

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

Schulungsunterlagen • Band I

D80170DE10

Production 1.0

Oktober 2013

Bestellnummer: D83981

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	4-13
Packagebodys mithilfe von SQL Developer erstellen	4-14
Packagespezifikationen – Beispiel: comm_pkg	4-15
Packagebody erstellen	4-16
Packagebodys – Beispiel: comm_pkg	4-17
Packageunterprogramme aufrufen – Beispiele	4-18
Packageunterprogramme aufrufen – mit SQL Developer	4-19
Packages ohne Body erstellen und verwenden	4-20
Packages im Data Dictionary anzeigen	4-21
Packages in SQL Developer anzeigen	4-22
Packages mit SQL Developer oder der SQL-Anweisung DROP entfernen	4-23
Packages erstellen – Richtlinien	4-24
Quiz	4-25
Zusammenfassung	4-26
Übungen zu Lektion 4 – Überblick: Packages erstellen und verwenden	4-27

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-28
Equi Join	E-29
Datensätze mithilfe von Equi Joins abrufen	E-30
Suchbedingungen mit den Operatoren AND und WHERE hinzufügen	E-31
Datensätze mithilfe von Non-Equi Joins abrufen	E-32
Datensätze mithilfe von USING-Klauseln abrufen	E-33
Datensätze mithilfe von ON-Klauseln abrufen	E-34
Left Outer Join	E-35
Right Outer Join	E-36
Full Outer Join	E-37
Self-Joins – Beispiel	E-38
Cross Join	E-39
Zusammenfassung	E-40

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

1

Einführung

ORACLE

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

Ziele

Nach Ablauf dieser Lektion haben Sie folgende Ziele erreicht:

- **Ziele des Kurses erörtern**
- **Verfügbare Umgebungen in diesem Kurs bestimmen**
- **Datenbankschema und Datenbanktabellen beschreiben, die in diesem Kurs verwendet werden**
- **Funktionen und Vorteile von Oracle Cloud 12c beschreiben**
- **Verfügbare Dokumentationsunterlagen und Ressourcen auflisten**

ORACLE

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

PL/SQL unterstützt zahlreiche Programmkonstrukte. In dieser Lektion wiederholen Sie Programmeinheiten in Form von anonymen Blöcken und erhalten eine Einführung in benannte PL/SQL-Blöcke. Benannte PL/SQL-Blöcke werden auch als Unterprogramme bezeichnet. Zu den benannten PL/SQL-Blöcken gehören Prozeduren und Funktionen.

Die Tabellen aus dem (in den Kursübungen verwendeten) Schema **Human Resources (HR)** werden kurz erläutert. Außerdem werden die Entwicklungstools zum Schreiben, Testen und Debuggen von PL/SQL aufgelistet. Darüber hinaus beinhaltet der Kurs eine Einführung in Oracle Cloud.

Lektionsagenda

- **Kursziele und Kursagenda**
- In diesem Kurs verwendete Schemas und Anhänge sowie verfügbare PL/SQL-Entwicklungsumgebungen
- Oracle Database 12c und verwandte Produkte – Überblick
- Oracle-Dokumentation und zusätzliche Ressourcen



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

Kursziele

Nach Ablauf dieses Kurses haben Sie folgende Ziele erreicht:

- **Folgende Elemente erstellen, ausführen und verwalten:**
 - Prozeduren und Funktionen
 - Packagekonstrukte
 - Datenbanktrigger
- **PL/SQL-Unterprogramme und -Trigger verwalten**
- **Bildschirm- und Dateiausgaben mithilfe einiger von Oracle bereitgestellter Packages generieren**
- **Verschiedene Techniken bestimmen, die die Überlegungen zum PL/SQL-Codedesign beeinflussen**
- **PL/SQL-Compiler verwenden und Abhängigkeiten verwalten**



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

Mit Datenbankprozeduren können Sie modularisierte Anwendungen entwickeln. Hierfür verwenden Sie beispielsweise folgende Datenbankobjekte:

- Prozeduren und Funktionen
- Packages
- Datenbanktrigger

Modulare Anwendungen verbessern:

- Funktionalität
- Sicherheit
- Gesamtperformance

Empfohlene Kursagenda

1. Tag:

- **Lektion 1: Einführung**
- **Lektion 2: Prozeduren erstellen**
- **Lektion 3: Funktionen erstellen und Unterprogramme debuggen**
- **Lektion 4: Packages erstellen**

2. Tag:

- **Lektion 5: Mit Packages arbeiten**
- **Lektion 6: Von Oracle bereitgestellte Packages zur Anwendungsentwicklung**
- **Lektion 7: Dynamisches SQL**
- **Lektion 8: Überlegungen zum Design von PL/SQL-Code**

ORACLE

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

Empfohlene Kursagenda

3. Tag:

- **Lektion 9: Trigger erstellen**
- **Lektion 10: Komplexe, DDL- und Datenbankereignis-trigger erstellen**
- **Lektion 11: PL/SQL-Compiler**
- **Lektion 12: Abhangigkeiten verwalten**

ORACLE

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

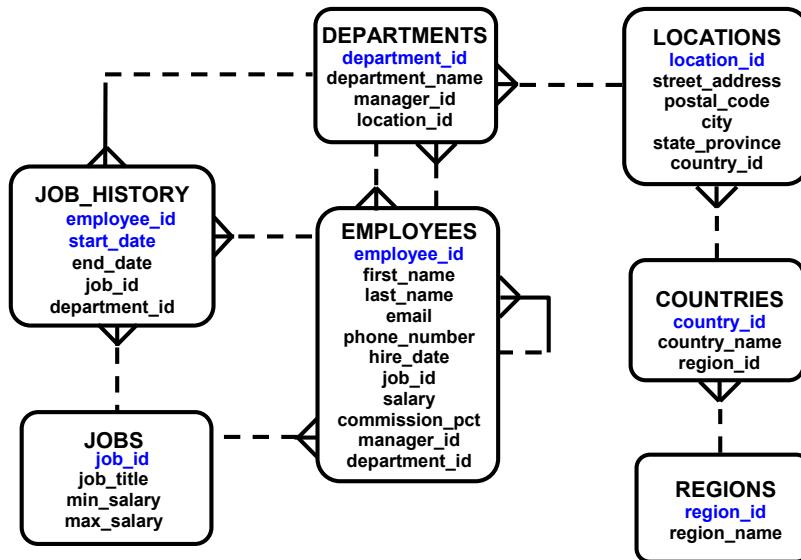
Lektionsagenda

- Kursziele und Kursagenda
- In diesem Kurs verwendete Schemas und Anhänge sowie verfügbare PL/SQL-Entwicklungsumgebungen
- Oracle Database 12c und verwandte Produkte – Überblick
- Oracle-Dokumentation und zusätzliche Ressourcen

ORACLE

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

Das in diesem Kurs verwendete Schema "Human Resources (HR)"



ORACLE

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

Schema "Human Resources (HR)" – Beschreibung

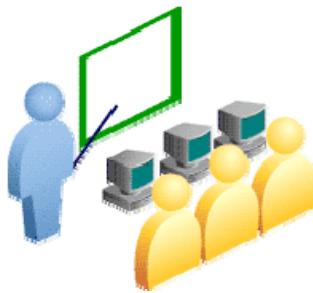
Das Schema **Human Resources (HR)** gehört zu den Oracle-Beispielschemas, die in einer Oracle-Datenbank installiert sein können. In den Übungen dieses Kurses werden Daten aus dem Schema HR verwendet.

Tabellenbeschreibungen

- **REGIONS** enthält Zeilen, die für eine Region wie Amerika, Asien usw. stehen.
- **COUNTRIES** enthält Zeilen für Länder, die jeweils einer Region zugeordnet sind.
- **LOCATIONS** enthält die spezifische Adresse eines bestimmten Büros, Lagers oder Produktionsstandortes eines in einem bestimmten Land ansässigen Unternehmens.
- **DEPARTMENTS** zeigt Details zu den Abteilungen, in denen Mitarbeiter arbeiten. Die Abteilungen können jeweils eine Beziehung zum Abteilungsleiter in der Tabelle **EMPLOYEES** aufweisen.
- **EMPLOYEES** enthält Details zu den einzelnen Mitarbeitern in einer Abteilung. Nicht alle Mitarbeiter müssen einer Abteilung zugeordnet sein.
- **JOBS** enthält die verschiedenen Tätigkeiten, die die einzelnen Mitarbeiter ausüben können.
- **JOB_HISTORY** enthält die Tätigkeitshistorie der Mitarbeiter. Wenn ein Mitarbeiter innerhalb einer Tätigkeit die Abteilung oder innerhalb einer Abteilung die Tätigkeit wechselt, wird eine neue Zeile mit den alten Tätigkeitsinformationen des Mitarbeiters in diese Tabelle eingefügt.

Informationen zu den Kursaccounts

- Es wurden geklonte HR-Account-IDs für Sie eingerichtet.
- Ihre Account-ID lautet ora61 oder ora62.
- Das Kennwort stimmt mit der Account-ID überein.
- Auf jedem Rechner ist eine eigene Datenbank installiert.
- Der Dozent verfügt über einen separaten Rechner.



ORACLE

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

Hinweis:

Verwenden Sie entweder die Account-ID ora61 oder ora62.

In diesem Kurs verwendete Anhänge

- **Anhang A: Tabellenbeschreibungen**
- **Anhang B: SQL Developer**
- **Anhang C: SQL*Plus**
- **Anhang D: REF-Cursor**
- **Anhang E: Häufig verwendete SQL-Befehle**
- **Anhang F: PL/SQL-Code verwalten**
- **Anhang G: PL/SQL – Wiederholung**
- **Anhang H: Trigger implementieren – Untersuchung**
- **Anhang I: DBMS_SCHEDULER- und HTTP-Packages**

ORACLE

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

PL/SQL-Entwicklungsumgebungen

Im Rahmen dieses Kurses werden die folgenden Tools zur Entwicklung von PL/SQL-Code bereitgestellt:

- **Oracle SQL Developer** (wird in diesem Kurs verwendet)
- **Oracle SQL*Plus**



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

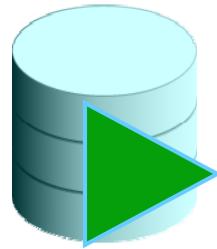
Es gibt viele Tools, die eine Umgebung zur Entwicklung von PL/SQL-Code bereitstellen. Auch Oracle bietet verschiedene Tools, mit denen Sie PL/SQL-Code erstellen können. In diesem Kurs können unter anderem die folgenden Entwicklungstools verwendet werden:

- **Oracle SQL Developer:** Grafisches Tool
- **Oracle SQL*Plus:** Fensterbasierte bzw. Befehlszeilenanwendung

Hinweis: Die im Kursmaterial dargestellten Code- und Bildschirmbeispiele stammen aus der Ausgabe der SQL Developer-Umgebung.

Was ist Oracle SQL Developer?

- Oracle SQL Developer ist ein kostenloses grafisches Tool zur Steigerung Ihrer Produktivität und zur Vereinfachung der Aufgaben bei der Datenbankentwicklung.
- Sie können sich mit der standardmäßigen Oracle-Datenbankauthentifizierung bei jedem Oracle-Zieldatenbankschema anmelden.
- In diesem Kurs wird SQL Developer verwendet.
- Anhang B enthält Details zur Verwendung von SQL Developer.



