TABLE.

.ename\_table(3) ...

Der Wertebereich von BINARY\_INTEGER liegt zwischen -2.147.483.647 und 2.147.483.647. Der Primärschlüsselwert kann daher negativ sein. Die Indizierung muss nicht mit 1 beginnen.

**Hinweis:** Die Anweisung *table*.EXISTS(*i*) gibt TRUE zurück, wenn mindestens eine Zeile mit dem Index *i* zurückgegeben wird. Mit der Anweisung EXISTS verhindern Sie, dass durch die Referenzierung eines nicht vorhandenen Tabellenelements eine Fehlermeldung ausgelöst wird.

## SELECT-Anweisungen in PL/SQL – Beispiel

Die Klausel `INTO` ist obligatorisch.

```
DECLARE
 v_deptid NUMBER(4);
 v_loc NUMBER(4);
BEGIN
 SELECT department_id, location_id
 INTO v_deptid, v_loc
 FROM departments
 WHERE department_name = 'Sales';
 ...
END;
```

ORACLE

Copyright © 2013, Oracle und/oder verbundene Unternehmen. All rights reserved. Alle Rechte vorbehalten.

### Klausel `INTO`

Die Klausel `INTO` ist obligatorisch und steht zwischen den Klauseln `SELECT` und `FROM`. Sie gibt die Namen der Variablen an, die die von SQL aus der Klausel `SELECT` zurückgegebenen Werte aufnehmen. Zu jedem gewählten Element muss eine Variable angegeben werden, wobei die Reihenfolge der Variablen den gewählten Elementen entsprechen muss.

Mit der Klausel `INTO` füllen Sie entweder PL/SQL-Variablen oder Host-Variablen.

### Abfragen müssen genau eine Zeile zurückgeben.

Die `SELECT`-Anweisungen in einem PL/SQL-Block fallen in die ANSI-Klassifikation für eingebettetes SQL, für die folgende Regel gilt:

Abfragen müssen genau eine Zeile zurückgeben. Bei Rückgabe von mehreren oder null Zeilen wird ein Fehler generiert.

PL/SQL behandelt diese Fehler durch die Auslösung von Standard-Exceptions, die Sie im Exception-Bereich des Blockes mit den Exceptions `NO_DATA_FOUND` und `TOO_MANY_ROWS` abfangen können. Codieren Sie die `SELECT`-Anweisungen so, dass eine einzelne Zeile zurückgegeben wird.

## Daten einfügen – Beispiel

Der Tabelle **EMPLOYEES** neue Mitarbeiterinformationen hinzufügen

```
DECLARE
 v.empid employees.employee_id%TYPE;
BEGIN
 SELECT employees_seq.NEXTVAL
 INTO v.empno
 FROM dual;
 INSERT INTO employees(employee_id, last_name,
 job_id, department_id)
 VALUES(v.empid, 'HARDING', 'PU_CLERK', 30);
END;
```

ORACLE

Copyright © 2013, Oracle und/oder verbundene Unternehmen. All rights reserved. Alle Rechte vorbehalten.

### Daten einfügen

- SQL-Funktionen wie `USER` und `SYSDATE` verwenden
- Mit Datenbank-Sequences Primärschlüsselwerte generieren
- Werte im PL/SQL-Block abrufen
- Standardwerte für Spalten hinzufügen

**Hinweis:** In der Anweisung `INSERT` kann es keine Eindeutigkeitsprobleme zwischen Bezeichnern und Spaltennamen geben. Jeder Bezeichner in der Klausel `INSERT` muss ein Name einer Datenbankspalte sein.

## Daten aktualisieren – Beispiel

**Die Gehälter aller Mitarbeiter aus der Einkaufsabteilung in der Tabelle EMPLOYEES erhöhen**

```
DECLARE
 v_sal_increase employees.salary%TYPE := 2000;
BEGIN
 UPDATE employees
 SET salary = salary + v_sal_increas
 WHERE job_id = 'PU_CLERK';
END;
```

ORACLE

Copyright © 2013, Oracle und/oder verbundene Unternehmen. All rights reserved. Alle Rechte vorbehalten.

### Daten aktualisieren

In der Klausel SET der Anweisung UPDATE kann es zu Mehrdeutigkeit kommen, da der Bezeichner auf der linken Seite des Zuweisungsoperators zwar immer eine Datenbankspalte ist, der Bezeichner auf der rechten Seite jedoch eine Datenbankspalte oder PL/SQL-Variable sein kann.

Denken Sie daran, dass in der Klausel WHERE festgelegt wird, welche Zeilen bearbeitet werden sollen. Wenn keine Zeilen geändert werden, tritt (anders als bei der Anweisung SELECT in PL/SQL) kein Fehler auf.

**Hinweis:** Für Zuweisungen von PL/SQL-Variablen wird immer der Operator :=, für Zuweisungen von SQL-Spalten der Operator = . verwendet. Wenn Spalten- und Bezeichnernamen in der Klausel WHERE identisch sind, sucht der Oracle-Server zunächst in der Datenbank nach dem Namen.

## Daten löschen – Beispiel

**Zeilen, die zur Abteilung 190 gehören, aus der Tabelle EMPLOYEES löschen**

```
DECLARE
 v_deptid employees.department_id%TYPE := 190;
BEGIN
 DELETE FROM employees
 WHERE department_id = v_deptid;
END;
```

ORACLE

Copyright © 2013, Oracle und/oder verbundene Unternehmen. All rights reserved. Alle Rechte vorbehalten.

Bestimmte Tätigkeit löschen:

```
DECLARE
 v_jobid jobs.job_id%TYPE := 'PR_REP';
BEGIN
 DELETE FROM jobs
 WHERE job_id = v_jobid;
END;
```

## Transaktionen mit den Anweisungen COMMIT und ROLLBACK steuern

- **Transaktion mit dem ersten DML-Befehl nach einer COMMIT oder ROLLBACK-Anweisung initiieren**
- **Mit den SQL-Anweisungen COMMIT und ROLLBACK Transaktionen explizit beenden**

ORACLE

Copyright © 2013, Oracle und/oder verbundene Unternehmen. All rights reserved. Alle Rechte vorbehalten.

Sie kontrollieren die Logik von Transaktionen mit den SQL-Anweisungen COMMIT und ROLLBACK, indem Sie bestimmte Datenbankänderungen festschreiben und andere verwerfen. Wie beim Oracle-Server beginnen die DML-(Data Manipulation Language-)Transaktionen beim ersten Befehl nach einem COMMIT oder ROLLBACK und enden beim nächsten erfolgreichen COMMIT oder ROLLBACK. Die Aktionen können in einem PL/SQL-Block oder als Ergebnis von Ereignissen in der Hostumgebung auftreten. Eine COMMIT-Anweisung beendet die aktuelle Transaktion, indem sie alle ausstehenden Datenbankänderungen festschreibt.

### Syntax

```
COMMIT [WORK];
ROLLBACK [WORK];
```

In dieser Syntax gibt WORK die Einhaltung des ANSI-Standards an.

**Hinweis:** Alle Befehle für die Transaktionskontrolle sind in PL/SQL gültig. Ihre Verwendung kann jedoch durch die Hostumgebung eingeschränkt sein.

Sie können auch explizite Sperrbefehle (beispielsweise LOCK TABLE und SELECT ... FOR UPDATE) in einen Block aufnehmen. Sie bleiben bis zum Ende der Transaktion gültig. Außerdem gilt, dass ein PL/SQL-Block nicht notwendigerweise eine Transaktion bedeutet.

## IF-, THEN- und ELSIF-Anweisungen – Beispiel

Für einen eingegebenen Wert einen berechneten Wert zurückgegeben

```
 . . .
IF v_start > 100 THEN
 v_start := 2 * v_start;
ELSIF v_start >= 50 THEN
 v_start := 0.5 * v_start;
ELSE
 v_start := 0.1 * v_start;
END IF;
. . .
```

ORACLE

Copyright © 2013, Oracle und/oder verbundene Unternehmen. All rights reserved. Alle Rechte vorbehalten.

### IF-, THEN- und ELSIF-Anweisungen

Verwenden Sie nach Möglichkeit die Klausel ELSIF anstelle verschachtelter IF-Anweisungen. Der Code ist einfacher zu lesen und leichter verständlich. Die Logik ist klarer definiert. Wenn die Aktion in der Klausel ELSE nur aus einer weiteren IF-Anweisung besteht, ist die Klausel ELSIF zweckmäßiger. Der Code wird übersichtlicher, da an das Ende von weiteren Bedingungs- und Aktionengruppen keine verschachtelten END IF-Anweisungen angefügt werden müssen.

#### Beispiel

```
IF condition1 THEN
 statement1;
ELSIF condition2 THEN
 statement2;
ELSIF condition3 THEN
 statement3;
END IF;
```

Die Anweisung auf der Folie ist außerdem wie folgt definiert:

Für einen eingegebenen Wert wird ein berechneter Wert zurückgegeben. Wenn der eingegebene Wert über 100 liegt, ist der berechnete Wert doppelt so hoch wie der eingegebene Wert. Liegt der eingegebene Wert zwischen 50 und 100, beträgt der berechnete Wert 50 % des Ausgangswertes. Liegt der eingegebene Wert unter 50, beträgt der berechnete Wert 10 % des Ausgangswertes.

**Hinweis:** Die Auswertung arithmetischer Ausdrücke, die Nullwerte enthalten, ergibt null.

## Basisschleifen – Beispiel

```
DECLARE
 v_ordid order_items.order_id%TYPE := 101;
 v_counter NUMBER(2) := 1;
BEGIN
 LOOP
 INSERT INTO order_items(order_id,line_item_id)
 VALUES(v_ordid, v_counter);
 v_counter := v_counter + 1;
 EXIT WHEN v_counter > 10;
 END LOOP;
END;
```

ORACLE

Copyright © 2013, Oracle und/oder verbundene Unternehmen. All rights reserved. Alle Rechte vorbehalten.

Das auf der Folie gezeigte Beispiel einer Basisschleife ist wie folgt definiert:

Für Bestellnummer 101 werden die ersten 10 neuen Positionen eingefügt.

**Hinweis:** Mit einer Basisschleife können Sie die zugehörigen Anweisungen selbst dann mindestens ein Mal ausführen, wenn die Bedingung beim Aufruf der Schleife bereits erfüllt ist.

## FOR-Schleifen – Beispiel

Die ersten 10 neuen Positionen für Bestellnummer 101 einfügen

```
DECLARE
 v_ordid order_items.order_id%TYPE := 101;
BEGIN
 FOR i IN 1..10 LOOP
 INSERT INTO order_items(order_id,line_item_id)
 VALUES(v_ordid, i);
 END LOOP;
END;
```

ORACLE

Copyright © 2013, Oracle und/oder verbundene Unternehmen. All rights reserved. Alle Rechte vorbehalten.

Das Beispiel zeigt eine FOR-Schleife, die 10 Zeilen in die Tabelle order\_items einfügt.

## WHILE-Schleifen – Beispiel

```
ACCEPT p_price PROMPT 'Enter the price of the item: '
ACCEPT p_itemtot -
 PROMPT 'Enter the maximum total for purchase of item: '
DECLARE
 ...
 v_qty NUMBER(8) := 1;
 v_running_total NUMBER(7,2) := 0;

BEGIN
 ...
 WHILE v_running_total < &p_itemtot LOOP
 ...
 v_qty := v_qty + 1;
 v_running_total := v_qty * &p_price;
 END LOOP;
 ...

```

ORACLE

Copyright © 2013, Oracle und/oder verbundene Unternehmen. All rights reserved. Alle Rechte vorbehalten.

Im Beispiel auf der Folie nimmt die Menge mit jeder Wiederholung der Schleife zu, bis der für diesen Artikel zulässige Maximalpreis erreicht ist.