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 bei der Datenbankentwicklung vereinfacht. Mit wenigen Mausklicks können Sie problemlos Stored Procedures erstellen und verwalten, SQL-Anweisungen testen und Optimizer-Pläne anzeigen.

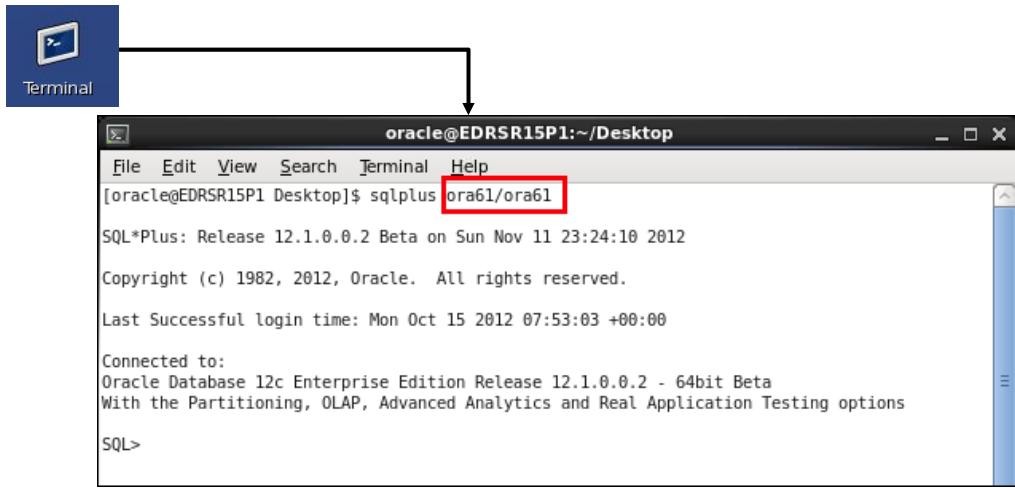
SQL Developer, das visuelle Tool zur Datenbankentwicklung, vereinfacht folgende Aufgaben:

- 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 und anschließend Vorgänge an den Objekten in der Datenbank ausführen.

Hinweis: In Anhang B dieses Kurses erhalten Sie eine Einführung in den Umgang mit der SQL Developer-Schnittstelle. Informieren Sie sich jetzt in Anhang B über das Erstellen einer Datenbankverbindung, die Interaktion mit Daten mittels SQL und PL/SQL und andere Themen.

PL/SQL in SQL*Plus codieren



ORACLE

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

Oracle SQL*Plus ist eine grafische Benutzeroberfläche (GUI) oder Befehlszeilenanwendung, mit der Sie SQL-Anweisungen und PL/SQL-Blöcke zur Ausführung weiterleiten und die Ergebnisse in einem Anwendungs- oder Befehlsfenster empfangen können.

SQL*Plus:

- ist im Lieferumfang der Datenbank enthalten
- wird auf einem Client und auf dem Datenbank-Serversystem installiert
- wird über ein Symbol oder die Befehlszeile aufgerufen

Beachten Sie die folgenden Aspekte, wenn Sie PL/SQL-Unterprogramme mit SQL*Plus codieren:

- Unterprogramme werden mit der SQL-Anweisung `CREATE` erstellt.
- Unterprogramme werden entweder über einen anonymen PL/SQL-Block oder den Befehl `EXECUTE` ausgeführt.
- Wenn Sie Text mit den Packageprozeduren `DBMS_OUTPUT` auf dem Bildschirm ausgeben, müssen Sie zunächst den Befehl `SET SERVEROUTPUT ON` in Ihrer Session ausführen.

Hinweis: Weitere Informationen zur Verwendung von SQL*Plus finden Sie im Anhang C.

Ausgaben von PL/SQL-Blöcken ermöglichen

1. Um Ausgaben in SQL Developer zu ermöglichen, folgenden Befehl vor dem PL/SQL-Block ausführen:

```
SET SERVEROUTPUT ON;
```

2. Ausgabe mithilfe des vordefinierten Oracle-Packages DBMS_OUTPUT und seiner Prozedur wie folgt anzeigen:
 - DBMS_OUTPUT.PUT_LINE

```
DBMS_OUTPUT.PUT_LINE('The First Name of the
Employee is ' || v_fname);
. . .
```

ORACLE

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

Im Beispiel auf der vorherigen Folie wurde ein Wert in der Variablen v_fname gespeichert. Der Wert wurde jedoch nicht ausgegeben.

PL/SQL verfügt über keine integrierte Eingabe- oder Ausgabefunktionalität. Daher werden vordefinierte Oracle-Packages für die Ein- und Ausgabe verwendet. Um Ausgaben zu generieren, gehen Sie wie folgt vor:

1. Führen Sie den folgenden SQL-Befehl aus:

```
SET SERVEROUTPUT ON
```

Hinweis: Um die Ausgabe in SQL*Plus zu ermöglichen, müssen Sie den Befehl SET SERVEROUTPUT ON explizit absetzen.

2. Um die Ausgabe anzuzeigen, verwenden Sie im PL/SQL-Block die Prozedur PUT_LINE des Packages DBMS_OUTPUT. Übergeben Sie den auszugebenden Wert als Argument an diese Prozedur, wie auf der Folie gezeigt. Die Prozedur gibt dann das Argument aus.

Lektionsagenda

- Kursziele und Kursagenda
- In diesem Kurs verwendete Schemas und Anhänge sowie verfügbare PL/SQL-Entwicklungsumgebungen
- **Oracle Database 12c und verwandte Produkte – Überblick**
- Oracle-Dokumentation und zusätzliche Ressourcen

ORACLE

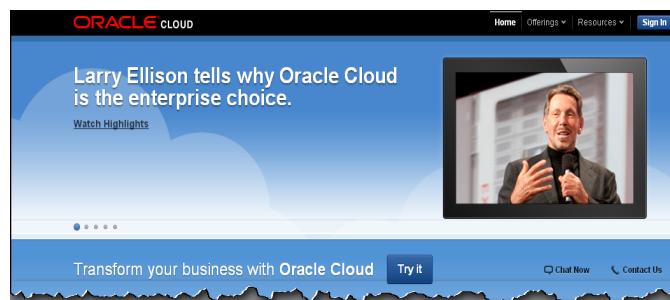
Copyright © 2013, Oracle und/oder verbundene Unternehmen. All rights reserved. Alle Rechte vorbehalten.

Oracle Cloud

Oracle Cloud ist eine Unternehmenscloud für die geschäftliche Nutzung. Sie besteht aus vielen verschiedenen Services, die sich durch einige gemeinsame Eigenschaften auszeichnen:

- **On-Demand Self-Service**
- **Ressourcen-Pooling**
- **Schnelle Elastizität**
- **Messbarer Service**
- **Breiter Netzwerkzugriff**

www.cloud.oracle.com



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

Oracle Cloud ist eine Unternehmenscloud für die geschäftliche Nutzung. Sie stellt eine integrierte Sammlung von Anwendungs- und Plattform-Cloudservices bereit, die auf den besten Produkten ihrer Klasse sowie offenen Java- und SQL-Standards basieren. Die zwei Hauptvorteile des Cloud-Computings liegen bei der Geschwindigkeit und den Kosten.

Dadurch sind die in Oracle Cloud bereitgestellten Anwendungen und Datenbanken portierbar. Sie können problemlos in eine oder aus einer Private Cloud oder Vor-Ort-Umgebung verschoben werden.

- Alle Oracle Cloud-Services können über eine Self-Service-Benutzeroberfläche durch Provisioning bereitgestellt werden. Benutzer können ihre Oracle Cloud-Services in einer integrierten Entwicklungs- und Bereitstellungsplattform mit Tools für die schnelle Erstellung und Erweiterung neuer Services erhalten.
- Oracle Cloud-Services werden in Oracle Exalogic Elastic Cloud und Oracle Exadata Database Machine erstellt. Zusammen ergeben sie eine Plattform, die höchste Performance, Redundanz und Skalierbarkeit bietet.

Dabei stechen fünf grundlegende Eigenschaften hervor:

- **On-Demand Self-Service:** Provisioning, Überwachung und Managementkontrolle
- **Ressourcen-Pooling:** Impliziert Freigaben und eine Abstraktionsebene zwischen Kunden und Services
- **Schnelle Elastizität:** Fähigkeit der bedarfsabhängigen vertikalen Skalierung
- **Messbarer Service:** Messung der Auslastung für interne Ausgleichsbuchungen (Private Cloud) oder externe Fakturierungen (Public Cloud)
- **Breiter Netzwerkzugriff:** Zugriff auf jedes Netzwerk-Device über einen Browser

Oracle Cloud-Services

Oracle Cloud bietet drei Typen von Services:

- **Software as a Service (SaaS)**
- **Platform as a Service (PaaS)**
- **Infrastructure as a Service (IaaS)**



ORACLE

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

Software as a Service (SaaS) bezieht sich im Allgemeinen auf Anwendungen, die Endbenutzern über das Internet zur Verfügung gestellt werden. Oracle CRM On Demand ist beispielsweise ein SaaS-Angebot, das je nach Kundenpräferenz mehrmandanten- oder einzelmandantenfähige Optionen bereitstellt.

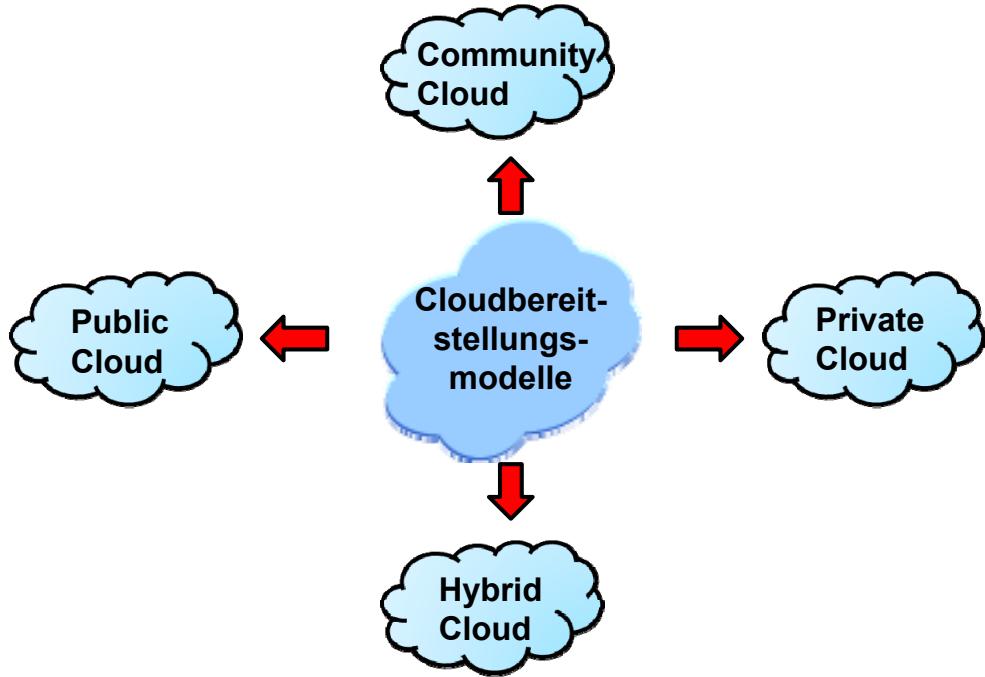
Platform as a Service (PaaS) bezieht sich im Allgemeinen auf eine Plattform für die Entwicklung und Bereitstellung von Anwendungen. Entwickler erhalten damit einen Service, der das schnelle Entwickeln und Bereitstellen einer SaaS-Anwendung für Endbenutzer erlaubt. Die Plattform umfasst in der Regel Datenbanken, Middleware und Entwicklungstools, die alle als Service über das Internet bereitgestellt werden.

Infrastructure as a Service (IaaS) bezieht sich auf Computing-Hardware (Server, Speicher und Netzwerk), die als Service bereitgestellt wird. Dazu gehören in der Regel auch die entsprechende Software sowie Betriebssysteme, Virtualisierung, Clustering usw. Zu den Beispielen für IaaS in der Public Cloud gehören Elastic Compute Cloud (EC2) und Simple Storage Service (S3) von Amazon.

Die Datenbankcloud wird innerhalb der Private Cloud-Umgebung eines Unternehmens als PaaS-Modell erstellt. Die Datenbankcloud bietet On-Demand-Zugriff auf Datenbankservices, der im Self-Service-Verfahren erfolgt, elastisch skalierbar ist und abgerechnet wird. Die Datenbankcloud hat überzeugende Vorteile hinsichtlich Kosten, Servicequalität und Agilität. Eine Datenbank kann auch innerhalb einer virtuellen Maschine auf einer IaaS-Plattform bereitgestellt werden.

Datenbankclouds können auf Oracle Exadata schnell bereitgestellt werden. Diese vorintegrierte und optimierte Hardwareplattform unterstützt OLTP- und DW-Workloads.

Cloubereitstellungsmodelle



ORACLE

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

- **Private Cloud:** Eine einzige Organisation verwendet eine Private Cloud, die sie in der Regel in privaten Data Centern steuert, verwaltet und hostet. Die Organisation kann das Hosting und den Betrieb jedoch auch an einen fremden Serviceprovider auslagern. Ein Beispiel für eine Private Cloud in einer externen Providerumgebung ist die Virtual Private Cloud von Amazon.
- **Public Cloud:** Mehrere Organisationen (Mandanten) verwenden eine Private Cloud auf gemeinsamer Basis. Die Cloud wird durch einen fremden Serviceprovider gehostet und verwaltet. Beispiele: Amazon Elastic Compute Cloud (EC2), IBM's Blue Cloud, Sun Cloud, Google AppEngine.
- **Community Cloud:** Eine Gruppe zusammengehöriger Organisationen, die eine gemeinsame Cloud-Computing-Umgebung nutzen möchte, verwendet die Community Cloud. Sie wird durch die teilnehmenden Organisationen oder einen fremden Serviceprovider verwaltet. Das Hosting erfolgt intern oder extern. Beispiel: Eine Community kann aus den verschiedenen Teilstreitkräften des Militärs, allen Universitäten in einer bestimmten Region oder allen Zulieferern eines großen Herstellers bestehen.
- **Hybrid Cloud:** Eine einzige Organisation, die Private und Public Clouds für eine einzige Anwendung nutzen möchte, verwendet die Hybrid Cloud. Ein drittes Modell, die Hybrid Cloud, wird durch interne und externe Provider betrieben. Beispiel: Eine Organisation könnte einen Public Cloud-Service wie Amazon Simple Storage Service (Amazon S3) für archivierte Daten verwenden, operative Kundendaten jedoch weiterhin intern speichern (Private Cloud).

Lektionsagenda

- **Kursziele und Kursagenda**
- **In diesem Kurs verwendete Schemas und Anhänge sowie verfügbare PL/SQL-Entwicklungsumgebungen**
- **Oracle Database 12c und verwandte Produkte – Überblick**
- **Oracle-Dokumentation und zusätzliche Ressourcen**

ORACLE

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

SQL- und PL/SQL-Dokumentation von Oracle

- ***Oracle Database New Features Guide***
- ***Oracle Database Advanced Application Developer's Guide***
- ***Oracle Database PL/SQL Language Reference***
- ***Oracle Database Reference***
- ***Oracle Database SQL Language Reference***
- ***Oracle Database Concepts***
- ***Oracle Database PL/SQL Packages and Types Reference***
- ***Oracle Database SQL Developer User's Guide***

ORACLE

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

Navigieren Sie zu <http://www.oracle.com/pls/db111/homepage>, und klicken Sie im linken Frame auf den Link **Master Book List**.

Zusätzliche Ressourcen

Weitere Informationen über die neuen SQL- und PL/SQL-Features in Oracle siehe:

- **Oracle Database: New Features Selfstudy**
- **Oracle By Example (OBE):**
 - http://apex.oracle.com/pls/apex/f?p=44785:2:0:FORCE_QUERY::2,RIR,%20%20CIR:P2_PRODUCT_ID,P2_PRODUCT_ID2:2011,3127
- **"What's New in PL/SQL in Oracle Database" im Oracle Technology Network (OTN):**
 - <http://www.oracle.com/technetwork/database/features/plsql/index.html>



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

Zusammenfassung

In dieser Lektion haben Sie Folgendes gelernt:

- **Ziele des Kurses erörtern**
- **Verfügbare Umgebungen in diesem Kurs bestimmen**
- **Datenbankschema und Datenbanktabellen beschreiben, die in diesem Kurs verwendet werden**
- **Verfügbare Dokumentationsunterlagen und Ressourcen auflisten**

ORACLE

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

Die PL/SQL-Sprache liefert verschiedene Programmkonstrukte für wiederverwendbare Codeblöcke. Mit unbenannten oder anonymen PL/SQL-Blöcken können Sie in SQL und PL/SQL erstellte Aktionen, Prozeduren, Funktionen und Packagekomponenten aufrufen. Zu den benannten PL/SQL-Blöcken, die auch als Unterprogramme bezeichnet werden, gehören:

- Prozeduren
- Funktionen
- Packageprozeduren und -funktionen
- Trigger

Oracle stellt verschiedene Tools zur Entwicklung der PL/SQL-Funktionalität bereit: eine clientseitige oder Middle Tier-PL/SQL-Laufzeitumgebung für Oracle Forms und Oracle Reports sowie eine PL/SQL-Runtime Engine innerhalb der Oracle-Datenbank. Prozeduren und Funktionen in der Datenbank können von jedem Anwendungscode aufgerufen werden, mit dem Sie sich bei einer Oracle-Datenbank anmelden und PL/SQL-Code ausführen können.

Übungen zu Lektion 1 – Überblick: Erste Schritte

Diese Übungen behandeln folgende Themen:

- **Verfügbare SQL Developer-Ressourcen prüfen**
- **SQL Developer starten, neue Datenbankverbindung erstellen und Schematabellen durchsuchen**
- **Verschiedene SQL Developer-Voreinstellungen festlegen**
- **SQL-Anweisungen und anonymen PL/SQL-Block mit dem SQL Worksheet ausführen**
- **Auf Oracle Database-Dokumentation sowie andere nützliche Websites zugreifen und Lesezeichen setzen**

ORACLE

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

In diesen Übungen führen Sie SQL-Anweisungen mithilfe von SQL Developer aus, um Daten im Schema zu prüfen. Außerdem erstellen Sie einen einfachen anonymen Block. Optional können Sie experimentieren, indem Sie den PL/SQL-Code in SQL*Plus erstellen und ausführen.

Hinweis: Für alle schriftlichen Übungen wird SQL Developer als Entwicklungsumgebung verwendet. SQL Developer wird als Umgebung empfohlen. Sie können jedoch auch die in diesem Kurs verfügbare SQL*Plus-Umgebung verwenden.

2

Prozeduren erstellen

ORACLE

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

Ziele

Nach Ablauf dieser Lektion haben Sie folgende Ziele erreicht:

- **Vorteile von modularisierten Unterprogrammen und Unterprogrammen mit Schichten bestimmen**
- **Prozeduren erstellen und aufrufen**
- **Formale und tatsächliche Parameter verwenden**
- **Parameter mithilfe positionaler, benannter oder gemischter Notation übergeben**
- **Verfügbare Modi für die Parameterübergabe bestimmen**
- **Exceptions in Prozeduren behandeln**
- **Prozeduren entfernen und ihre Informationen anzeigen**
- **Reine PL/SQL-Typen binden**



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

Lernziel

In dieser Lektion wird beschrieben, wie Sie Prozeduren mit oder ohne Parameter erstellen, ausführen und entfernen. Prozeduren sind die Grundlage für die modulare Programmierung in PL/SQL. Um Prozeduren flexibler zu machen, müssen variierende Daten entweder berechnet oder mit Eingabeparametern an eine Prozedur übergeben werden. Berechnete Ergebnisse können mit OUT-Parametern an den Aufrufer einer Prozedur zurückgegeben werden.

Um robuste Programme zu erhalten, sollten Sie Exception-Bedingungen immer mit den Exception-Behandlungsfeatures von PL/SQL verwalten.

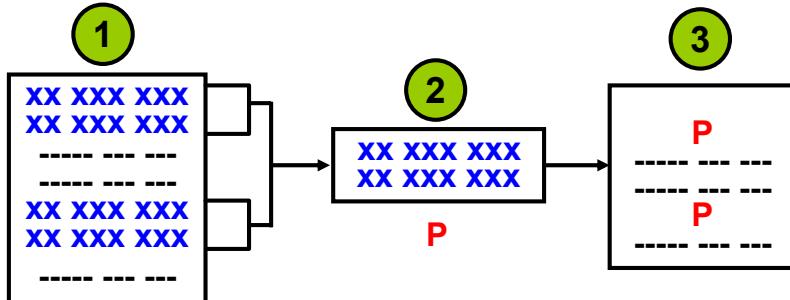
Lektionsagenda

- **Modularisierte Unterprogramme sowie Unterprogramme mit Schichten verwenden und Vorteile von Unterprogrammen bestimmen**
- **Mit Prozeduren arbeiten:**
 - Prozeduren erstellen und aufrufen
 - Verfügbare Modi für die Parameterübergabe bestimmen
 - Formale und tatsächliche Parameter verwenden
 - Positionale, benannte oder gemischte Notation verwenden
- **Exceptions in Prozeduren behandeln, Prozeduren entfernen und Prozedurinformationen anzeigen**
- **Reine PL/SQL-Typen binden**

ORACLE®

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

Modularisierte Unterprogramme erstellen



Code in Unterprogramme modularisieren:

1. Codesequenzen suchen, die mehrfach wiederholt werden
2. Unterprogramm P erstellen, das den wiederholten Code enthält
3. Ursprünglichen Code so ändern, dass das neue Unterprogramm aufgerufen wird

ORACLE

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

Modularisierte Unterprogramme und Unterprogramme mit Schichten erstellen

Das Diagramm veranschaulicht das Prinzip der Modularisierung mit Unterprogrammen: Flexibler und wiederverwendbarer Code wird in kleinere verwaltbare Teile aufgeteilt. Um Flexibilität zu erreichen, werden Unterprogramme mit Parametern verwendet, sodass derselbe Code für verschiedene Eingabewerte wiederverwendet werden kann. Um vorhandenen Code zu modularisieren, gehen Sie wie folgt vor:

1. Suchen und identifizieren Sie wiederkehrende Codesequenzen.
2. Verschieben Sie den sich wiederholenden Code in ein PL/SQL-Unterprogramm.
3. Ersetzen Sie den ursprünglichen sich wiederholenden Code durch Aufrufe des neuen PL/SQL-Unterprogramms.