## Implizite SQL-Cursorattribute

### Mit SQL-Cursorattributen das Ergebnis der SQL-Anweisungen testen

| SQL-Cursorattribute       | Beschreibung                                                                                                                 |
|---------------------------|------------------------------------------------------------------------------------------------------------------------------|
| <code>SQL%ROWCOUNT</code> | Anzahl der Zeilen, die von der letzten SQL-Anweisung bearbeitet wurden (Ganzahlwert)                                         |
| <code>SQL%FOUND</code>    | Boolesches Attribut, das <code>TRUE</code> ist, wenn die letzte SQL-Anweisung mindestens eine Zeile betrifft                 |
| <code>SQL%NOTFOUND</code> | Boolesches Attribut, dessen Auswertung <code>TRUE</code> ergibt, wenn die letzte SQL-Anweisung keine Zeilen betrifft         |
| <code>SQL%ISOPEN</code>   | Boolesches Attribut, das immer <code>FALSE</code> ist, da PL/SQL implizite Cursor unmittelbar nach ihrer Ausführung schließt |

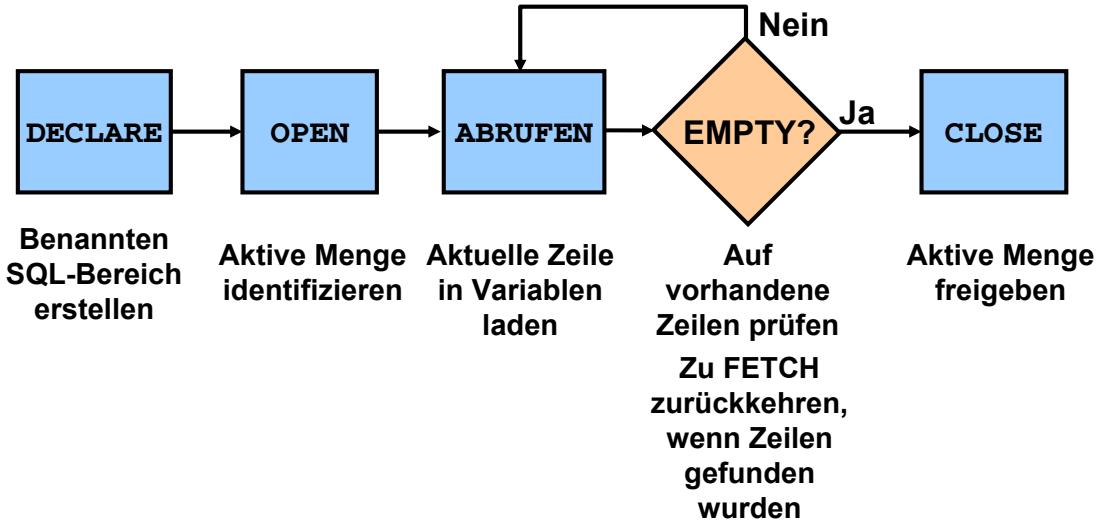
**ORACLE**

Copyright © 2013, Oracle und/oder verbundene Unternehmen. All rights reserved. Alle Rechte vorbehalten.

Mit den SQL-Cursorattributen können Sie feststellen, was bei der letzten Verwendung eines impliziten Cursors passiert ist. Sie verwenden diese Attribute in PL/SQL-Anweisungen wie Funktionen. In SQL-Anweisungen können die Attribute nicht verwendet werden.

Mit den Attributen `SQL%ROWCOUNT`, `SQL%FOUND`, `SQL%NOTFOUND` und `SQL%ISOPEN` können Sie im Exception-Bereich eines Blockes Informationen zur Ausführung einer DML-Anweisung sammeln. In PL/SQL werden DML-Anweisungen, die keine Zeilen ändern, nicht als Fehlerbedingung betrachtet. Jedoch gibt die Anweisung `SELECT` eine Exception zurück, wenn sie keine Zeilen findet.

# Explizite Cursor steuern



ORACLE

Copyright © 2013, Oracle und/oder verbundene Unternehmen. All rights reserved. Alle Rechte vorbehalten.

## Explizite Cursor

### Explizite Cursor mit vier Befehlen steuern

1. Deklarieren Sie den Cursor, indem Sie ihn benennen und die Struktur der auszuführenden Abfrage definieren.
2. Öffnen Sie den Cursor. Die Anweisung `OPEN` führt die Abfrage aus und bindet alle referenzierten Variablen. Die von der Abfrage identifizierten Zeilen werden als *aktive Menge* bezeichnet. Sie stehen jetzt für den Zeilenabruf (`Fetch`) zur Verfügung.
3. Lesen Sie Daten aus dem Cursor. Die Anweisung `FETCH` lädt die aktuelle Zeile aus dem Cursor in Variablen. Bei jedem Zeilenabruf wird der Cursorzeiger auf die jeweils nächste Zeile in der aktiven Menge bewegt, sodass jeder Zeilenabruf auf eine andere von der Abfrage zurückgegebene Zeile zugreift. Im Ablaufdiagramm auf der Folie wird bei jedem Zeilenabruf geprüft, ob im Cursor noch weitere Zeilen vorhanden sind. Falls ja, wird die aktuelle Zeile in Variablen geladen. Falls nein, wird der Cursor geschlossen.
4. Schließen Sie den Cursor. Die Anweisung `CLOSE` gibt die aktive Zeilenmenge frei. Sie können den Cursor jetzt erneut öffnen, um eine neue aktive Menge zu erstellen.

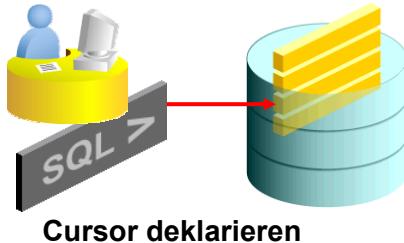
## Explizite Cursor steuern – Cursor deklarieren

```

DECLARE
 CURSOR c1 IS
 SELECT employee_id, last_name
 FROM employees;

 CURSOR c2 IS
 SELECT *
 FROM departments
 WHERE department_id = 10;
BEGIN
 ...

```



ORACLE

Copyright © 2013, Oracle und/oder verbundene Unternehmen. All rights reserved. Alle Rechte vorbehalten.

### Explizite Cursor deklarieren

Rufen Sie die Mitarbeiter nacheinander ab.

```

DECLARE
 v.empid employees.employee_id%TYPE;
 v.ename employees.last_name%TYPE;
 CURSOR c1 IS
 SELECT employee_id, last_name
 FROM employees;
BEGIN
 ...

```

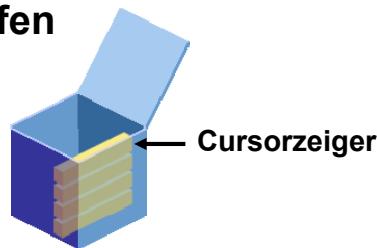
**Hinweis:** In der Abfrage können Variablen referenziert werden. Diese Variablen müssen jedoch vor der Ausführung der CURSOR-Anweisung deklariert werden.

## Explizite Cursor steuern – Cursor öffnen

```
OPEN cursor_name;
```

- **Cursor öffnen, um die Abfrage auszuführen und die aktive Menge zu identifizieren**
- **Gibt die Abfrage keine Zeilen zurück, wird keine Exception ausgelöst.**
- **Das Ergebnis eines Zeilenabrufs mithilfe von Cursorattributen prüfen**

Cursor öffnen



**ORACLE**

Copyright © 2013, Oracle und/oder verbundene Unternehmen. All rights reserved. Alle Rechte vorbehalten.

### Anweisung OPEN

Öffnen Sie den Cursor, um die Abfrage auszuführen und die Ergebnismenge zu ermitteln. Die Ergebnismenge enthält alle Zeilen, die die Suchkriterien der Abfrage erfüllen. Der Cursor zeigt jetzt auf die erste Zeile in der Ergebnismenge.

In der Syntax ist `cursor_name` der Name des zuvor deklarierten Cursors.

`OPEN` ist eine ausführbare Anweisung, die die folgenden Vorgänge ausführt:

1. Sie weist einem Kontextbereich, der nach und nach wichtige Verarbeitungsdaten speichert, dynamisch Speicher zu.
2. Sie parst die Anweisung `SELECT`.
3. Sie bindet die Eingabevervariablen, d. h. legt ihren Wert fest, indem sie ihre Speicheradressen abruft.
4. Sie ermittelt die Ergebnismenge, d. h. die Zeilen, die mit den Suchkriterien übereinstimmen. Die Zeilen in der Ergebnismenge werden nicht mit der Anweisung `OPEN`, sondern mit der Anweisung `FETCH` in Variablen abgerufen.
5. Sie positioniert den Zeiger direkt vor die erste Zeile der aktiven Menge.

**Hinweis:** Wenn die Abfrage bei geöffnetem Cursor keine Zeilen zurückgibt, löst PL/SQL keine Exception aus. Sie können den Cursorstatus jedoch nach einem Zeilenabruf prüfen.

Bei Cursorn, die mit der Klausel `FOR UPDATE` deklariert wurden, sperrt die Anweisung `OPEN` auch die entsprechenden Zeilen.

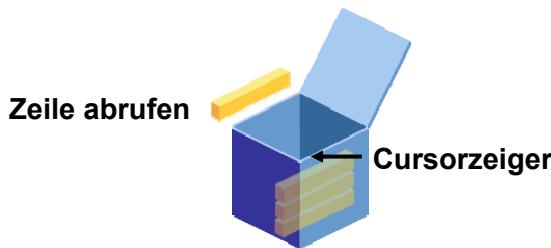
## Explizite Cursor steuern – Daten aus dem Cursor abrufen

```

FETCH c1 INTO v_empid, v_ename;

...
OPEN defined_cursor;
LOOP
 FETCH defined_cursor INTO defined_variables
 EXIT WHEN ...;
 ...
 -- Process the retrieved data
 ...
END;

```



ORACLE

Copyright © 2013, Oracle und/oder verbundene Unternehmen. All rights reserved. Alle Rechte vorbehalten.

### Anweisung **FETCH**

Mit der Anweisung **FETCH** werden die aktuellen Zeilenwerte in AusgabevARIABLEn abgerufen. Nach dem Zeilenabruf können Sie die Variablen mit weiteren Anweisungen bearbeiten. Zu jedem Spaltenwert, der von der Abfrage im Zusammenhang mit dem zugeordneten Cursor zurückgegeben wird, muss eine entsprechende Variable in der Liste **INTO** vorhanden sein. Außerdem müssen auch die entsprechenden Datentypen kompatibel sein. Rufen Sie die ersten 10 Mitarbeiter nacheinander ab:

```

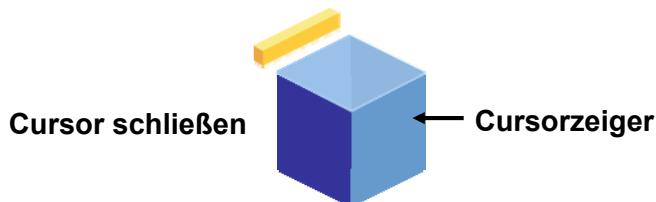
DECLARE
 v_empid employees.employee_id%TYPE;
 v_ename employees.last_name%TYPE;
 i NUMBER := 1;
 CURSOR c1 IS
 SELECT employee_id, last_name
 FROM employees;
BEGIN
 OPEN c1;
 FOR i IN 1..10 LOOP
 FETCH c1 INTO v_empid, v_ename;
 ...
 END LOOP;
END;

```

## Explizite Cursor steuern – Cursor schließen

```
CLOSE cursor_name;
```

- Nach Verarbeitung der Zeilen den Cursor schließen
- Cursor gegebenenfalls wieder öffnen
- Keine Daten aus einem Cursor abrufen, nachdem er geschlossen wurde



ORACLE®

Copyright © 2013, Oracle und/oder verbundene Unternehmen. All rights reserved. Alle Rechte vorbehalten.