Durch die Anordnung des Codes in Modulen und Schichten lässt sich der Code einfacher verwalten, insbesondere wenn sich die Geschäftsregeln ändern. Wenn Sie darüber hinaus die SQL-Logik einfach und ohne komplexe Geschäftslogik konzipieren, können Sie auch von der Arbeit von Oracle Database Optimizer profitieren. Diese Anwendung kann geparsete SQL-Anweisungen wiederverwenden, um serverseitige Ressourcen besser zu nutzen.

Unterprogramme mit Schichten erstellen

Unterprogrammschichten für die Anwendung erstellen:

- **Unterprogrammschicht mit SQL-Logik für den Datenzugriff**
- **Unterprogrammschicht für Geschäftslogik, die gegebenenfalls die Datenzugriffsschicht verwenden kann**

ORACLE

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

Da PL/SQL die nahtlose Einbettung von SQL-Anweisungen in die Logik ermöglicht, kann es schnell passieren, dass überall im Code SQL-Anweisungen verteilt sind. Es empfiehlt sich jedoch, die SQL-Logik von der Geschäftslogik zu trennen. Zu diesem Zweck müssen Sie ein Anwendungsdesign entwickeln, das über mindestens zwei Schichten verfügt:

- **Schicht für den Datenzugriff:** Für untergeordnete Routinen, die über SQL-Anweisungen auf die Daten zugreifen
- **Schicht für die Geschäftslogik:** Für Unterprogramme, die die geschäftlichen Verarbeitungsregeln implementieren, die wiederum die Routinen der Datenzugriffsschicht aufrufen können

Entwicklung mit PL/SQL-Blöcken modularisieren

- **PL/SQL ist eine blockstrukturierte Sprache. Der PL/SQL-Codeblock unterstützt die Modularisierung des Codes durch:**
 - anonyme Blöcke
 - Prozeduren und Funktionen
 - Packages
 - Datenbanktrigger
- **Vorteile der Verwendung von modularen Programmkonstrukten:**
 - Einfache Verwaltung
 - Verbesserte Datensicherheit und -integrität
 - Verbesserte Performance
 - Klarere Codestrukturierung

ORACLE®

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

Unterprogramme basieren auf PL/SQL-Standardstrukturen. Sie bestehen aus einem deklarativen Bereich, einem ausführbaren Bereich und einem optionalen Bereich zur Exception-Behandlung (beispielsweise anonyme Blöcke, Prozeduren, Funktionen, Packages und Trigger).

Unterprogramme können kompiliert und in der Datenbank gespeichert werden. Sie bieten Modularität, Erweiterbarkeit, Wiederverwendbarkeit und Verwaltbarkeit.

Bei der Modularisierung werden große Codeblöcke in kleinere Codegruppen, sogenannte Module, konvertiert. Nach der Modularisierung können die Module vom selben Programm wieder verwendet oder mit anderen Programmen gemeinsam verwendet werden. Code, der aus kleineren Modulen besteht, lässt sich einfacher verwalten und debuggen als Code in einem einzigen großen Programm. Module lassen sich darüber hinaus einfacher anpassen und erweitern. Falls nötig kann weitere Funktionalität aufgenommen werden, ohne dass die übrigen Module des Programms betroffen sind.

Unterprogramme erleichtern die Verwaltung, da sich der Code an einem einzigen Speicherort befindet und erforderliche Änderungen am Unterprogramm zentral vorgenommen werden können. Unterprogramme gewährleisten eine höhere Datenintegrität und -sicherheit. Auf die Datenobjekte wird über das Unterprogramm zugegriffen. Benutzer können das Unterprogramm nur aufrufen, wenn ihnen die entsprechenden Zugriffsberechtigungen erteilt wurden.

Hinweis: Als Voraussetzung für diesen Kurs müssen Sie wissen, wie anonyme Blöcke entwickelt werden. Weitere Informationen zu anonymen Blöcken finden Sie im Kurs *Oracle: PL/SQL Fundamentals*.

Anonyme Blöcke – Überblick

Anonyme Blöcke:

- bilden die grundlegende PL/SQL-Blockstruktur
- starten PL/SQL-Verarbeitungsaufgaben in Anwendungen
- können im ausführbaren Bereich beliebiger PL/SQL-Blöcke verschachtelt werden

```
[DECLARE      -- Declaration Section (Optional)
    variable declarations; ... ]
BEGIN        -- Executable Section (Mandatory)
    SQL or PL/SQL statements;
[EXCEPTION    -- Exception Section (Optional)
    WHEN exception THEN statements; ]
END;         -- End of Block (Mandatory)
```

ORACLE

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

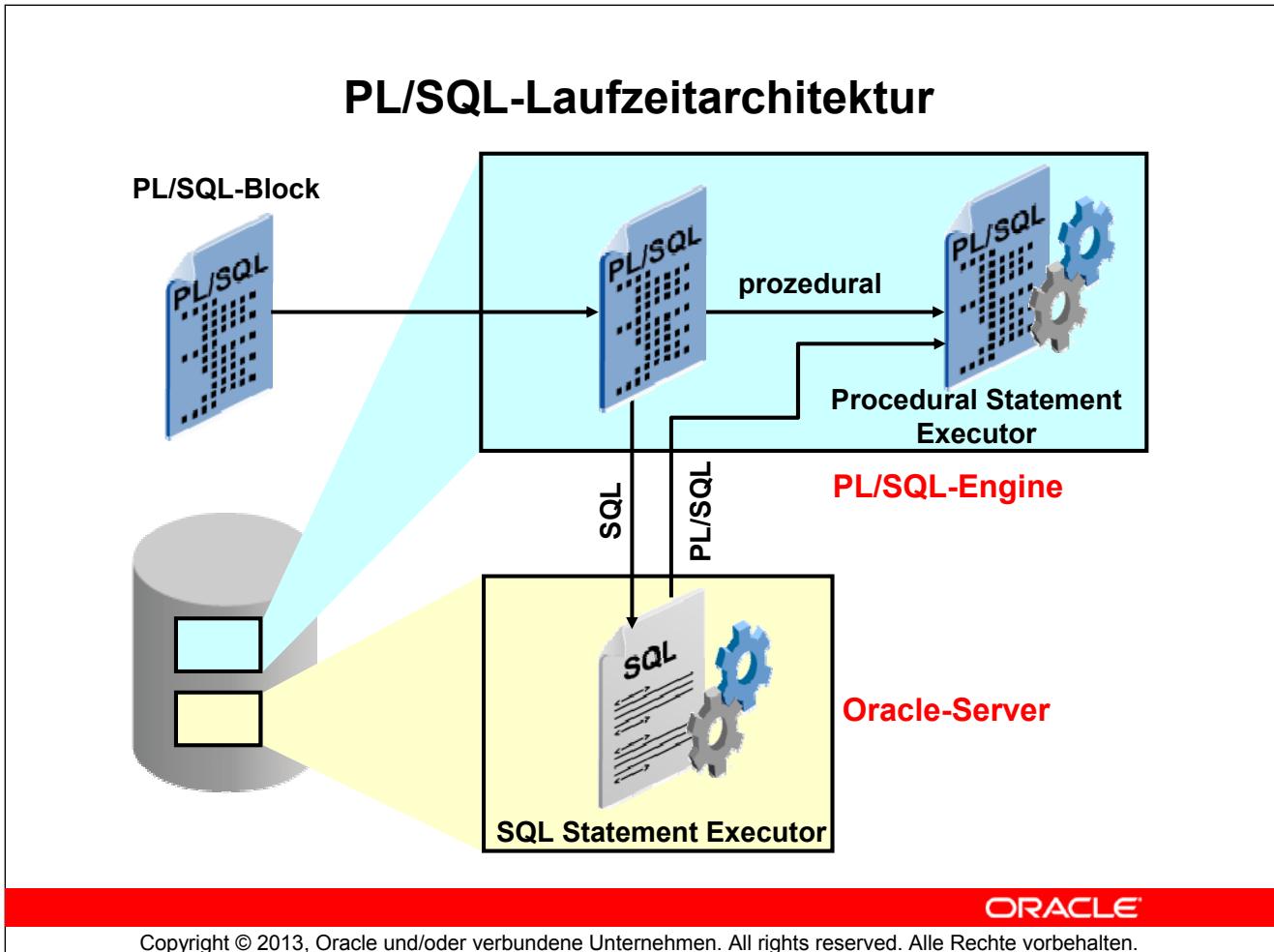
Anonyme Blöcke werden in der Regel für die folgenden Aufgaben verwendet:

- Triggercode für Oracle Forms-Komponenten erstellen
- Aufrufe von Prozeduren, Funktionen und Packagekonstrukten starten
- Exception-Behandlung in einem Codeblock isolieren
- Verschachtelung in anderen PL/SQL-Blöcken zur Verwaltung der Codeablaufsteuerung

Das Schlüsselwort `DECLARE` ist optional, jedoch obligatorisch, wenn Sie Variablen, Konstanten und Exceptions deklarieren, die innerhalb des PL/SQL-Blockes verwendet werden sollen.

`BEGIN` und `END` sind obligatorisch. Sie umschließen mindestens eine Anweisung (SQL und/oder PL/SQL).

Der Exception-Bereich ist optional. Er dient der Behandlung von Fehlern, die innerhalb des Gültigkeitsbereichs des PL/SQL-Blockes auftreten. Exceptions können an den Aufrufer des anonymen Blockes propagiert werden, indem ein Exception Handler für die spezifische Exception ausgeschlossen wird. Dadurch wird eine sogenannte *nicht behandelte* Exception erstellt.



Das Diagramm zeigt einen PL/SQL-Block, der von der PL/SQL-Engine ausgeführt wird. Die PL/SQL-Engine befindet sich:

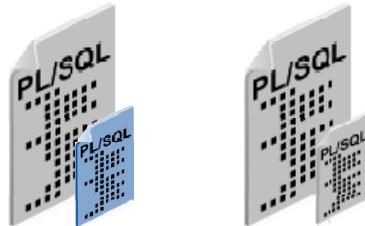
- in der Oracle-Datenbank zur Ausführung von Stored Subprograms
- im Oracle Forms-Client, wenn Client-/Serveranwendungen ausgeführt werden, oder in Oracle Application Server, wenn Oracle Forms Services zur Ausführung von Forms im Web verwendet wird

Unabhängig von der PL/SQL-Laufzeitumgebung bleibt die grundlegende Architektur stets gleich. Daher werden alle PL/SQL-Anweisungen im Procedural Statement Executor verarbeitet. Alle SQL-Anweisungen müssen zur Verarbeitung durch die Oracle-Serverprozesse an den SQL Statement Executor gesendet werden. Die SQL-Umgebung kann auch die PL/SQL-Umgebung aufrufen, beispielsweise wenn eine Funktion in einer SELECT-Anweisung verwendet wird.

Die PL/SQL-Engine ist eine virtuelle Maschine, die im Speicher gehalten wird und in PL/SQL erstellte m-Code-Anweisungen verarbeitet. Wenn die PL/SQL-Engine auf eine SQL-Anweisung trifft, erfolgt ein Kontextwechsel, um die SQL-Anweisung an die Oracle-Serverprozesse zu übergeben. Die PL/SQL-Engine wartet, bis die SQL-Anweisung abgeschlossen ist und die Ergebnisse zurückgegeben werden. Dann fährt sie mit der Verarbeitung nachfolgender Anweisungen im PL/SQL-Block fort. Die Oracle Forms-PL/SQL-Engine wird bei der Client-/Serverimplementierung im Client und bei der Forms Services-Implementierung im Anwendungsserver ausgeführt. In beiden Fällen werden SQL-Anweisungen in der Regel zur Verarbeitung über ein Netzwerk an einen Oracle-Server gesendet.

Was sind PL/SQL-Unterprogramme?

- Ein PL/SQL-Unterprogramm ist ein benannter PL/SQL-Block, der mit einer Gruppe von Parametern aufgerufen werden kann.
- Ein Unterprogramm lässt sich entweder in einem PL/SQL-Block oder einem anderen Unterprogramm deklarieren und definieren.
- Ein Unterprogramm besteht aus einer Spezifikation und einem Body.
- Ein Unterprogramm kann eine Prozedur oder eine Funktion sein.
- Im Allgemeinen verwenden Sie eine Prozedur zum Durchführen einer Aktion, während Sie mit einer Funktion einen Wert berechnen und zurückgeben.
- Unterprogramme können in PL/SQL-Packages gruppiert werden.



ORACLE

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

Ein PL/SQL-Unterprogramm ist ein benannter PL/SQL-Block, der mit einer Gruppe von Parametern aufgerufen werden kann. Ein Unterprogramm lässt sich entweder in einem PL/SQL-Block oder einem anderen Unterprogramm deklarieren und definieren.

Bestandteile von Unterprogrammen: Ein Unterprogramm besteht aus einer Spezifikation und einem Body. Für die Deklaration eines Unterprogramms müssen Sie die Spezifikation bereitstellen, die unter anderem Beschreibungen aller Parameter enthält. Für die Definition eines Unterprogramms sind sowohl die Spezifikation als auch der Body bereitzustellen. Sie können ein Unterprogramm vorab deklarieren und später im selben Block oder Unterprogramm definieren oder aber zum selben Zeitpunkt deklarieren und definieren.

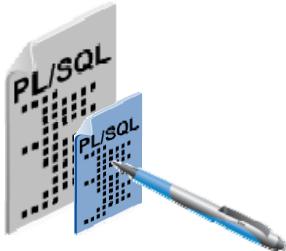
Typen von Unterprogrammen: PL/SQL kennt zwei Typen von Unterprogrammen: Prozeduren und Funktionen. Im Allgemeinen verwenden Sie eine Prozedur zum Durchführen einer Aktion, während Sie mit einer Funktion einen Wert berechnen und zurückgeben.

Prozeduren und Funktionen weisen dieselbe Struktur auf, allerdings enthalten nur Funktionen einige zusätzliche Elemente wie die Klausel RETURN oder die Anweisung RETURN.

Die Klausel RETURN gibt den Datentyp des Rückgabewertes an (obligatorisch). Die Anweisung RETURN gibt den Rückgabewert an (obligatorisch). In der nächsten Lektion, "Funktionen erstellen und Unterprogramme debuggen", wird ausführlich auf Funktionen eingegangen.

Unterprogramme lassen sich in PL/SQL-Packages gruppieren, was die Wiederverwendbarkeit und die Verwaltbarkeit von Code weiter verbessert. Ausführlichere Informationen finden Sie in den Lektionen zu Packages (Lektionen 4 und 5).

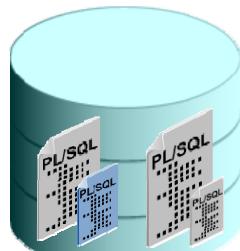
PL/SQL-Unterprogramme – Vorteile



Einfache Verwaltung



Klarere Codestrukturierung



**Unterprogramme:
Stored Procedures
und Stored Functions**



**Verbesserte
Datensicherheit und
-integrität**



**Verbesserte
Performance**

ORACLE®

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

Unterprogramme – Vorteile

Aufgrund der Modularisierung des Codes bieten Prozeduren und Funktionen zahlreiche Vorteile:

- **Einfache Verwaltung:** Alle Unterprogramme befinden sich an einem einzigen Speicherort. Änderungen müssen nur an einer Stelle vorgenommen werden und wirken sich auf mehrere Anwendungen aus. Die übermäßige Zahl an Testvorgängen kann reduziert werden.
- **Verbesserte Datensicherheit:** Der indirekte Zugriff auf Datenbankobjekte durch nicht berechtigte Benutzer wird durch Sicherheitsberechtigungen kontrolliert. Die Unterprogramme werden standardmäßig mit den Rechten des Eigentümers ausgeführt. Die EXECUTE-Berechtigung gestattet den aufrufenden Benutzern keinen direkten Zugriff auf Objekte, die für die Unterprogramme zugänglich sind.
- **Datenintegrität:** Zusammengehörige Aktionen werden entweder gemeinsam oder gar nicht ausgeführt.
- **Verbesserte Performance:** Der geparsste PL/SQL-Code, der im Shared SQL-Bereich des Servers verfügbar wird, kann wiederverwendet werden. Bei nachfolgenden Aufrufen des Unterprogramms muss der Code nicht erneut geparsst werden. Da der PL/SQL-Code bei der Komplilierung geparsst wird, lässt sich der Parsing-Overhead von SQL-Anweisungen zur Laufzeit vermeiden. Sie können Code erstellen, der die Anzahl der Netzwerkaufrufe an die Datenbank und damit den Netzwerkverkehr reduziert.
- **Klarere Codestrukturierung:** Durch die Verwendung geeigneter Namen und Konventionen zur Beschreibung der von den Routinen durchgeführten Aktionen werden weniger Kommentare benötigt, und die Lesbarkeit des Codes wird verbessert.

Anonyme Blöcke und Unterprogramme – Unterschiede

Anonyme Blöcke	Unterprogramme
Unbenannte PL/SQL-Blöcke	Benannte PL/SQL-Blöcke
Werden stets neu kompiliert	Werden nur einmal kompiliert
Werden nicht in der Datenbank gespeichert	Werden in der Datenbank gespeichert
Können von anderen Anwendungen nicht aufgerufen werden	Benannt und daher von anderen Anwendungen aufrufbar
Geben keine Werte zurück	Unterprogramme, die Funktionen sind, müssen Werte zurückgeben.
Können keine Parameter annehmen	Können Parameter annehmen

ORACLE

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

In der Tabelle auf der Folie werden nicht nur die Unterschiede zwischen anonymen Blöcken und Unterprogrammen beleuchtet, sondern auch die allgemeinen Vorteile von Unterprogrammen.

Anonyme Blöcke sind keine persistenten Datenbankobjekte. Sie werden nur einmal kompiliert und ausgeführt. Sie werden nicht zur Wiederverwendung in der Datenbank gespeichert. Wenn Sie einen anonymen Block wiederverwenden möchten, müssen Sie das Skript zu seiner Erstellung erneut ausführen. Dies bewirkt eine Rekomplilierung und Ausführung.

Prozeduren und Funktionen werden kompiliert und in der Datenbank in kompilierter Form gespeichert. Sie werden nur bei Änderungen rekompiliert. Da Unterprogramme in der Datenbank gespeichert sind, stehen sie jeder Anwendung bei entsprechender Berechtigung zur Verfügung. Die aufrufende Anwendung kann Parameter an die Prozeduren übergeben, sofern diese auf die Annahme von Parametern ausgelegt sind. Analog kann eine aufrufende Anwendung einen Wert abrufen, wenn sie eine Funktion oder Prozedur auuft.

Lektionsagenda

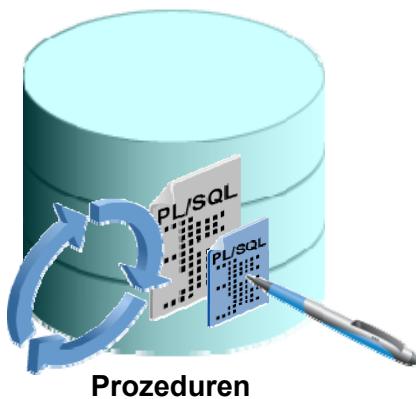
- Modularisierte Unterprogramme sowie Unterprogramme mit Schichten verwenden und Vorteile von Unterprogrammen bestimmen
- Mit Prozeduren arbeiten:
 - Prozeduren erstellen und aufrufen
 - Verfügbare Modi für die Parameterübergabe bestimmen
 - Formale und tatsächliche Parameter verwenden
 - Positionale, benannte oder gemischte Notation verwenden
- Exceptions in Prozeduren behandeln, Prozeduren entfernen und Prozedurinformationen anzeigen
- Reine PL/SQL-Typen binden

ORACLE®

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

Was sind Prozeduren?

- **Unterprogrammtyp, der eine Aktion durchführt**
- **Können in der Datenbank als Schemaobjekt gespeichert werden**
- **Fördern die Wiederverwendbarkeit und Verwaltbarkeit**



ORACLE

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

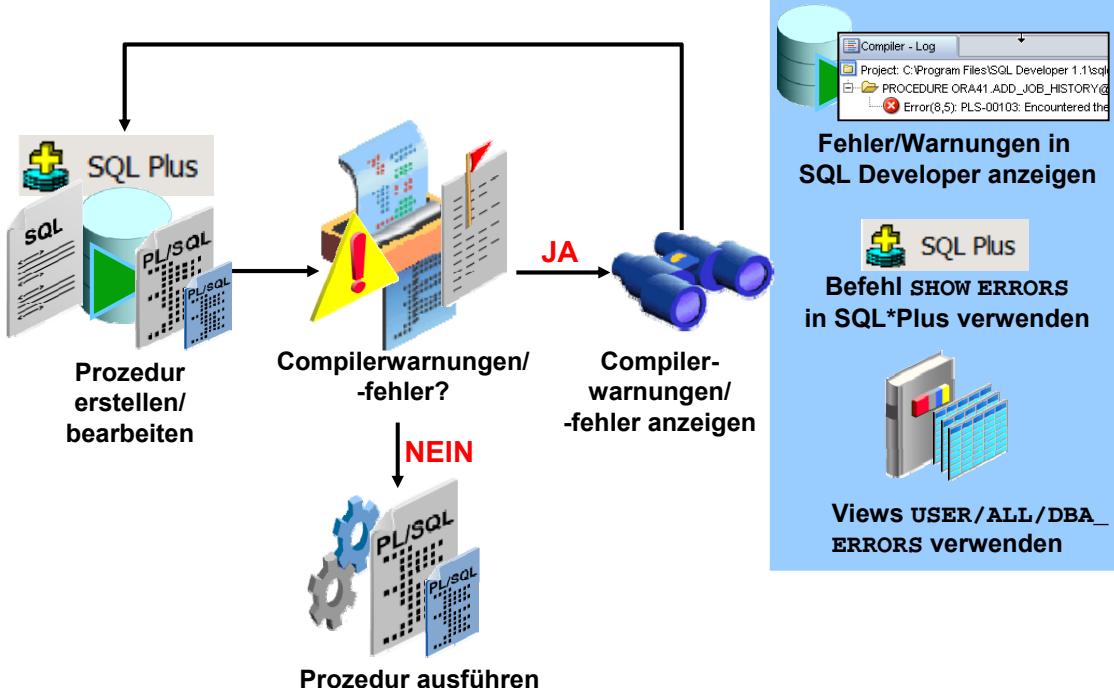
Prozeduren – Definition

Prozeduren sind benannte PL/SQL-Blöcke, die Parameter (manchmal als Argumente bezeichnet) annehmen können. Im Allgemeinen führen Sie mit Prozeduren Aktionen aus. Prozeduren bestehen aus einem Header, einem deklarativen Bereich, einem ausführbaren Bereich und einem optionalen Bereich zur Exception-Behandlung. Prozeduren werden über ihren Namen im ausführbaren Bereich eines anderen PL/SQL-Blockes aufgerufen.