### Anweisung CLOSE

Die Anweisung `CLOSE` deaktiviert den Cursor. Die Ergebnismenge ist damit nicht mehr definiert. Schließen Sie den Cursor, wenn die Verarbeitung der Anweisung `SELECT` abgeschlossen ist. Auf diese Weise kann der Cursor gegebenenfalls erneut geöffnet werden. Sie haben also die Möglichkeit, eine aktive Menge mehrmals zu erstellen.

In der Syntax ist `cursor_name` der Name des zuvor deklarierten Cursors.

Rufen Sie keine Daten aus einem Cursor ab, nachdem er geschlossen wurde, da sonst die Exception `INVALID_CURSOR` ausgelöst wird.

**Hinweis:** Die Anweisung `CLOSE` gibt den Kontextbereich frei. Der PL/SQL-Block kann zwar beendet werden, ohne den Cursor zu schließen, allerdings sollten Sie deklarierte Cursor stets explizit schließen, um Ressourcen freizugeben. Die maximale Anzahl der pro Benutzer geöffneten Cursor ist begrenzt. Sie wird im Feld der Datenbankparameter über den Parameter `OPEN_CURSORS` festgelegt. Der Standardwert für `OPEN_CURSORS` ist 50.

```
...
FOR i IN 1..10 LOOP
 FETCH c1 INTO v.empid, v.ename; ...
END LOOP;
CLOSE c1;
END;
```

## Attribute von expliziten Cursorn

### Statusinformationen zu einem Cursor abrufen

| Attribut  | Typ     | Beschreibung                                                                         |
|-----------|---------|--------------------------------------------------------------------------------------|
| ISOPEN    | BOOLEAN | Ist TRUE, wenn der Cursor geöffnet ist                                               |
| %NOTFOUND | BOOLEAN | Ist TRUE, wenn der letzte Zeilenabruf keine Zeile zurückgibt                         |
| %FOUND    | BOOLEAN | Ist TRUE, wenn der letzte Zeilenabruf eine Zeile zurückgibt. Gegenstück zu %NOTFOUND |
| %ROWCOUNT | NUMBER  | Ist die Gesamtanzahl der bis zu diesem Zeitpunkt zurückgegebenen Zeilen              |

ORACLE

Copyright © 2013, Oracle und/oder verbundene Unternehmen. All rights reserved. Alle Rechte vorbehalten.

Wie implizite Cursor haben auch explizite Cursor vier Attribute, mit denen Sie Statusinformationen abrufen können. Die an den Cursor oder die Cursorvariable angehängten Attribute geben nützliche Informationen zur Ausführung einer DML-Anweisung zurück.

**Hinweis:** Referenzieren Sie Cursorattribute nicht direkt in einer SQL-Anweisung.

## Cursor FOR-Schleifen – Beispiel

Alle vorhandenen Mitarbeiter nacheinander abrufen

```
DECLARE
 CURSOR c1 IS
 SELECT employee_id, last_name
 FROM employees;
BEGIN
 FOR emp_record IN c1 LOOP
 -- implicit open and implicit fetch occur
 IF emp_record.employee_id = 134 THEN
 ...
 END LOOP; -- implicit close occurs
END;
```

ORACLE

Copyright © 2013, Oracle und/oder verbundene Unternehmen. All rights reserved. Alle Rechte vorbehalten.

### Cursor FOR-Schleifen

Eine Cursor FOR-Schleife verarbeitet Zeilen in einem expliziten Cursor. Nach dem Öffnen des Cursors erfolgt ein einmaliger Zeilenabruf pro Schleifeniteration. Nachdem alle Zeilen verarbeitet wurden, wird der Cursor automatisch geschlossen. Die Schleife selbst wird am Ende der Iteration mit dem Abruf der letzten Zeile automatisch beendet. Im Beispiel auf der Folie ist `emp_record` in der Cursor FOR-Schleife ein implizit deklarierter Record, der im Konstrukt `FOR LOOP` verwendet wird.

## Klausel FOR UPDATE – Beispiel

**Am heutigen Tag verarbeitete Bestellungen mit Beträgen über \$ 1.000 abrufen**

```
DECLARE
 CURSOR c1 IS
 SELECT customer_id, order_id
 FROM orders
 WHERE order_date = SYSDATE
 AND order_total > 1000.00
 ORDER BY customer_id
 FOR UPDATE NOWAIT;
```

ORACLE

Copyright © 2013, Oracle und/oder verbundene Unternehmen. All rights reserved. Alle Rechte vorbehalten.

### Klausel FOR UPDATE

Wenn der Datenbankserver die für die Anweisung `SELECT FOR UPDATE` erforderlichen Zeilensperren nicht zuteilen kann, wartet er auf unbestimmte Zeit. Mit der Klausel `NOWAIT` in der Anweisung `SELECT FOR UPDATE` können Sie den Fehlercode prüfen, der bei einer nicht erfolgreichen Sperrenzuteilung in einer Schleife zurückgegeben wurde. Aus diesem Grund können Sie vor Beendigung des PL/SQL-Blockes beliebig oft versuchen, den Cursor zu öffnen.

Wenn Sie Zeilen mit der Klausel `WHERE CURRENT OF` aktualisieren oder löschen möchten, müssen Sie in der Klausel `FOR UPDATE OF` einen Spaltennamen angeben.

Bei einer sehr umfangreichen Tabelle erzielen Sie eine bessere Performance, wenn Sie mit der Anweisung `LOCK TABLE` alle Zeilen in der Tabelle sperren. Allerdings können Sie bei Verwendung von `LOCK TABLE` nicht die Klausel `WHERE CURRENT OF` einsetzen und müssen die Notation `WHERE column = identifier` verwenden.

## Klausel WHERE CURRENT OF – Beispiel

```
DECLARE
 CURSOR c1 IS
 SELECT salary FROM employees
 FOR UPDATE OF salary NOWAIT;
BEGIN
 ...
 FOR emp_record IN c1 LOOP
 UPDATE ...
 WHERE CURRENT OF c1;
 ...
 END LOOP;
 COMMIT;
END;
```



Copyright © 2013, Oracle und/oder verbundene Unternehmen. All rights reserved. Alle Rechte vorbehalten.

### Klausel WHERE CURRENT OF

Zeilen lassen sich auf Basis der Kriterien aus einem Cursor aktualisieren.

Darüber hinaus können Sie eine DELETE- oder UPDATE-Anweisung mit der Klausel WHERE CURRENT OF cursor\_name schreiben, die die letzte von der Anweisung FETCH verarbeitete Zeile referenziert. Wenn Sie diese Klausel verwenden, muss der von Ihnen referenzierte Cursor vorhanden sein und die Klausel FOR UPDATE in der Cursorabfrage enthalten. Andernfalls erhalten Sie eine Fehlermeldung. Mit dieser Klausel können Sie Aktualisierungs- oder Löschvorgänge auf die aktuelle Zeile anwenden, ohne die Pseudospalte ROWID explizit referenzieren zu müssen.

## Vordefinierte Oracle-Serverfehler abfangen

- **Standardnamen in der Exception-Behandlungsroutine referenzieren**
- **Beispiele für vordefinierte Exceptions:**
  - **NO\_DATA\_FOUND**
  - **TOO\_MANY\_ROWS**
  - **INVALID\_CURSOR**
  - **ZERO\_DIVIDE**
  - **DUP\_VAL\_ON\_INDEX**



Copyright © 2013, Oracle und/oder verbundene Unternehmen. All rights reserved. Alle Rechte vorbehalten.

Sie fangen einen vordefinierten Oracle-Serverfehler ab, indem Sie seinen Standardnamen innerhalb der entsprechenden Exception-Behandlungsroutine referenzieren.

**Hinweis:** PL/SQL deklariert vordefinierte Exceptions im Package STANDARD.

Es empfiehlt sich, die gängigsten Exceptions NO\_DATA\_FOUND und TOO\_MANY\_ROWS generell zu berücksichtigen.

## Vordefinierte Oracle-Serverfehler abfangen – Beispiel

```
BEGIN SELECT ... COMMIT;
EXCEPTION
 WHEN NO_DATA_FOUND THEN
 statement1;
 statement2;
 WHEN TOO_MANY_ROWS THEN
 statement1;
 WHEN OTHERS THEN
 statement1;
 statement2;
 statement3;
END;
```

ORACLE

Copyright © 2013, Oracle und/oder verbundene Unternehmen. All rights reserved. Alle Rechte vorbehalten.

Im Beispiel auf der Folie wird bei jeder Exception eine Meldung an den Benutzer ausgegeben. Es wird immer nur eine Exception ausgelöst und behandelt.

## Nicht vordefinierte Fehler

### Oracle-Serverfehler -2292 (Verletzung eines Integritäts-Constraints) abfangen

```

DECLARE
 e_products_invalid EXCEPTION; 1
 PRAGMA EXCEPTION_INIT (
 e_products_invalid, -2292); 2
 v_message VARCHAR2(50);
BEGIN
 . . .
 EXCEPTION
 WHEN e_products_invalid THEN 3
 :g_message := 'Product ID
 specified is not valid.';
 . . .
END;

```

**ORACLE**

Copyright © 2013, Oracle und/oder verbundene Unternehmen. All rights reserved. Alle Rechte vorbehalten.

### Nicht vordefinierte Oracle-Server-Exceptions abfangen

1. Deklarieren Sie den Namen der Exception im deklarativen Bereich.

#### Syntax

```
exception EXCEPTION;
```

In dieser Syntax ist *exception* der Name der Exception.

2. Ordnen Sie die deklarierte Exception der Standardfehlernummer des Oracle-Servers zu. Verwenden Sie dazu die Anweisung `PRAGMA EXCEPTION_INIT`.

#### Syntax

```
PRAGMA EXCEPTION_INIT(exception, error_number);
```

In dieser Syntax gilt:

|                  |                                     |
|------------------|-------------------------------------|
| <i>exception</i> | Ist die zuvor deklarierte Exception |
|------------------|-------------------------------------|

|                     |                                                                |
|---------------------|----------------------------------------------------------------|
| <i>error_number</i> | ist die Fehlernummer eines Standardfehlers des Oracle-Servers. |
|---------------------|----------------------------------------------------------------|

3. Referenzieren Sie die deklarierte Exception innerhalb der entsprechenden Exception-Behandlungsroutine.

Im Beispiel auf der Folie gilt: Wenn ein Produkt vorrätig ist, die Verarbeitung unterbrechen und für den Benutzer eine Meldung ausgeben.

# Benutzerdefinierte Exceptions – Beispiel

```
[DECLARE]
 e_amount_remaining EXCEPTION; 1
 .
 .
 .
BEGIN
 .
 .
 .
 RAISE e_amount_remaining; 2
 .
 .
 .
EXCEPTION
 WHEN e_amount_remaining THEN
 :g_message := 'There is still an amount
 in stock.';
 .
 .
 .
END;
```

**ORACLE**

Copyright © 2013, Oracle und/oder verbundene Unternehmen. All rights reserved. Alle Rechte vorbehalten.

## Benutzerdefinierte Exceptions abfangen

Fangen Sie benutzerdefinierte Exceptions ab, indem Sie sie deklarieren und explizit auslösen.

1. Deklarieren Sie den Namen der benutzerdefinierten Exception im deklarativen Bereich.

**Syntax:** exception EXCEPTION;

**Dabei gilt:** exception Ist der Name der Exception

2. Lösen Sie die Exception mithilfe der Anweisung RAISE im ausführbaren Bereich explizit aus.

**Syntax:** RAISE exception;

**Dabei gilt:** exception Ist die zuvor deklarierte Exception

3. Referenzieren Sie die deklarierte Exception innerhalb der entsprechenden Exception- Behandlungsroutine.

Im Beispiel auf der Folie gilt: Die Geschäftsregel des Kunden gibt vor, dass Produkte, die noch vorrätig sind, nicht aus der Datenbank entfernt werden können. Da es keine Constraints zur Durchsetzung dieser Regel gibt, verarbeitet der Entwickler sie explizit in der Anwendung. Bevor ein DELETE-Vorgang für die Tabelle PRODUCT\_INFORMATION durchgeführt wird, stellt der Block durch Abfrage der Tabelle INVENTORIES fest, ob das Produkt noch auf Lager ist. Ist dies der Fall, wird eine Exception ausgelöst.

**Hinweis:** Um dieselbe Exception in der aufrufenden Umgebung auszulösen, verwenden Sie die Anweisung RAISE allein in einem Exception Handler.

## RAISE\_APPLICATION\_ERROR-Prozeduren

```
raise_application_error (error_number,
 message[, {TRUE | FALSE}]);
```

- **Benutzerdefinierte Fehlermeldungen aus Stored Subprograms ausgeben**
- **Aufruf nur über ein ausgeführtes Stored Subprogram möglich**

ORACLE

Copyright © 2013, Oracle und/oder verbundene Unternehmen. All rights reserved. Alle Rechte vorbehalten.

Mit der Prozedur RAISE\_APPLICATION\_ERROR können Sie eine vordefinierte Exception durch Rückgabe eines speziellen Fehlercodes und einer entsprechenden Fehlermeldung interaktiv übermitteln. Mit der Prozedur RAISE\_APPLICATION\_ERROR können Sie den Anwendungen Fehler melden und die Rückgabe von nicht behandelten Exceptions vermeiden.

In der Syntax ist *error\_number* eine benutzerdefinierte Exception-Nummer zwischen -20.000 and -20.999. *message* ist die benutzerdefinierte Exception-Meldung. Hierbei handelt es sich um eine Zeichenfolge mit einer maximalen Länge von 2.048 Byte.

TRUE | FALSE ist ein optionaler boolescher Parameter. Wenn dieser TRUE ist, wird der Fehler im Stack der vorherigen Fehler platziert. Ist er FALSE (Standardwert), ersetzt der Fehler alle vorherigen Fehler.

### Beispiel:

```
...
EXCEPTION
WHEN NO_DATA_FOUND THEN
 RAISE_APPLICATION_ERROR (-20201,
 'Manager is not a valid employee.');
```

END;

## **RAISE\_APPLICATION\_ERROR-Prozeduren**

- **Verwendung in zwei Bereichen:**
  - Bereich **Executable**
  - Bereich **Exception**
- **Rückgabe der Fehlerbedingungen erfolgt konsistent mit anderen Oracle-Serverfehlern**

**ORACLE**

Copyright © 2013, Oracle und/oder verbundene Unternehmen. All rights reserved. Alle Rechte vorbehalten.

### **RAISE\_APPLICATION\_ERROR-Prozeduren – Beispiel**

```
...
DELETE FROM employees
WHERE manager_id = v_mgr;
IF SQL%NOTFOUND THEN
 RAISE_APPLICATION_ERROR(-20202,
 'This is not a valid manager');
END IF;
...
```

## Zusammenfassung

### In diesem Anhang haben Sie Folgendes gelernt:

- **Blockstruktur für anonyme PL/SQL-Blöcke prüfen**
- **PL/SQL-Variablen deklarieren**
- **PL/SQL-Records und -Tabellen erstellen**
- **Daten einfügen, aktualisieren und löschen**
- **IF-, THEN- und ELSIF-Anweisungen verwenden**
- **Basis-, FOR- und WHILE-Schleifen verwenden**
- **Explizite Cursor mit Parametern deklarieren und verwenden**
- **Cursor FOR-Schleifen sowie FOR UPDATE- und WHERE CURRENT OF-Klauseln verwenden**
- **Vordefinierte und benutzerdefinierte Exceptions abfangen**

**ORACLE**

Copyright © 2013, Oracle und/oder verbundene Unternehmen. All rights reserved. Alle Rechte vorbehalten.

Um SQL-Anweisungen auszuführen und Verarbeitungsinformationen zu speichern, verwendet der Oracle-Server Arbeitsbereiche. Mit einem PL/SQL-Konstrukt, das als *cursor* bezeichnet wird, können Sie einen Arbeitsbereich benennen und auf die darin gespeicherten Informationen zugreifen. Es gibt zwei Typen von Cursorn: implizite und explizite Cursor. PL/SQL deklariert einen Cursor implizit für alle SQL-Anweisungen zur Datenmanipulation, einschließlich Abfragen, die nur eine Zeile zurückgeben. Für Abfragen, die mehrere Zeilen zurückgeben, müssen Sie einen Cursor explizit deklarieren, um die Zeilen einzeln zu verarbeiten.

Jeder explizite Cursor und jede Cursorvariable hat vier Attribute: %FOUND, %ISOPEN, %NOTFOUND und %ROWCOUNT. Wenn Sie die Attribute an den Namen der Cursorvariablen anhängen, geben sie nützliche Informationen über die Ausführung einer SQL-Anweisung zurück. Sie können Cursorattribute in prozeduralen Anweisungen, nicht jedoch in SQL-Anweisungen verwenden.

Bearbeiten Sie die vom Cursor gelesenen Zeilen mithilfe einfacher oder Cursor FOR-Schleifen. Wenn Sie einfache Schleifen verwenden, müssen Sie den Cursor öffnen, die Zeilen lesen und den Cursor schließen. Cursor FOR-Schleifen führen diese Vorgänge implizit durch. Sperren, die Sie aktualisieren oder löschen, sollten Sie mit der Klausel FOR UPDATE sperren. Sie stellen damit sicher, dass die von Ihnen verwendeten Daten nicht durch eine andere Session aktualisiert werden, nachdem Sie den Cursor geöffnet haben. Um auf die aktuell vom Cursor gelesene Zeile zu verweisen, verwenden Sie die Klausel WHERE CURRENT OF in Verbindung mit der Klausel FOR UPDATE.



# **Trigger implementieren – Untersuchung**

**ORACLE**

Copyright © 2013, Oracle und/oder verbundene Unternehmen. All rights reserved. Alle Rechte vorbehalten.

## Ziele

**Nach Beendigung dieses Anhangs haben Sie folgende Ziele erreicht:**

- **Datenbanksicherheit mit Triggern verbessern**
- **Datenintegrität mit DML-Triggern durchsetzen**
- **Referenzielle Integrität mit Triggern aufrechterhalten**
- **Daten mit Triggern zwischen Tabellen replizieren**
- **Abgeleitete Daten mit Triggern automatisch berechnen**
- **Funktionen zur Ereignisprotokollierung mit Triggern bereitstellen**



Copyright © 2013, Oracle und/oder verbundene Unternehmen. All rights reserved. Alle Rechte vorbehalten.

In dieser Lektion lernen Sie, wie Sie mit Datenbanktriggern Features erweitern, die andernfalls vom Oracle-Server nicht implementiert werden können. Je nach Situation können die vom Oracle-Server bereitgestellten Funktionen auch ausreichen, sodass Sie keine Trigger benötigen.

In dieser Lektion werden die folgenden Szenarios für Geschäftsanwendungen behandelt:

- Sicherheit
- Auditing
- Datenintegrität
- Referenzielle Integrität
- Tabellenreplikation
- Automatisches Berechnen abgeleiteter Daten
- Ereignisprotokollierung

## Sicherheit im Server steuern

Datenbanksicherheit mit der Anweisung **GRANT** aktivieren.

```
GRANT SELECT, INSERT, UPDATE, DELETE
 ON employees
 TO clerk; -- database role
GRANT clerk TO scott;
```

ORACLE

Copyright © 2013, Oracle und/oder verbundene Unternehmen. All rights reserved. Alle Rechte vorbehalten.

Mithilfe von Schemas und Rollen können Sie im Oracle-Server die Sicherheit von Datenvorgängen in Tabellen steuern und an der jeweiligen Benutzeridentität ausrichten.

- Berechtigungen basierend auf dem bei der Datenbankanmeldung angegebenen Benutzernamen vergeben
- Zugriff auf Tabellen, Views, Synonyme und Sequences festlegen
- Berechtigungen in Bezug auf Abfragen, Datenmanipulationen und Datendefinitionen festlegen

## Sicherheit mit Datenbanktriggern steuern

```
CREATE OR REPLACE TRIGGER secure_emp
 BEFORE INSERT OR UPDATE OR DELETE ON employees
DECLARE
 dummy PLS_INTEGER;
BEGIN
 IF (TO_CHAR (SYSDATE, 'DY') IN ('SAT','SUN')) THEN
 RAISE_APPLICATION_ERROR(-20506,'You may only
 change data during normal business hours.');
 END IF;
 SELECT COUNT(*) INTO dummy FROM holiday
 WHERE holiday_date = TRUNC (SYSDATE);
 IF dummy > 0 THEN
 RAISE_APPLICATION_ERROR(-20507,
 'You may not change data on a holiday.');
 END IF;
END;
/
```

ORACLE

Copyright © 2013, Oracle und/oder verbundene Unternehmen. All rights reserved. Alle Rechte vorbehalten.

Mithilfe von Triggern können Sie auch komplexeren Sicherheitsanforderungen gerecht werden.

- Berechtigungen basierend auf Datenbankwerten vergeben, beispielsweise Tageszeit oder Wochentag
- Zugriff auf Tabellen beschränken
- Nur Berechtigungen zur Datenmanipulation festlegen

## Datenintegrität im Server durchsetzen

```
ALTER TABLE employees ADD
CONSTRAINT ck_salary CHECK (salary >= 500);
```



Copyright © 2013, Oracle und/oder verbundene Unternehmen. All rights reserved. Alle Rechte vorbehalten.

Sie können Datenintegrität im Oracle-Server durchsetzen und Trigger entwickeln, mit denen auch komplexere Datenintegritätsregeln umgesetzt werden können.

Die als Standard vorgegebenen Datenintegritätsregeln sind NOT NULL, UNIQUE, PRIMARY KEY und FOREIGN KEY.

Mit diesen Regeln können Sie:

- konstante Standardwerte angeben
- statische Constraints durchsetzen
- dynamische Aktivierungen und Deaktivierungen durchführen

### Beispiel

Der im Beispiel aufgeführte Code stellt sicher, dass das Gehalt mindestens 500 \$ beträgt.

## Datenintegrität mit Triggern schützen

```
CREATE OR REPLACE TRIGGER check_salary
 BEFORE UPDATE OF salary ON employees
 FOR EACH ROW
 WHEN (NEW.salary < OLD.salary)
BEGIN
 RAISE_APPLICATION_ERROR (-20508,
 'Do not decrease salary.');
END;
/
```

ORACLE

Copyright © 2013, Oracle und/oder verbundene Unternehmen. All rights reserved. Alle Rechte vorbehalten.

Mit Triggern können Sie die Datenintegrität schützen und Datenintegritätsprüfungen durchsetzen, die nicht zu den Standardprüfungen gehören:

- Variable Standardwerte angeben
- Dynamische Constraints durchsetzen
- Dynamische Aktivierungen und Deaktivierungen durchführen
- Zum Schutz der Datenintegrität deklarative Constraints in die Tabellendefinition aufnehmen

### Beispiel

Der im Beispiel aufgeführte Code stellt sicher, dass das Gehalt nie gesenkt wird.

## Referenzielle Integrität im Server durchsetzen

```
ALTER TABLE employees
 ADD CONSTRAINT emp_deptno_fk
 FOREIGN KEY (department_id)
 REFERENCES departments(department_id)
 ON DELETE CASCADE;
```

ORACLE

Copyright © 2013, Oracle und/oder verbundene Unternehmen. All rights reserved. Alle Rechte vorbehalten.

Nehmen Sie referenzielle Integritäts-Constraints in die Tabellendefinition auf, um Inkonsistenz von Daten zu verhindern und referenzielle Integrität im Server durchzusetzen:

- Aktualisierungs- und Löschvorgänge einschränken
- Löschvorgänge kaskadieren
- Dynamische Aktivierungen und Deaktivierungen durchführen

### Beispiel

Wenn eine Abteilung aus der übergeordneten Tabelle DEPARTMENTS entfernt wird, kaskadieren Sie den Löschvorgang auf die entsprechenden Zeilen in der untergeordneten Tabelle EMPLOYEES.