Eine Prozedur wird kompiliert und als Schemaobjekt in der Datenbank gespeichert. Wenn Sie die Prozeduren mit Oracle Forms und Oracle Reports verwenden, können sie innerhalb der ausführbaren Dateien von Oracle Forms oder Oracle Reports kompiliert werden.

Prozeduren fördern die Wiederverwendbarkeit und Verwaltbarkeit. Wenn sie validiert sind, können sie in beliebig vielen Anwendungen verwendet werden. Ändern sich die Anforderungen, müssen nur die Prozeduren aktualisiert werden.

Prozeduren erstellen – Überblick



ORACLE®

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

Um eine Prozedur mit einem Tool wie SQL Developer zu entwickeln, gehen Sie wie folgt vor:

1. Erstellen Sie die Prozedur mithilfe des Object Navigator-Baumes oder des SQL Worksheet-Bereichs von SQL Developer.
2. Komplizieren Sie die Prozedur. Die Prozedur wird in der Datenbank erstellt und kompliziert. Die CREATE PROCEDURE-Anweisung erstellt und speichert den Quellcode und den komplizierten *m*-Code in der Datenbank. Um die Prozedur zu komplizieren, klicken Sie im Object Navigator-Baum mit der rechten Maustaste auf den Namen der Prozedur. Klicken Sie anschließend auf **Compile**.
3. Im Falle von Komplizierungsfehlern wird der *m*-Code nicht gespeichert. Dann müssen Sie den Quellcode korrigieren. Prozeduren mit Komplizierungsfehlern lassen sich nicht aufrufen. Sie können die Komplizierungsfehler in SQL Developer, SQL*Plus oder den entsprechenden Data Dictionary Views anzeigen, wie auf der Folie dargestellt.
4. Führen Sie nach erfolgreicher Komplizierung die Prozedur für die gewünschte Aktion aus. Sie können die Prozedur mit SQL Developer oder über den Befehl EXECUTE in SQL*Plus ausführen.

Hinweis: Überschreiben Sie bei Komplizierungsfehlern den vorhandenen Code mit der Anweisung CREATE OR REPLACE PROCEDURE, wenn Sie zuvor eine CREATE PROCEDURE-Anweisung verwendet haben. Löschen Sie andernfalls zuerst die Prozedur (mit DROP), und führen Sie dann die CREATE PROCEDURE-Anweisung aus.

Prozeduren mit der SQL-Anweisung CREATE OR REPLACE erstellen

- **Mit der Klausel CREATE eine Standalone-Prozedur erstellen, die in der Oracle-Datenbank gespeichert wird**
- **Vorhandene Prozedur mit der Option OR REPLACE überschreiben**

```
CREATE [OR REPLACE] PROCEDURE procedure_name
  [(parameter1 [mode] datatype1,
    parameter2 [mode] datatype2, ...)]
IS|AS
  [local_variable_declarations; ...]
BEGIN
  -- actions;
END [procedure_name];
```

PL/SQL-Block

ORACLE

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

Mit der SQL-Anweisung CREATE PROCEDURE können Sie Standalone-Prozeduren erstellen, die in einer Oracle-Datenbank gespeichert werden. Eine Prozedur entspricht einem kleinen Programm: Sie führt eine bestimmte Aktion aus. Sie geben den Namen der Prozedur, ihre Parameter, ihre lokalen Variablen und den Block BEGIN-END an, der den Code der Prozedur enthält und eventuelle Exceptions behandelt.

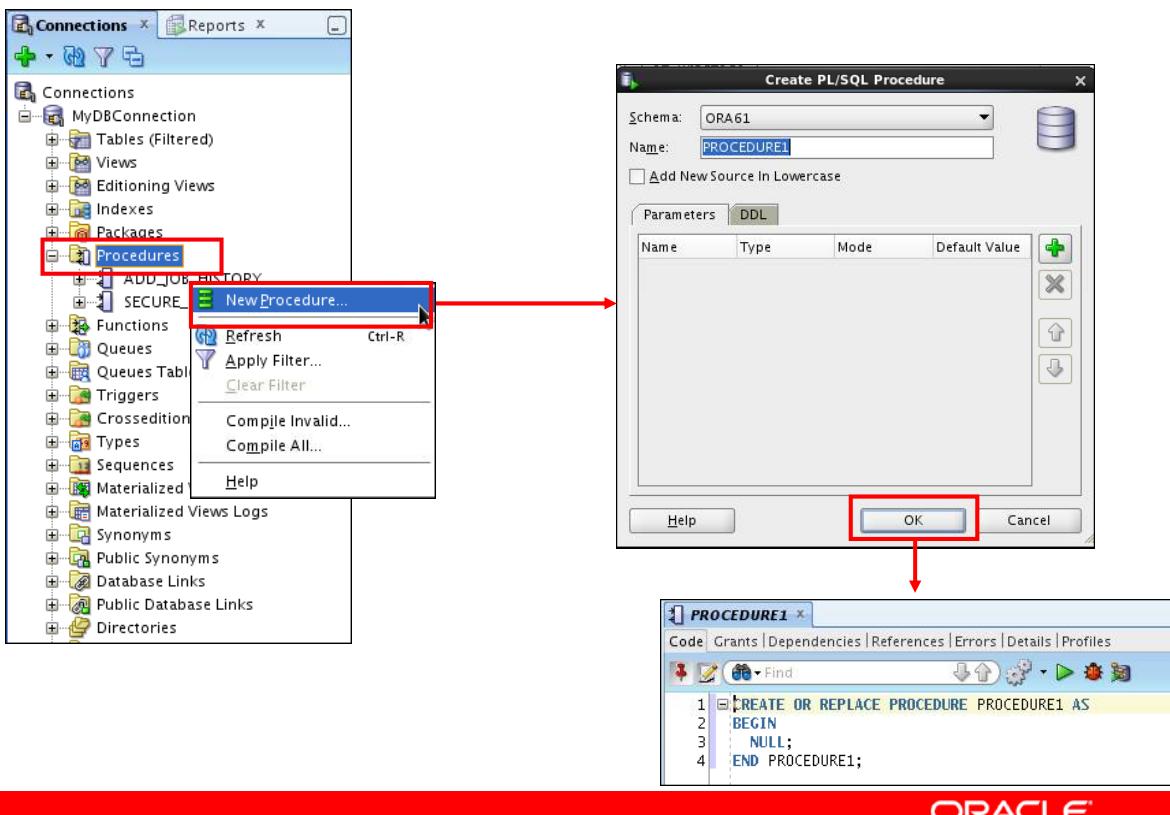
- PL/SQL-Blöcke beginnen mit BEGIN. Wahlweise geht eine Deklaration lokaler Variablen voraus. Sie enden entweder mit END oder END *procedure_name*.
- Die Option REPLACE gibt an, dass die (vorhandene) Prozedur gelöscht und durch die von der Anweisung erstellte neue Version ersetzt wird. Die Berechtigungen, die der Prozedur zugeordnet sind, werden durch die Option REPLACE nicht gelöscht.

Weitere syntaktische Elemente

- *parameter1* steht für den Namen eines Parameters.
- Die Option *mode* definiert die Verwendung eines Parameters: IN (Standard), OUT oder IN OUT.
- *datatype1* gibt den Datentyp des Parameters ohne Gesamtstelleanzahl an.

Hinweis: Parameter können als lokale Variablen angesehen werden. Austauschvariablen und Host- oder Bind-Variablen können in der Definition einer PL/SQL-Stored Procedure nicht referenziert werden. Die Option OR REPLACE erfordert keine Änderung der Objektsicherheit, sofern Sie Eigentümer des Objekts sind und über die Berechtigung CREATE [ANY] PROCEDURE verfügen.

Prozeduren mit SQL Developer erstellen



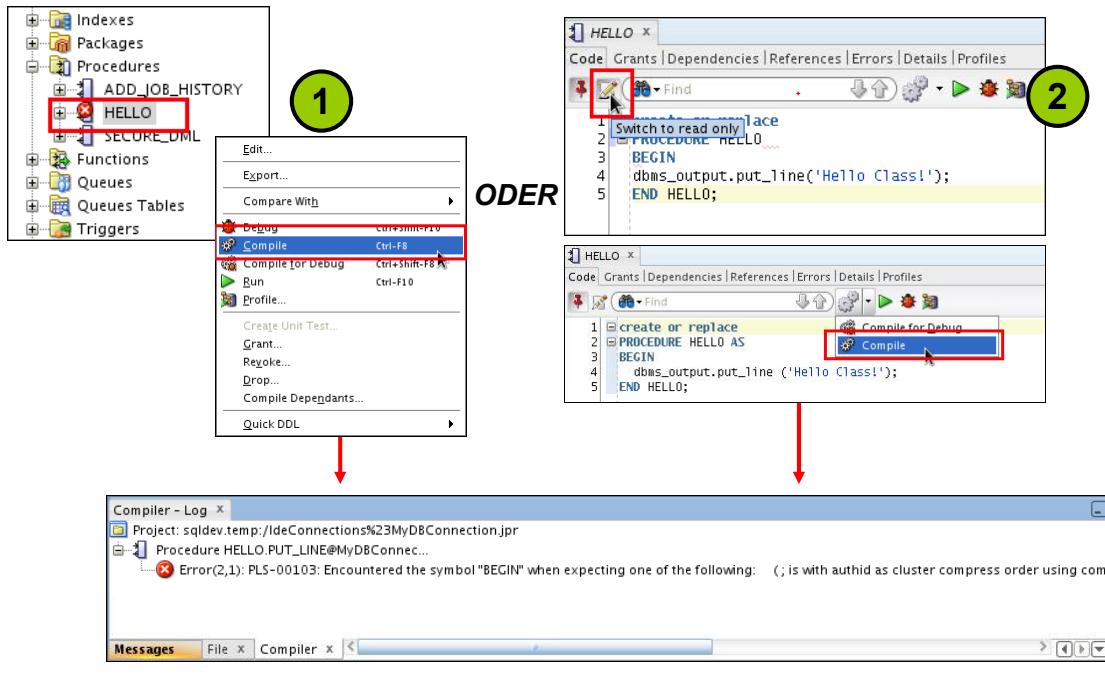
Copyright © 2013, Oracle und/oder verbundene Unternehmen. All rights reserved. Alle Rechte vorbehalten.

1. Klicken Sie in der Registerkarte **Connections** mit der rechten Maustaste auf den Knoten **Procedures**.
2. Wählen Sie im Kontextmenü die Option **New Procedure**. Das Dialogfeld **Create PL/SQL Procedure** wird angezeigt. Geben Sie die Informationen für die neue Prozedur an. Klicken Sie dann auf **OK**, um das Unterprogramm zu erstellen und im Editorfenster anzuzeigen, wo Sie die Details eingeben können.