## Referenzielle Integrität mit Triggern schützen

```
CREATE OR REPLACE TRIGGER cascade_updates
 AFTER UPDATE OF department_id ON departments
 FOR EACH ROW
BEGIN
 UPDATE employees
 SET employees.department_id=:NEW.department_id
 WHERE employees.department_id=:OLD.department_id;
 UPDATE job_history
 SET department_id=:NEW.department_id
 WHERE department_id=:OLD.department_id;
END;
/
```

ORACLE

Copyright © 2013, Oracle und/oder verbundene Unternehmen. All rights reserved. Alle Rechte vorbehalten.

Die folgenden referenziellen Integritätsregeln werden nicht von deklarativen Constraints unterstützt:

- Aktualisierungsvorgänge kaskadieren
- Für Aktualisierungs- und Löschvorgänge den Wert `NULL` einstellen
- Für Aktualisierungs- und Löschvorgänge einen Standardwert einstellen
- Referenzielle Integrität in einem verteilten System durchsetzen
- Dynamische Aktivierungen und Deaktivierungen durchführen

Um diese Integritätsregeln zu implementieren, können Sie Trigger entwickeln.

### Beispiel

Sie setzen die referenzielle Integrität mit einem Trigger durch. Wenn sich der Wert von `DEPARTMENT_ID` in der übergeordneten Tabelle `DEPARTMENTS` ändert, kaskadieren Sie den Aktualisierungsvorgang auf die entsprechenden Zeilen in der untergeordneten Tabelle `EMPLOYEES`.

Für eine vollständige und umfassende referenzielle Integritätslösung mithilfe von Triggern reicht ein einzelner Trigger nicht aus.

# Tabellen im Server replizieren

```
CREATE MATERIALIZED VIEW emp_copy
NEXT sysdate + 7
AS SELECT * FROM employees@ny;
```

ORACLE

Copyright © 2013, Oracle und/oder verbundene Unternehmen. All rights reserved. Alle Rechte vorbehalten.

## Materialized Views erstellen

In Materialized Views können Sie Kopien von Remote-Daten zu Replikationszwecken auf Ihrem lokalen Knoten verwalten. Daten werden in Materialized Views auf die gleiche Weise ausgewählt wie in herkömmlichen Datenbanktabellen oder Views. Materialized Views sind Datenbankobjekte, die die Ergebnisse einer Abfrage oder eine Kopie einer Datenbank zu einer Abfrage enthalten. In Abfragen von Materialized Views können in der Klausel `FROM` Tabellen, Views und andere Materialized Views benannt werden.

Bei Verwendung von Materialized Views wird die Replikation implizit vom Oracle-Server ausgeführt. So wird eine bessere Performance erreicht, als mit benutzerdefinierten PL/SQL-Triggern. Materialized Views:

- kopieren Daten aus lokalen und Remote-Tabellen asynchron in benutzerdefinierten Intervallen
- können auf mehreren Mastertabellen basieren
- sind standardmäßig schreibgeschützt, sofern nicht das Feature Oracle Advanced Replication verwendet wird
- verbessern die Performance der Datenmanipulation in der Mastertabelle

Alternativ können Sie Tabellen auch mithilfe von Triggern replizieren.

Das Beispiel auf der Folie erstellt eine Kopie der Remote-Tabelle `EMPLOYEES` aus New York. In der Klausel `NEXT` wird ein Datetime-Ausdruck für das Intervall zwischen automatischen Aktualisierungen angegeben.

# Tabellen mit Triggern replizieren

```

CREATE OR REPLACE TRIGGER emp_replica
 BEFORE INSERT OR UPDATE ON employees FOR EACH ROW
BEGIN /* Proceed if user initiates data operation,
NOT through the cascading trigger.*/
 IF INSERTING THEN
 IF :NEW.flag IS NULL THEN
 INSERT INTO employees@sf
 VALUES(:new.employee_id,...,'B');
 :NEW.flag := 'A';
 END IF;
 ELSE /* Updating. */
 IF :NEW.flag = :OLD.flag THEN
 UPDATE employees@sf
 SET ename=:NEW.last_name,...,flag=:NEW.flag
 WHERE employee_id = :NEW.employee_id;
 END IF;
 IF :OLD.flag = 'A' THEN :NEW.flag := 'B';
 ELSE :NEW.flag := 'A';
 END IF;
 END IF;
END;

```

**ORACLE**

Copyright © 2013, Oracle und/oder verbundene Unternehmen. All rights reserved. Alle Rechte vorbehalten.

Sie können Tabellen auch mit Triggern replizieren. Mit der Replikation können Sie:

- Tabellen synchron in Echtzeit kopieren
- Replikate auf eine einzige Mastertabelle basieren
- Daten aus Replikaten lesen und in Replikate schreiben

**Hinweis:** Die übermäßige Verwendung von Triggern kann die Performance der Datenmanipulation in der Mastertabelle beeinträchtigen, insbesondere, wenn Netzwerkfehler auftreten.

## Beispiel

Sie replizieren in New York die lokale Tabelle EMPLOYEES für San Francisco.

## Abgeleitete Daten im Server berechnen

```
UPDATE departments
 SET total_sal=(SELECT SUM(salary)
 FROM employees
 WHERE employees.department_id =
 departments.department_id);
```

ORACLE

Copyright © 2013, Oracle und/oder verbundene Unternehmen. All rights reserved. Alle Rechte vorbehalten.

Mit dem Server können Sie Batchjobs planen oder den Datenbank-Scheduler für die folgenden Szenarios verwenden:

- Abgeleitete Spaltenwerte asynchron in benutzerdefinierten Intervallen berechnen
- Abgeleitete Werte nur in Datenbanktabellen speichern
- Daten in einem Durchlauf in der Datenbank ändern und in einem zweiten Durchlauf abgeleitete Daten berechnen

Alternativ können Sie auch Trigger verwenden, um abgeleitete Daten fortlaufend berechnen zu lassen.

### Beispiel

Sie berechnen die Gehaltssumme für die einzelnen Abteilungen in einer speziellen Spalte (TOTAL\_SALARY) der Tabelle DEPARTMENTS.

## Abgeleitete Werte mit Triggern berechnen

```
CREATE PROCEDURE increment_salary
 (id NUMBER, new_sal NUMBER) IS
BEGIN
 UPDATE departments
 SET total_sal = NVL (total_sal, 0)+ new_sal
 WHERE department_id = id;
END increment_salary;
```

```
CREATE OR REPLACE TRIGGER compute_salary
AFTER INSERT OR UPDATE OF salary OR DELETE
ON employees FOR EACH ROW
BEGIN
 IF DELETING THEN increment_salary(
 :OLD.department_id, (-1*:OLD.salary));
 ELSIF UPDATING THEN increment_salary(
 :NEW.department_id, (:NEW.salary-:OLD.salary));
 ELSE increment_salary(
 :NEW.department_id,:NEW.salary); --INSERT
 END IF;
END;
```

**ORACLE**

Copyright © 2013, Oracle und/oder verbundene Unternehmen. All rights reserved. Alle Rechte vorbehalten.

## Abgeleitete Datenwerte mit Triggern berechnen

Mithilfe eines Triggers können Sie die folgenden Aufgaben ausführen:

- Abgeleitete Spalten synchron in Echtzeit berechnen
- Abgeleitete Werte in Datenbanktabellen oder globalen Packagevariablen speichern
- In einem einzigen Durchlauf Daten in der Datenbank ändern und abgeleitete Daten berechnen

### Beispiel

Sie berechnen die fortlaufende Gehaltssumme für die einzelnen Abteilungen in der speziellen Spalte `TOTAL_SALARY` der Tabelle `DEPARTMENTS`.

## Ereignisse mit Triggern protokollieren

```

CREATE OR REPLACE TRIGGER notify_reorder_rep
BEFORE UPDATE OF quantity_on_hand, reorder_point
ON inventories FOR EACH ROW
DECLARE
 dsc product_descriptions.product_description%TYPE;
 msg_text VARCHAR2(2000);
BEGIN
 IF :NEW.quantity_on_hand <=
 :NEW.reorder_point THEN
 SELECT product_description INTO dsc
 FROM product_descriptions
 WHERE product_id = :NEW.product_id;
 msg_text := 'ALERT: INVENTORY LOW ORDER:' ||
 'Yours,' || CHR(10) || user || '.' || CHR(10);
 ELSIF :OLD.quantity_on_hand >=
 :NEW.quantity_on_hand THEN
 msg_text := 'Product #' ||... CHR(10);
 END IF;
 UTL_MAIL.SEND('inv@oracle.com','ord@oracle.com',
 message=>msg_text, subject='Inventory Notice');
END;

```

**ORACLE**

Copyright © 2013, Oracle und/oder verbundene Unternehmen. All rights reserved. Alle Rechte vorbehalten.

Im Server können Sie Ereignisse über Datenabfragen und manuelle Vorgänge protokollieren. Wenn der Lagerbestand eines bestimmten Produkts die Mindestmenge unterschreitet, wird eine E-Mail verschickt. Dieser Trigger verwendet zum Senden der E-Mail-Nachricht das von Oracle bereitgestellte Package UTL\_MAIL.

### Ereignisse im Server protokollieren

1. Führen Sie eine explizite Datenabfrage aus, um festzustellen, ob ein Vorgang erforderlich ist.
2. Führen Sie den Vorgang aus, beispielsweise das Senden einer Nachricht.

### Trigger zur Ereignisprotokollierung verwenden

1. Führen Sie die Vorgänge implizit aus, beispielsweise den automatischen Versand eines elektronischen Memos.
2. Ändern Sie die Daten, und führen Sie im selben Schritt die damit zusammenhängenden Vorgänge aus.
3. Die Ereignisse werden automatisch protokolliert, wenn Datenänderungen auftreten.

## Ereignisse transparent protokollieren

Im Triggercode gilt:

- `CHR(10)` steht für einen Zeilenvorschub.
- `Reorder_point` ist `NULL`.
- Eine andere Transaktion kann die Meldung in der Pipe empfangen und lesen.

Beispiel

```

CREATE OR REPLACE TRIGGER notify_reorder_rep
BEFORE UPDATE OF amount_in_stock, reorder_point
ON inventory FOR EACH ROW
DECLARE
 dsc product.descrip%TYPE;
 msg_text VARCHAR2(2000);
BEGIN
 IF :NEW.amount_in_stock <= :NEW.reorder_point THEN
 SELECT descrip INTO dsc
 FROM PRODUCT WHERE prodid = :NEW.product_id;
 msg_text := 'ALERT: INVENTORY LOW ORDER:' || CHR(10) ||
 'It has come to my personal attention that, due to recent' ||
 'transactions, our inventory for product #' ||
 TO_CHAR(:NEW.product_id) || '-- ' || dsc ||
 ' -- has fallen below acceptable levels.' || CHR(10) ||
 'Yours,' || CHR(10) || user || '.' || CHR(10) || CHR(10);
 ELSIF :OLD.amount_in_stock >= :NEW.amount_in_stock THEN
 msg_text := 'Product #' || TO_CHAR(:NEW.product_id)
 || ' ordered. ' || CHR(10) || CHR(10);
 END IF;
 UTL_MAIL.SEND('inv@oracle.com', 'ord@oracle.com',
 message => msg_text, subject => 'Inventory Notice');
END;

```

## Zusammenfassung

In diesem Anhang haben Sie Folgendes gelernt:

- Datenbanksicherheit mit Triggern verbessern
- Datenintegrität mit DML-Triggern durchsetzen
- Referenzielle Integrität mit Triggern aufrechterhalten
- Daten mit Triggern zwischen Tabellen replizieren
- Abgeleitete Daten mit Triggern automatisch berechnen
- Funktionen zur Ereignisprotokollierung mit Triggern bereitstellen



Copyright © 2013, Oracle und/oder verbundene Unternehmen. All rights reserved. Alle Rechte vorbehalten.