Das Dialogfeld **Create PL/SQL Procedure** setzt sich wie folgt zusammen:

- **Schema:** Datenbankschema, in dem das PL/SQL-Unterprogramm zu erstellen ist
- **Name:** Name des Unterprogramms, der innerhalb eines Schemas eindeutig sein muss
- **Add New Source in Lowercase:** Ist dieses Kontrollkästchen aktiviert, wird neuer Text in Kleinbuchstaben angezeigt, auch wenn Sie Groß-/Kleinschreibung verwenden. Diese Option wirkt sich lediglich auf die Darstellung des Codes aus, da PL/SQL bei der Ausführung die Groß-/Kleinschreibung nicht beachtet.
- **Registerkarte "Parameters":** Um einen Parameter hinzuzufügen, klicken Sie auf das Symbol **Add** (+). Geben Sie für jeden Parameter, der in der Prozedur erstellt werden soll, den Parameternamen, den Datentyp, den Modus und optional den Standardwert an. Verwenden Sie das Symbol **Remove** (X) und die Pfeilsymbole, um einen Parameter zu löschen oder in der Liste nach oben oder unten zu verschieben.
- **Registerkarte "DDL":** Diese Registerkarte enthält eine schreibgeschützte Anzeige einer SQL-Anweisung, die die tatsächliche Definition des Unterprogramms widerspiegelt.

In SQL Developer Prozeduren kompilieren und Kompilierungsfehler anzeigen



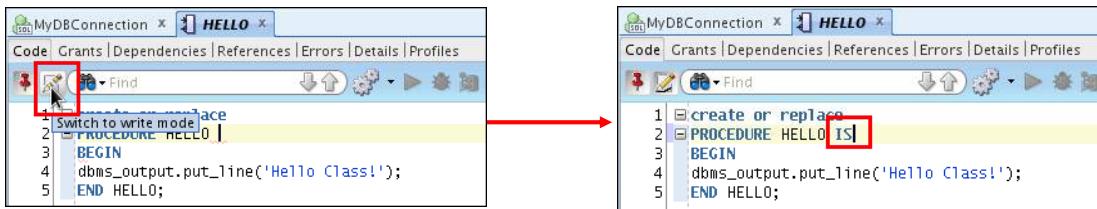
ORACLE

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

Prozeduren können Sie mit einer der folgenden zwei Methoden kompilieren:

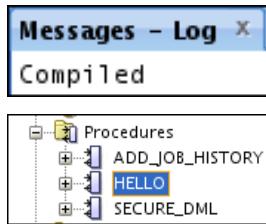
- Navigieren Sie im Object Navigator-Baum zum Knoten **Procedures**. Klicken Sie mit der rechten Maustaste auf den Namen der Prozedur, und wählen Sie dann im Kontextmenü die Option **Compile**. Um eventuelle Kompilierungsfehler anzuzeigen, wählen Sie in der Registerkarte **Compiler – Log** die Unterregisterkarte **Messages**.
- Bearbeiten Sie die Prozedur mit dem Symbol **Edit** in der Codesymbolleiste der Prozedur. Nehmen Sie die nötigen Änderungen vor, und klicken Sie dann in der Codesymbolleiste auf das Symbol **Compile**. Um eventuelle Kompilierungsfehler anzuzeigen, wählen Sie in der Registerkarte **Compiler – Log** die Unterregisterkarte **Messages**.

Kompilierungsfehler in SQL Developer korrigieren

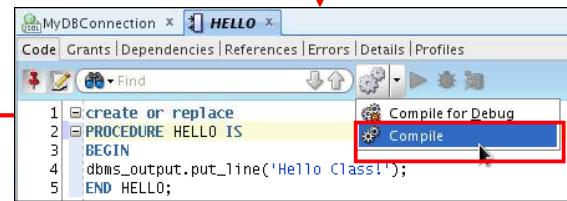


1. Prozedur bearbeiten

2. Fehler korrigieren
(Schlüsselwort **IS** hinzufügen)



4. Rekompilierung erfolgreich



3. Prozedur rekompilieren

ORACLE

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

1. Bearbeiten Sie die Prozedur mit dem Symbol **Switch to write mode** in der Codesymbolleiste der Prozedur.
2. Nehmen Sie die nötigen Korrekturen vor.
3. Klicken Sie in der Codesymbolleiste auf das Symbol **Compile**.
4. Um eventuelle Kompilierungsfehler anzuzeigen, wählen Sie in der Registerkarte **Compiler - Log** die Unterregisterkarte **Messages**. War die Kompilierung der Prozedur erfolgreich, wird außerdem im Object Navigator-Baum das rote X vor dem Namen der Prozedur entfernt.

In diesem Kurs verwendete PL/SQL-Strukturen – Benennungskonventionen

PL/SQL-Struktur	Konvention	Beispiel
Variable	<code>v_variable_name</code>	<code>v_rate</code>
Konstante	<code>c_constant_name</code>	<code>c_rate</code>
Unterprogrammparameter	<code>p_parameter_name</code>	<code>p_id</code>
Bind- oder Hostvariable	<code>b_bind_name</code>	<code>b_salary</code>
Cursor	<code>cur_cursor_name</code>	<code>cur_emp</code>
Record	<code>rec_record_name</code>	<code>rec_emp</code>
Typ	<code>type_name_type</code>	<code>ename_table_type</code>
Exception	<code>e_exception_name</code>	<code>e_products_invalid</code>
Datei-Handle	<code>f_file_handle_name</code>	<code>f_file</code>

ORACLE

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

In der Tabelle auf der Folie sind einige Beispiele für Benennungskonventionen von PL/SQL-Strukturen aufgeführt, die in diesem Kurs verwendet werden.

Was sind Parameter und Parametermodi?

- **Werden nach dem Namen des Unterprogramms im PL/SQL-Header deklariert**
- **Übergeben oder kommunizieren Daten zwischen der aufrufenden Umgebung und dem Unterprogramm**
- **Werden wie lokale Variablen verwendet, hängen jedoch vom Modus der Parameterübergabe ab:**
 - Der Parametermodus **IN** (Standard) liefert Werte für die Verarbeitung durch ein Unterprogramm.
 - Der Parametermodus **OUT** gibt dem Aufrufer einen Wert zurück.
 - Der Parametermodus **IN OUT** stellt einen Eingabewert bereit, der als modifizierter Wert zurückgegeben werden kann (**Ausgabe**).

ORACLE

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

Was sind Parameter?

Mit Parametern werden Datenwerte zwischen der aufrufenden Umgebung und der Prozedur (oder dem Unterprogramm) übertragen. Parameter werden im Unterprogrammheader nach dem Namen und vor dem deklarativen Bereich für lokale Variablen deklariert.

Sie unterliegen einem der drei Modi für die Parameterübergabe: **IN**, **OUT** oder **IN OUT**.

- Der Parameter **IN** übergibt einen konstanten Wert aus der aufrufenden Umgebung an die Prozedur.
- Der Parameter **OUT** übergibt einen Wert aus der Prozedur an die aufrufende Umgebung.
- Der Parameter **IN OUT** übergibt einen Wert aus der aufrufenden Umgebung an die Prozedur und einen möglicherweise abweichenden Wert aus der Prozedur zurück an die aufrufende Umgebung, wobei derselbe Parameter verwendet wird.

Parameter sind eine spezielle Form der lokalen Variablen. Ihre Eingabewerte werden durch die aufrufende Umgebung initialisiert, wenn das Unterprogramm aufgerufen wird. Ihre Ausgabewerte werden an die aufrufende Umgebung zurückgegeben, wenn das Unterprogramm die Kontrolle an den Aufrufer zurückgibt.

Formale und tatsächliche Parameter

- **Formale Parameter:** Lokale Variablen, die in der Parameterliste einer Unterprogrammspezifikation deklariert werden
- **Tatsächliche Parameter (oder Argumente):** Literalwerte, Variablen oder Ausdrücke, die in der Parameterliste des aufrufenden Unterprogramms verwendet werden

```
-- Procedure definition, Formal parameters
CREATE PROCEDURE raise_sal(p_id NUMBER, p_sal NUMBER) IS
BEGIN
  . . .
END raise_sal;

-- Procedure calling, Actual parameters (arguments)
v_emp_id := 100;
raise_sal(v_emp_id, 2000)
```

ORACLE

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

Formale Parameter sind lokale Variablen, die in der Parameterliste einer Unterprogrammspezifikation deklariert werden. Im ersten Beispiel stehen die Variablenbezeichner **p_id** und **p_sal** in der Prozedur **raise_sal** für die formalen Parameter.

Die tatsächlichen Parameter können Literalwerte, Variablen und Ausdrücke sein, die in der Parameterliste eines aufrufenden Unterprogramms bereitgestellt werden. Im zweiten Beispiel wird **raise_sal** aufgerufen. Die Variable **v_emp_id** stellt den tatsächlichen Parameterwert für den formalen Parameter **p_id** bereit, während für **p_sal** der tatsächliche Parameterwert 2000 bereitgestellt wird.

Tatsächliche Parameter:

- werden während des Unterprogrammaufrufs formalen Parametern zugeordnet
- können auch Ausdrücke sein. Beispiel:
`raise_sal(v_emp_id, raise+100);`

Formale und tatsächliche Parameter müssen kompatible Datentypen aufweisen. Falls erforderlich, konvertiert PL/SQL den Datentyp des tatsächlichen Parameterwertes in den Datentyp des formalen Parameters, bevor ein Wert zugewiesen wird.

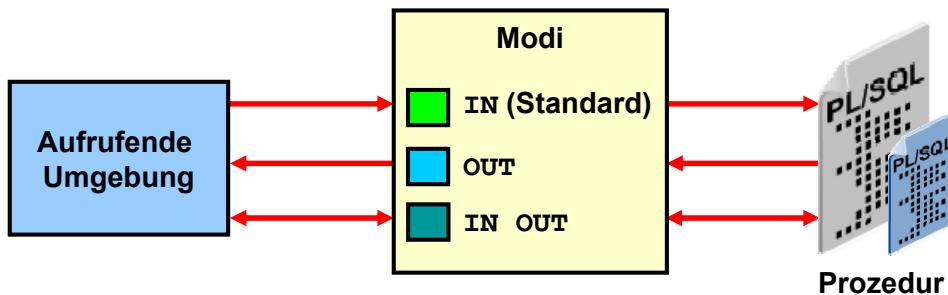
Hinweis: Tatsächliche Parameter werden auch *tatsächliche Argumente* genannt.

Prozedurale Parametermodi

- **Parametermodi werden in der Deklaration formaler Parameter nach dem Parameternamen und vor dem Datentyp angegeben.**
- **Der Modus IN ist der Standardmodus, falls kein Modus angegeben wird.**

```
CREATE PROCEDURE proc_name(param_name [mode] datatype)
...

```



ORACLE

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

Wenn Sie eine Prozedur erstellen, definiert der formale Parameter einen Variablenamen, dessen Wert im ausführbaren Bereich des PL/SQL-Blockes verwendet wird. Mit dem tatsächlichen Parameter werden beim Aufrufen der Prozedur Eingabewerte bereitgestellt oder Ausgabeergebnisse empfangen.