In dieser Lektion wurden die Funktionen des Oracle-Datenbankservers hinsichtlich Sicherheit, Auditing, Datenintegrität, Replikation und Protokollierung im Detail verglichen. Außerdem wurde erläutert, wie Sie dieselben Features mit Datenbanktriggern implementieren und dabei die Funktionen des Datenbankservers erweitern können. Für einige Aktivitäten (z. B. das Berechnen abgeleiteter Daten) müssen Sie Trigger verwenden, da der Oracle-Server ohne einen gewissen Programmieraufwand nicht weiß, wie diese Art von Geschäftsregel implementiert werden soll.



# I

## **DBMS\_SCHEDULER- und HTP-Packages**

**ORACLE**

Copyright © 2013, Oracle und/oder verbundene Unternehmen. All rights reserved. Alle Rechte vorbehalten.

## Ziele

**Nach Beendigung dieses Anhangs haben Sie folgende Ziele erreicht:**

- **Mit dem HTP-Package einfache Webseiten generieren**
- **Package DBMS\_SCHEDULER für die Ausführungsplanung von PL/SQL-Code aufrufen**

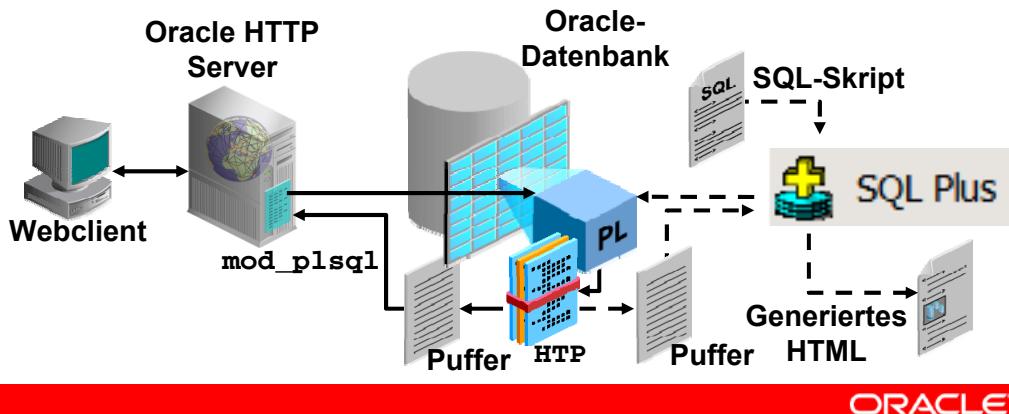


Copyright © 2013, Oracle und/oder verbundene Unternehmen. All rights reserved. Alle Rechte vorbehalten.

In dieser Lektion wird beschrieben, wie Sie einige der von Oracle bereitgestellten Packages und ihre Funktionsmöglichkeiten nutzen. Der Schwerpunkt dieser Lektion liegt auf Packages, die eine webbasierte Ausgabe generieren, und auf den bereitgestellten Funktionen zur Ausführungsplanung.

## Webseiten mit dem HTP-Package generieren

- Die im HTP-Package integrierten Prozeduren generieren HTML-Tags.
- Das Package HTP wird verwendet, um HTML-Dokumente dynamisch zu generieren. Der Aufruf erfolgt über:
  - Browser mit Oracle HTTP Server und PL/SQL-Gatewayservices (`mod_plsql`)
  - SQL\*Plus-Skript zur Anzeige der HTML-Ausgabe



Copyright © 2013, Oracle und/oder verbundene Unternehmen. All rights reserved. Alle Rechte vorbehalten.

Das Package HTP enthält Prozeduren zur Generierung von HTML-Tags. In der Regel umschließen die generierten HTML-Tags die Daten, die den verschiedenen Prozeduren als Parameter bereitgestellt werden. Die Folie veranschaulicht die beiden Verwendungsmöglichkeiten des Packages HTP:

- Zumeist werden Prozeduren mithilfe der PL/SQL-Gateway Services über die von Oracle HTTP Server bereitgestellte Komponente `mod_plsql` aufgerufen, die zu Oracle Application Server gehört (im Diagramm mit durchgezogenen Linien dargestellt).
- Als Alternative (im Diagramm mit gestrichelten Linien dargestellt) können Prozeduren aus SQL\*Plus aufgerufen werden, sodass die generierte HTML-Ausgabe angezeigt wird, die kopiert und in eine Datei eingefügt werden kann. In diesem Kurs wird dieses Verfahren angewendet, da in der Kursumgebung keine Oracle Application Server-Software installiert ist.

**Hinweis:** Die HTP-Prozeduren geben Informationen in einen Sessionpuffer aus, der sich auf dem Datenbankserver befindet. Im Kontext von Oracle HTTP Server empfängt `mod_plsql` nach Abschluss der Prozedur automatisch den Pufferinhalt, der anschließend als HTTP-Antwort an den Browser zurückgegeben wird. In SQL\*Plus müssen Sie Folgendes manuell ausführen:

- Befehl `SET SERVEROUTPUT ON`
- Prozedur zur HTML-Generierung im Puffer
- Prozedur `OWA_UTIL.SHOWPAGE` zur Anzeige des Pufferinhalts

## HTP-Packageprozeduren verwenden

- Ein oder mehrere HTML-Tags generieren. Beispiel:

```
htp.bold('Hello'); -- Hello
htp.print('Hi World'); -- Hi World
```

- Wohlgeformtes HTML-Dokument erstellen:

|                                                                                                                                                                                                                                                                       |                                                                                                                                                                  |
|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <pre>BEGIN   htp.htmlOpen;      -----&gt;   htp.headOpen;     -----&gt;   htp.title('Welcome');  --&gt;   htp.headClose;    -----&gt;   htp.bodyOpen;     -----&gt;   htp.print('My home page');   htp.bodyClose;   -----&gt;   htp.htmlClose;   -----&gt; END;</pre> | <b>-- Generates:</b> <pre>&lt;HTML&gt; &lt;HEAD&gt; &lt;TITLE&gt;Welcome&lt;/TITLE&gt; &lt;/HEAD&gt; &lt;BODY&gt; My home page &lt;/BODY&gt; &lt;/HTML&gt;</pre> |
|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------|

**ORACLE**

Copyright © 2013, Oracle und/oder verbundene Unternehmen. All rights reserved. Alle Rechte vorbehalten.

Das Package HTP verfügt über eine Struktur zur 1:1-Zuordnung von Prozeduren zu standardmäßigen HTML-Tags. Um beispielsweise Text auf einer Webseite fett anzuseigen, wird er in die HTML-Tags `<B>` und `</B>` eingeschlossen. Das erste Codefeld auf der Folie zeigt, wie das Wort Hello in HTML mit HTP.BOLD, der entsprechenden Prozedur des Packages HTP, fett formatiert wird. Die Prozedur HTP.BOLD nimmt einen Textparameter an und gewährleistet, dass dieser in der generierten HTML-Ausgabe in die entsprechenden HTML-Tags eingeschlossen wird.

Die Prozedur HTP.PRINT kopiert ihren Textparameter in den Puffer. Das Beispiel auf der Folie zeigt einen für die Prozedur HTP.PRINT bereitgestellten Parameter, der HTML-Tags enthält. Dieses Verfahren empfiehlt sich nur dann, wenn Sie HTML-Tags verwenden müssen, die nicht mithilfe der Prozeduren aus dem Package HTP generiert werden können.

Das zweite Beispiel auf der Folie zeigt einen PL/SQL-Block, mit dem das Grundformat eines HTML-Dokuments generiert wird. Das Beispiel veranschaulicht, wie die einzelnen Prozeduren die entsprechende HTML-Zeile im eingeschlossenen Textfeld auf der rechten Seite generieren.

Die Verwendung des Packages HTP hat den Vorteil, dass Sie wohlgeformte HTML-Dokumente erstellen und somit die manuelle Eingabe der HTML-Tags um die einzelnen Datenelemente entfällt.

**Hinweis:** Informationen zu allen Prozeduren des Packages HTP finden Sie im Dokument *PL/SQL Packages and Types Reference*.

# HTML-Dateien mit SQL\*Plus erstellen

**So erstellen Sie eine HTML-Datei mit SQL\*Plus:**

**1. Erstellen Sie ein SQL-Skript mit folgenden Befehlen:**

```
SET SERVEROUTPUT ON
ACCEPT procname PROMPT "Procedure: "
EXECUTE &procname
EXECUTE owa_util.showpage
UNDEFINE proc
```

- 2. Laden Sie das Skript in SQL\*Plus, und führen Sie es aus. Geben Sie dabei Werte für Austauschvariablen ein.**
- 3. Markieren Sie den im Browser generierten HTML-Text, und kopieren Sie ihn in eine HTML-Datei.**
- 4. Öffnen Sie die HTML-Datei im Browser.**

Copyright © 2013, Oracle und/oder verbundene Unternehmen. All rights reserved. Alle Rechte vorbehalten.

Das Beispiel auf der Folie zeigt, wie Sie HTML mit einer beliebigen Prozedur generieren und die Ausgabe in einer HTML-Datei speichern. Gehen Sie dazu wie folgt vor:

1. Aktivieren Sie die Serverausgabe mit dem Befehl `SET SERVEROUTPUT ON`. Andernfalls erhalten Sie bei der Ausführung von Prozeduren, die Aufrufe an das Package `HTP` beinhalten, Exception-Meldungen.
2. Führen Sie die Prozedur aus, die Aufrufe an das Package `HTP` enthält.  
**Hinweis:** Dieser Schritt führt *nur* zu einer Ausgabe, wenn die Prozedur Aufrufe an das Package `DBMS_OUTPUT` enthält.
3. Führen Sie die Prozedur `OWA_UTIL.SHOWPAGE` aus, um den Text anzuzeigen. Mit diesem Aufruf wird der aus dem Puffer generierte HTML-Inhalt angezeigt.

Der Befehl `ACCEPT` fordert zur Eingabe des Namens der auszuführenden Prozedur auf. Beim Aufruf von `OWA_UTIL.SHOWPAGE` werden die HTML-Tags im Browserfenster angezeigt.

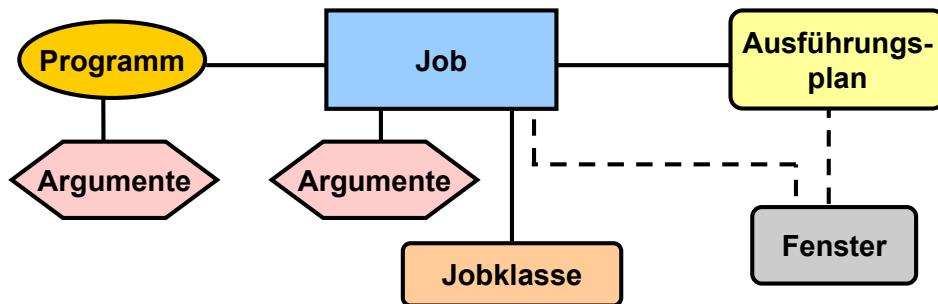
Anschließend können Sie die generierten HTML-Tags aus dem Browserfenster kopieren und in eine HTML-Datei (normalerweise mit der Erweiterung `.htm` oder `.html`) einfügen.

**Hinweis:** In SQL\*Plus können Sie die HTML-Ausgabe mit dem Befehl `SPOOL` direkt in eine HTML-Datei leiten.

## Package DBMS\_SCHEDULER

**Der Datenbank-Scheduler umfasst mehrere Komponenten für die Ausführung von Jobs. Mit dem Package DBMS\_SCHEDULER erstellen Sie die einzelnen Jobs mit einem:**

- eindeutigen Jobnamen
- Programm ("Was" soll ausgeführt werden?)
- Ausführungsplan ("Wann" soll der Job ausgeführt werden?)



**ORACLE**

Copyright © 2013, Oracle und/oder verbundene Unternehmen. All rights reserved. Alle Rechte vorbehalten.

Oracle Database stellt eine Reihe von Unterprogrammen im Package DBMS\_SCHEDULER bereit, die die Verwaltung vereinfachen und eine umfassende Funktionalität für komplexe Aufgaben der Ausführungsplanung bieten. Diese Unterprogramme werden zusammenfassend als Scheduler bezeichnet und können aus jedem PL/SQL-Programm aufgerufen werden. Mithilfe des Schedulers können Datenbankadministratoren und Anwendungsentwickler den Ausführungszeitpunkt bestimmter Tasks steuern. Indem Sie sicherstellen, dass viele routinemäßige Datenbanktasks ohne manuellen Eingriff durchgeführt werden, können Sie die Betriebskosten senken, verlässlichere Routinen implementieren und Benutzerfehler minimieren.

Das Diagramm zeigt die nachstehenden Komponenten des Schedulers:

- Ein **Job** ist die Kombination aus einem Programm und einem Ausführungsplan. Die für das Programm erforderlichen Argumente können mit dem Programm oder dem Job bereitgestellt werden. Alle Jobnamen weisen das Format [schema.]name auf. Beim Erstellen von Jobs geben Sie den Jobnamen, ein Programm, einen Ausführungsplan und (optional) Jobmerkmale an, die über eine **job class** bereitgestellt werden können.
- Ein **Programm** legt fest, was ausgeführt werden soll. Zu jedem automatisierten Job gehört ein bestimmtes ausführbares Programm, sei es ein PL/SQL-Block, eine Stored Procedure, ein natives binäres Programm oder ein Shellskript. Programme stellen Metadaten zu bestimmten ausführbaren Programmen bereit und benötigen möglicherweise eine Liste mit Argumenten.
- Ein **Schedule** oder Ausführungsplan gibt an, wann und wie oft ein Job ausgeführt wird.

- Eine **Jobklasse** definiert eine Kategorie von Jobs mit den gleichen Anforderungen bezüglich der Ressourcennutzung und weiteren gemeinsamen Merkmalen. Jeder Job kann jeweils nur einer Jobklasse angehören. Eine Jobklasse hat folgende Attribute:
  - **Servicename** der Datenbank. Die Jobs der Jobklasse haben eine Affinität zum angegebenen Service, das heißt, sie werden auf den Instanzen ausgeführt, die den angegebenen Service anbieten.
  - **Ressourcennutzungsgruppe**. Dieses Attribut klassifiziert eine Gruppe von Benutzersessions mit gemeinsamen Anforderungen an die Ressourcenverarbeitung. Eine Benutzersession oder Jobklasse kann stets nur einer einzigen Ressourcennutzungsgruppe angehören. Die der Jobklasse zugeordnete Ressourcennutzungsgruppe entscheidet darüber, welche Ressourcen der Jobklasse zugeordnet werden.
- Ein **Fenster** wird durch ein Zeitintervall mit festgelegtem Anfang und Ende dargestellt. Mit ihm werden zu verschiedenen Zeiten unterschiedliche Ressourcenpläne aktiviert.

Auf der Folie steht der Job als primäre Komponente im Mittelpunkt. Jedoch können Sie auch die Komponenten Programm, Ausführungsplan, Fenster und Jobklasse als individuelle Entitäten erstellen und den Jobs für die Ausführung durch den Scheduler zuordnen. Sie können Jobs so erstellen, dass sie alle erforderlichen Informationen inline enthalten, d. h. in dem Aufruf, mit dem der Job erstellt wird. Als Alternative können Sie bei der Joberstellung zuvor definierte Programme oder Ausführungspläne referenzieren. Beispiele dazu finden Sie auf den nächsten Seiten.

Weitere Informationen zum Scheduler finden Sie im Onlinekurs *Oracle Database: Configure and Manage Jobs with the Scheduler*.

## Jobs erstellen

- **Jobs können auf unterschiedliche Weise mit einer Kombination aus Inlineparametern, benannten Programmen und benannten Ausführungsplänen erstellt werden.**
- **Sie können Jobs mit der Prozedur CREATE\_JOB wie folgt erstellen:**
  - Mit als Parametern angegebenen Inlineinformationen ("was") und Ausführungsplänen
  - Mit benannten (gespeicherten) Programmen und inline unter Angabe des Ausführungsplanes
  - Mit inline auszuführenden Aktionen und benannten Ausführungsplänen
  - Mit benannten Programmen und Ausführungsplan-komponenten

**ORACLE**

Copyright © 2013, Oracle und/oder verbundene Unternehmen. All rights reserved. Alle Rechte vorbehalten.

Die Komponente, die für eine Ausführung zu einem bestimmten Zeitpunkt verantwortlich ist, wird als **Job** bezeichnet. Mit der Prozedur DBMS\_SCHEDULER.CREATE\_JOB des Packages DBMS\_SCHEDULER erstellen Sie einen Job, der standardmäßig deaktiviert ist. Jobs können erst nach ihrer expliziten Aktivierung für die Ausführung geplant werden. Um einen Job zu erstellen, gehen Sie wie folgt vor:

- Namen im Format [schema.]name angeben
- Berechtigung CREATE JOB erteilen

**Hinweis:** Benutzer mit der Berechtigung CREATE ANY JOB können Jobs in allen Schemas außer SYS erstellen. Für die Zuordnung von Jobs zu einer bestimmten Klasse ist für diese Klasse die Berechtigung EXECUTE erforderlich.

Vereinfacht ausgedrückt: Sie können Jobs erstellen, indem Sie alle Details – das auszuführende Programm (was) und den Ausführungsplan (wann) – in den Argumenten der Prozedur CREATE\_JOB angeben. Als Alternative können Sie die vordefinierten Komponenten Programm und Ausführungsplan verwenden. Wenn ein benanntes Programm und ein benannter Ausführungsplan vorhanden sind, können sie angegeben oder mit Inlineargumenten kombiniert werden, um eine größtmögliche Flexibilität bei der Joberstellung zu gewährleisten.

Die Angaben des Ausführungsplanes werden einer einfachen logischen Prüfung unterzogen (Prüfung der Datumsparameter bei der Joberstellung). Die Datenbank prüft, ob das Enddatum nach dem Anfangsdatum liegt. Bezieht sich das Anfangsdatum auf einen Zeitpunkt in der Vergangenheit, wird es in das aktuelle Datum geändert.

## Jobs mit Inlineparametern erstellen

**In den Argumenten der Prozedur CREATE\_JOB den Codetyp, den Code, die Anfangszeit und die Häufigkeit des auszuführenden Jobs angeben**

```
-- Schedule a PL/SQL block every hour:

BEGIN
 DBMS_SCHEDULER.CREATE_JOB(
 job_name => 'JOB_NAME',
 job_type => 'PLSQL_BLOCK',
 job_action => 'BEGIN ...; END;',
 start_date => SYSTIMESTAMP,
 repeat_interval=>'FREQUENCY=HOURLY;INTERVAL=1',
 enabled => TRUE);
END;
/
```

ORACLE

Copyright © 2013, Oracle und/oder verbundene Unternehmen. All rights reserved. Alle Rechte vorbehalten.

Mit der Prozedur DBMS\_SCHEDULER.CREATE\_JOB können Sie Jobs erstellen, um PL/SQL-Blöcke, Stored Procedures oder externe Programme auszuführen. Sie können die Prozedur CREATE\_JOB direkt verwenden, ohne ein Programm oder einen Ausführungsplan zu erstellen.

Das Beispiel auf der Folie zeigt, wie Sie alle Jobdetails inline angeben können. Die Parameter der Prozedur CREATE\_JOB definieren, "was" ausgeführt werden soll, sowie den Ausführungsplan und andere Jobattribute. Die nachstehenden Parameter definieren, was ausgeführt werden soll:

- Der Parameter `job_type` kann einen von drei Werten annehmen:
  - `PLSQL_BLOCK` für beliebige PL/SQL-Blöcke oder SQL-Anweisungen. Dieser Jobtyp kann keine Argumente annehmen.
  - `STORED_PROCEDURE` für Stored Procedures (Standalone oder in ein Package integriert). Die Prozeduren können Argumente annehmen, die mit dem Job bereitgestellt werden.
  - `EXECUTABLE` für über die Befehlszeile des Betriebssystems ausführbare Anwendungen
- Der Ausführungsplan wird mit folgenden Parametern angegeben:
  - `start_date` nimmt einen Zeitstempel an, `repeat_interval` ist eine Zeichenfolge, die als Kalender oder PL/SQL-Ausdruck dient. Die Angabe von `end_date` ist möglich.

**Hinweis:** Als `repeat_interval` angegebene Zeichenfolgenausdrücke werden im weiteren Verlauf des Kurses behandelt. Das Beispiel oben gibt ein stündliches Wiederholungsintervall für den Job vor.

## Jobs mit Programmen erstellen

- **Mit CREATE\_PROGRAM ein Programm erstellen:**

```
BEGIN
 DBMS_SCHEDULER.CREATE_PROGRAM(
 program_name => 'PROG_NAME',
 program_type => 'PLSQL_BLOCK',
 program_action => 'BEGIN ...; END;');
END;
```

- **Überladene Prozedur CREATE\_JOB mit dem Parameter program\_name verwenden:**

```
BEGIN
 DBMS_SCHEDULER.CREATE_JOB('JOB_NAME',
 program_name => 'PROG_NAME',
 start_date => SYSTIMESTAMP,
 repeat_interval => 'FREQ=DAILY',
 enabled => TRUE);
END;
```

ORACLE

Copyright © 2013, Oracle und/oder verbundene Unternehmen. All rights reserved. Alle Rechte vorbehalten.

Die Prozedur DBMS\_SCHEDULER.CREATE\_PROGRAM definiert ein Programm, dem ein eindeutiger Name zugeordnet werden muss. Wenn Sie das Programm separat für einen Job erstellen, können Sie:

- die Aktion einmal definieren und in mehreren Jobs wiederverwenden
- den Ausführungsplan für einen Job ändern, ohne den PL/SQL-Block neu erstellen zu müssen
- das ausgeführte Programm ändern, ohne alle Jobs ändern zu müssen

Die Zeichenfolge für die Programmaktion gibt eine Prozedur, eine ausführbare Anwendung oder einen PL/SQL-Block an, je nach dem Wert des Parameters `program_type`. Mögliche Werte:

- PLSQL\_BLOCK zum Ausführen anonymer Blöcke oder SQL-Anweisungen
- STORED\_PROCEDURE zum Ausführen von Stored Procedures wie PL/SQL, Java oder C
- EXECUTABLE zum Ausführen von Befehlszeilenprogrammen des Betriebssystems

Das Beispiel auf der Folie zeigt den Aufruf eines anonymen PL/SQL-Blockes. Sie können auch externe Prozeduren innerhalb von Programmen aufrufen, wie im folgenden Beispiel gezeigt:

```
DBMS_SCHEDULER.CREATE_PROGRAM(program_name => 'GET_DATE',
 program_action => '/usr/local/bin/date',
 program_type => 'EXECUTABLE');
```

Um einen Job mit einem Programm zu erstellen, geben Sie im Aufruf der Prozedur DBMS\_SCHEDULER.CREATE\_JOB den Programmnamen im Argument `program_name` an, wie im Beispiel auf der Folie gezeigt.

# Jobs für Programme mit Argumenten erstellen

- Programm erstellen:

```
DBMS_SCHEDULER.CREATE_PROGRAM(
 program_name => 'PROG_NAME',
 program_type => 'STORED_PROCEDURE',
 program_action => 'EMP_REPORT');
```

- Argument definieren:

```
DBMS_SCHEDULER.DEFINE_PROGRAM_ARGUMENT(
 program_name => 'PROG_NAME',
 argument_name => 'DEPT_ID',
 argument_position=> 1, argument_type=> 'NUMBER',
 default_value => '50');
```

- Job zur Angabe der Anzahl von Argumenten erstellen:

```
DBMS_SCHEDULER.CREATE_JOB ('JOB_NAME', program_name
 => 'PROG_NAME', start_date => SYSTIMESTAMP,
 repeat_interval => 'FREQ=DAILY',
 number_of_arguments => 1, enabled => TRUE);
```

**ORACLE**

Copyright © 2013, Oracle und/oder verbundene Unternehmen. All rights reserved. Alle Rechte vorbehalten.

Für Programme wie PL/SQL oder externe Prozeduren sind möglicherweise Eingabeargumente erforderlich. Mit der Prozedur DBMS\_SCHEDULER.DEFINE\_PROGRAM\_ARGUMENT definieren Sie Argumente für vorhandene Programme. Beispiele für Parameter der Prozedur DEFINE\_PROGRAM\_ARGUMENT:

- program\_name gibt ein vorhandenes Programm an, das geändert werden soll.
- argument\_name gibt einen eindeutigen Argumentnamen für das Programm an.
- argument\_position gibt die Position an, an der das Argument bei Aufruf des Programms übergeben wird.
- argument\_type gibt den Datentyp des Argumentwertes an, der an das aufgerufene Programm übergeben wird.
- default\_value gibt einen Standardwert an, der an das Programm übergeben wird, wenn der Job zur Planung der Programmausführung keinen Wert bereitstellt.

Die Folie zeigt, wie ein Job zur Programmausführung mit einem Argument erstellt wird. Der Standardwert des Programmarguments ist 50. So ändern Sie diesen Wert für einen Job:

```
DBMS_SCHEDULER.SET_JOB_ARGUMENT_VALUE (
 job_name => 'JOB_NAME',
 argument_name => 'DEPT_ID', argument_value => '80');
```

## Jobs mit Ausführungsplänen erstellen

- Mit **CREATE\_SCHEDULE** einen Ausführungsplan erstellen:

```
BEGIN
 DBMS_SCHEDULER.CREATE_SCHEDULE('SCHED_NAME' ,
 start_date => SYSTIMESTAMP,
 repeat_interval => 'FREQ=DAILY',
 end_date => SYSTIMESTAMP +15);
END;
```

- Mit **CREATE\_JOB** den Ausführungsplan im Parameter **schedule\_name** referenzieren:

```
BEGIN
 DBMS_SCHEDULER.CREATE_JOB('JOB_NAME' ,
 schedule_name => 'SCHED_NAME',
 job_type => 'PLSQL_BLOCK',
 job_action => 'BEGIN ...; END;',
 enabled => TRUE);
END;
```

**ORACLE**

Copyright © 2013, Oracle und/oder verbundene Unternehmen. All rights reserved. Alle Rechte vorbehalten.

Sie können einen allgemeinen Ausführungsplan erstellen und auf verschiedene Jobs anwenden, ohne die Details des Planes immer wieder angeben zu müssen. Die Erstellung eines Ausführungsplanes hat mehrere Vorteile:

- Er ist wiederverwendbar und kann verschiedenen Jobs zugeordnet werden.
- Änderungen am Plan betreffen alle Jobs, die ihn verwenden. So müssen die Ausführungspläne für die Jobs nur einmal geändert werden.

Ausführungspläne haben nur einen Genauigkeitsbereich von einer Sekunde. Der Datentyp **TIMESTAMP** ist zwar exakter, doch werden genauere Werte vom Scheduler auf die volle Sekunde auf- oder abgerundet.

Anfangs- und Endzeit eines Ausführungsplanes werden mit dem Datentyp **TIMESTAMP** angegeben. Das Enddatum (**end\_date**) für einen gespeicherten Ausführungsplan ist das Datum, nach dem der Ausführungsplan nicht mehr gültig ist. Der Ausführungsplan im Beispiel ist 15 Tage nach seiner Verwendung in einem angegebenen Job gültig.

Das Wiederholungsintervall (**repeat\_interval**) eines gespeicherten Ausführungsplanes muss mit einem Kalenderausdruck erstellt werden. Ein **NULL**-Wert für **repeat\_interval** gibt an, dass der Job nur ein Mal ausgeführt wird.

**Hinweis:** Sie können das Wiederholungsintervall eines gespeicherten Ausführungsplanes nicht mit PL/SQL-Ausdrücken angeben.

## Wiederholungsintervalle für Jobs einstellen

- Mit einem Kalenderausdruck:

```
repeat_interval=> 'FREQ=HOURLY; INTERVAL=4'
repeat_interval=> 'FREQ=DAILY'
repeat_interval=> 'FREQ=MINUTELY; INTERVAL=15'
repeat_interval=> 'FREQ=YEARLY;
 BYMONTH=MAR, JUN, SEP, DEC;
 BYMONTHDAY=15'
```

- Mit einem PL/SQL-Ausdruck:

```
repeat_interval=> 'SYSDATE + 36/24'
repeat_interval=> 'SYSDATE + 1'
repeat_interval=> 'SYSDATE + 15/(24*60)'
```

ORACLE

Copyright © 2013, Oracle und/oder verbundene Unternehmen. All rights reserved. Alle Rechte vorbehalten.

## Jobs mit benannten Programmen und Ausführungsplänen erstellen

- **Mit der Prozedur CREATE\_PROGRAM das benannte Programm PROG\_NAME erstellen**
- **Mit der Prozedur CREATE\_SCHEDULE das benannte Programm SCHED\_NAME erstellen**
- **Job erstellen, der das benannte Programm und den benannten Ausführungsplan referenziert:**

```
BEGIN
 DBMS_SCHEDULER.CREATE_JOB(
 'JOB_NAME',
 program_name => 'PROG_NAME',
 schedule_name => 'SCHED_NAME',
 enabled => TRUE);
END;
/
```

ORACLE

Copyright © 2013, Oracle und/oder verbundene Unternehmen. All rights reserved. Alle Rechte vorbehalten.

Das Beispiel auf der Folie zeigt die endgültige Form für die Verwendung der Prozedur DBMS\_SCHEDULER.CREATE\_JOB. Das benannte Programm (PROG\_NAME) und der benannte Ausführungsplan (SCHED\_NAME) werden hier in ihrem jeweiligen Parameter im Aufruf an die Prozedur DBMS\_SCHEDULER.CREATE\_JOB angegeben.

Dieses Beispiel verdeutlicht, wie einfach die Erstellung von Jobs mit vordefinierten Programmen und Ausführungsplänen ist.

Einige Jobs und Ausführungspläne sind zu komplex, um sie in diesem Kurs behandeln zu können. So können Sie beispielsweise Fenster für wiederkehrende Zeitpläne erstellen und eine Zuordnung zwischen Ressourcenplan und Fenster vornehmen. Ressourcenpläne definieren Attribute für die Ressourcen, die über den vom Ausführungsfenster definierten Zeitraum erforderlich sind.

Weitere Informationen zum Scheduler finden Sie im Onlinekurs *Oracle Database: Configure and Manage Jobs with the Scheduler*.

## Jobs verwalten

- Job ausführen:

```
DBMS_SCHEDULER.RUN_JOB('SCHEMA.JOB_NAME');
```

- Job stoppen:

```
DBMS_SCHEDULER.STOP_JOB('SCHEMA.JOB_NAME');
```

- Job löschen, selbst wenn er gerade ausgeführt wird:

```
DBMS_SCHEDULER.DROP_JOB('JOB_NAME' , TRUE);
```

**ORACLE**

Copyright © 2013, Oracle und/oder verbundene Unternehmen. All rights reserved. Alle Rechte vorbehalten.

Nach der Erstellung eines Jobs können Sie:

- den Job durch Aufruf der Prozedur `RUN_JOB` unter Angabe des Jobnamens ausführen. Der Job wird sofort in Ihrer aktuellen Session ausgeführt.
- den Job mit der Prozedur `STOP_JOB` stoppen. Wird der Job gerade ausgeführt, wird er sofort gestoppt. Die Prozedur `STOP_JOB` hat zwei Argumente:
  - **job\_name**: Ist der Name des zu stoppenden Jobs
  - **force**: Versucht, den Job ordnungsgemäß zu beenden. Wenn dies nicht gelingt und `force` auf `TRUE` eingestellt ist, wird der Jobunterprozess beendet. (Der Standardwert ist `FALSE`.) Um `force` verwenden zu können, benötigen Sie die Systemberechtigung `MANAGE SCHEDULER`.
- den Job mit der Prozedur `DROP_JOB` löschen. Diese Prozedur besitzt zwei Argumente:
  - **job\_name**: Der Name des Jobs, der gelöscht werden soll
  - **force**: Gibt an, ob der Job gestoppt und gelöscht werden soll, wenn er gerade ausgeführt wird. (Der Standardwert ist `FALSE`.)

Wenn der angegebene Job beim Aufruf der Prozedur `DROP_JOB` gerade ausgeführt wird, ist der Befehl nur erfolgreich, wenn `force` auf `TRUE` eingestellt ist. In diesem Fall wird jede aktive Instanz des Jobs gestoppt, und der Job wird gelöscht.

**Hinweis:** Um Jobs, die anderen Benutzern gehören, auszuführen, zu stoppen oder zu löschen, benötigen Sie `ALTER`-Berechtigungen oder die Systemberechtigung `CREATE ANY JOB`.

## Data Dictionary Views

- [DBA | ALL | USER]\_SCHEDULER\_JOBS
- [DBA | ALL | USER]\_SCHEDULER\_RUNNING\_JOBS
- [DBA | ALL]\_SCHEDULER\_JOB\_CLASSES
- [DBA | ALL | USER]\_SCHEDULER\_JOB\_LOG
- [DBA | ALL | USER]\_SCHEDULER\_JOB\_RUN\_DETAILS
- [DBA | ALL | USER]\_SCHEDULER\_PROGRAMS

ORACLE

Copyright © 2013, Oracle und/oder verbundene Unternehmen. All rights reserved. Alle Rechte vorbehalten.

Die View DBA\_SCHEDULER\_JOB\_LOG zeigt alle abgeschlossenen Jobinstanzen, sowohl die erfolgreichen als auch die nicht erfolgreichen.

Mit folgender Abfrage zeigen Sie den Zustand Ihrer Jobs an:

```
SELECT job_name, program_name, job_type, state
FROM USER_SCHEDULER_JOBS;
```

Mit folgender Abfrage stellen Sie fest, in welcher Instanz ein Job ausgeführt wird:

```
SELECT owner, job_name, running_instance, resource_consumer_group
FROM DBA_SCHEDULER_RUNNING_JOBS;
```

Mit folgender Abfrage ermitteln Sie, wie ein Job ausgeführt wurde:

```
SELECT job_name, instance_id, req_start_date, actual_start_date,
status
FROM ALL_SCHEDULER_JOB_RUN_DETAILS;
```

Mit folgender Abfrage ermitteln Sie den Status Ihrer Jobs:

```
SELECT job_name, status, error#, run_duration, cpu_used
FROM USER_SCHEDULER_JOB_RUN_DETAILS;
```

## Zusammenfassung

In diesem Anhang haben Sie Folgendes gelernt:

- Mit dem Package `HTP` einfache Webseiten generieren
- Package `DBMS_SCHEDULER` für die Ausführungsplanung von PL/SQL-Code aufrufen



Copyright © 2013, Oracle und/oder verbundene Unternehmen. All rights reserved. Alle Rechte vorbehalten.

In dieser Lektion wurde ein kleiner Teil der mit der Oracle-Datenbank bereitgestellten Packages behandelt. Sie haben mit `DBMS_OUTPUT` ausführliches Debugging durchgeführt und prozedural generierte Informationen auf dem Bildschirm in SQL\*Plus angezeigt.

Außerdem haben Sie gelernt, wie Sie mit dem Package `DBMS_SCHEDULER` die Ausführung von PL/SQL und externem Code planen.

**Hinweis:** Weitere Informationen zu allen PL/SQL-Packages und -Typen finden Sie im Dokument *PL/SQL Packages and Types Reference*.