Der Parametermodus **IN** ist der Standardübergabemodus. Wenn also in einer Parameterdeklaration kein Modus angegeben ist, wird vom Parameter **IN** ausgegangen. Die Parametermodi **OUT** und **IN OUT** müssen in den Parameterdeklarationen explizit angegeben werden.

Die Angabe des Parameters **datatype** erfolgt ohne Größenspezifikation. Er wird folgendermaßen angegeben:

- Als expliziter Datentyp
- Mit der Definition **%TYPE**
- Mit der Definition **%ROWTYPE**

Hinweis: Sie können einen oder mehrere formale Parameter deklarieren, die jeweils durch ein Komma getrennt sein müssen.

Parametermodi – Vergleich

IN	OUT	IN OUT
Standardmodus	Muss angegeben werden	Muss angegeben werden
Wert wird an Unterprogramm übergeben.	Wert wird an aufrufende Umgebung zurückgegeben.	Wert wird an Unterprogramm übergeben und an aufrufende Umgebung zurückgegeben.
Formaler Parameter fungiert als Konstante.	Nicht initialisierte Variable	Initialisierte Variable
Tatsächlicher Parameter kann Literal, Ausdruck, Konstante oder initialisierte Variable sein.	Muss eine Variable sein	Muss eine Variable sein
Standardwert kann zugewiesen werden.	Standardwert kann nicht zugewiesen werden.	Standardwert kann nicht zugewiesen werden.

ORACLE

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

Der Parametermodus `IN` ist der Standardmodus, wenn in der Deklaration kein Modus angegeben wird. Die Parametermodi `OUT` und `IN OUT` müssen in den Parameterdeklarationen explizit angegeben werden.

Einem formalen Parameter im Modus `IN` kann kein Wert zugewiesen werden. Er lässt sich im Body der Prozedur nicht ändern. Standardmäßig wird der Parameter `IN` per Referenz übergeben. Dem Parameter `IN` kann in der Deklaration der formalen Parameter ein Standardwert zugewiesen werden. In diesem Fall muss der Aufrufer keinen Wert für den Parameter bereitstellen, wenn der Standardwert gültig ist.

Dem Parameter `OUT` oder `IN OUT` muss ein Wert zugewiesen werden, bevor er an die aufrufende Umgebung zurückgegeben wird. Den Parametern `OUT` und `IN OUT` können keine Standardwerte zugewiesen werden. Um die Performance dieser Parameter zu verbessern, können Sie die Übergabe mit dem Compiler-Hint `NOCOPY` per Referenz anfordern.

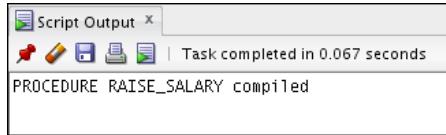
Hinweis: Die Verwendung von `NOCOPY` wird im weiteren Verlauf dieses Kurses erörtert.

Parametermodus IN – Beispiel

```

CREATE OR REPLACE PROCEDURE raise_salary
  (p_id          IN employees.employee_id%TYPE,
   p_percent     IN NUMBER)
IS
BEGIN
  UPDATE employees
  SET    salary = salary * (1 + p_percent/100)
  WHERE employee_id = p_id;
END raise_salary;
/

```



```
EXECUTE raise_salary(176, 10)
```

ORACLE

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

IN-Parameter – Beispiel

Das Beispiel auf der Folie zeigt eine Prozedur mit zwei IN-Parametern. Wenn Sie das erste Beispiel auf der Folie ausführen, wird in der Datenbank die Prozedur `raise_salary` erstellt. Im zweiten Beispiel auf der Folie wird `raise_salary` aufgerufen und 176 als erster Parameterwert für die Personalnummer sowie eine Gehaltserhöhung um 10 Prozent als zweiter Parameterwert bereitgestellt.

Um eine Prozedur über das SQL Worksheet von SQL Developer oder mithilfe von SQL*Plus aufzurufen, verwenden Sie den Befehl `EXECUTE`, der im zweiten Codebeispiel auf der Folie zu sehen ist.

Um eine Prozedur aus einer anderen Prozedur aufzurufen, verwenden Sie einen direkten Aufruf innerhalb eines ausführbaren Bereichs des aufrufenden Blockes. Geben Sie den Namen der Prozedur und die tatsächlichen Parameter an der Position ein, an der die neue Prozedur aufgerufen wird. Beispiel:

```

...
BEGIN
  raise_salary (176, 10);
END;

```

Hinweis: IN-Parameter werden als schreibgeschützte Werte aus der aufrufenden Umgebung an die Prozedur übergeben. Beim Versuch, den Wert eines IN-Parameters zu ändern, wird ein Kompilierungszeitfehler angezeigt.

Parametermodus OUT – Beispiel

```
CREATE OR REPLACE PROCEDURE query_emp
  (p_id      IN employees.employee_id%TYPE,
   p_name    OUT employees.last_name%TYPE,
   p_salary  OUT employees.salary%TYPE) IS
BEGIN
  SELECT last_name, salary INTO p_name, p_salary
  FROM   employees
  WHERE  employee_id = p_id;
END query_emp;
/
```

```
SET SERVEROUTPUT ON
DECLARE
  v_emp_name employees.last_name%TYPE;
  v_emp_sal  employees.salary%TYPE;
BEGIN
  query_emp(171, v_emp_name, v_emp_sal);
  DBMS_OUTPUT.PUT_LINE(v_emp_name || ' earns ' ||
    to_char(v_emp_sal, '$999,999.00'));
END;
/
```

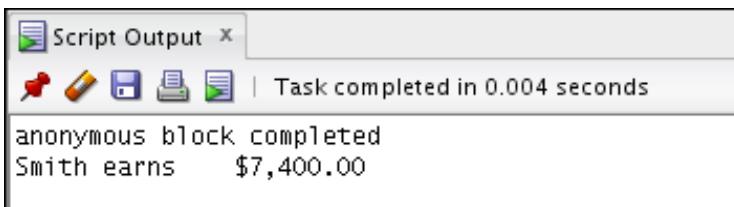
ORACLE

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

OUT-Parameter – Beispiel

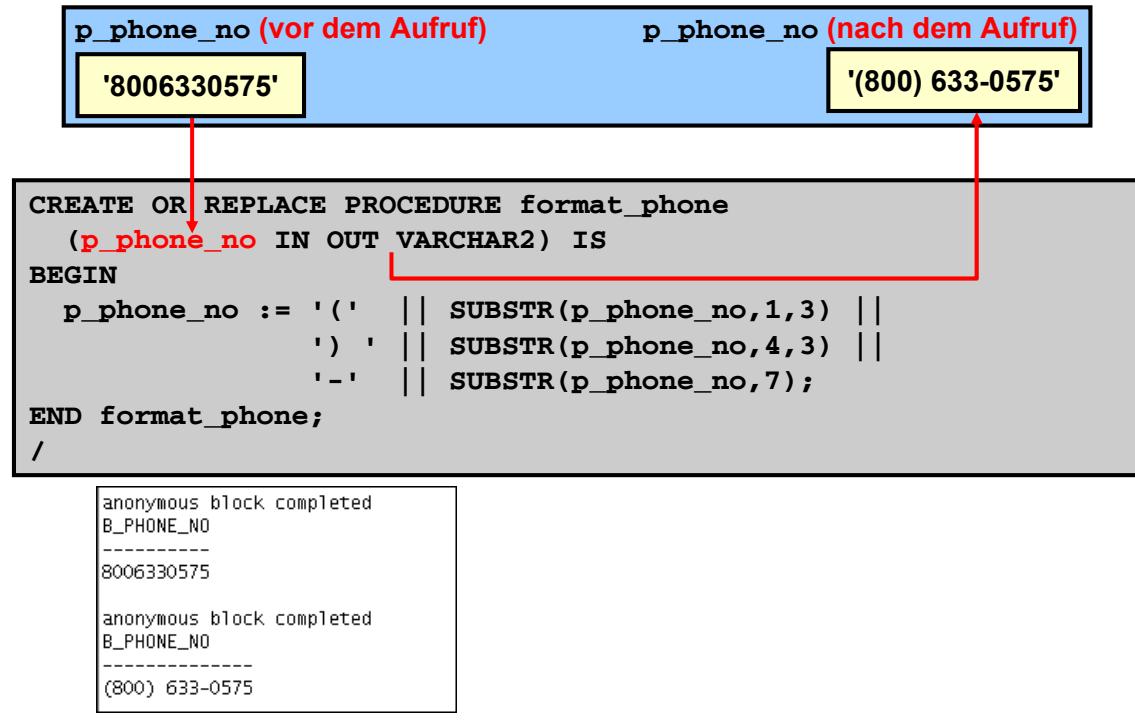
Im Beispiel auf der Folie wird eine Prozedur mit OUT-Parametern erstellt, um Informationen über einen Mitarbeiter abzurufen. Die Prozedur nimmt den Wert 171 als Personalnummer an und ruft den Namen und das Gehalt des Mitarbeiters mit der Nummer 171 in die zwei OUT-Parameter ab. Die Prozedur `query_emp` hat drei formale Parameter. Zwei von ihnen sind OUT-Parameter, die Werte an die aufrufende Umgebung zurückgeben, wie im zweiten Codefeld auf der Folie gezeigt. Die Prozedur nimmt über den Parameter `p_id` einen Wert für die Personalnummer an. Die Variablen `v_emp_name` und `v_emp_salary` werden mit den Informationen gefüllt, die von der Abfrage in ihre entsprechenden OUT-Parameter abgerufen werden. Wenn der Code aus dem zweiten Codebeispiel auf der Folie ausgeführt wird, erhalten Sie folgendes Ergebnis: `v_emp_name` nimmt den Wert **Smith** auf und `v_emp_salary` den Wert **7400**.

Hinweis: Stellen Sie sicher, dass der Datentyp für die tatsächlichen Parametervariablen, mit denen Werte von OUT-Parametern abgerufen werden, groß genug ist, um die zurückgegebenen Datenwerte aufzunehmen.



Parametermodus IN OUT – Beispiel

Aufrufende Umgebung



ORACLE

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

IN OUT-Parameter – Beispiel

Mit einem IN OUT-Parameter können Sie einen Wert an eine Prozedur übergeben, der aktualisiert werden kann. Der von der aufrufenden Umgebung bereitgestellte tatsächliche Parameterwert kann entweder den unveränderten Originalwert oder einen neuen Wert zurückgeben, der in der Prozedur festgelegt ist.

Hinweis: Ein IN OUT-Parameter fungiert als initialisierte Variable.

Das Beispiel auf der Folie erstellt eine Prozedur mit einem IN OUT-Parameter, der eine aus Ziffern bestehende Zeichenfolge für eine Telefonnummer annimmt. Die Prozedur gibt die Telefonnummer formatiert zurück. Die ersten drei Zeichen sind von einer Klammer umschlossen, nach der sechsten Ziffer ist ein Bindestrich eingefügt. Beispiel: Die Telefonnummer 8006330575 wird als (800) 633-0575 zurückgegeben.

Der folgende Code stellt mit der Hostvariablen b_phone_no von SQL*Plus den Eingabewert bereit, der an die Prozedur FORMAT_PHONE übergeben wird. Die Prozedur wird ausgeführt und gibt eine aktualisierte Zeichenfolge in der Hostvariablen b_phone_no zurück. Die Ausgabe des folgenden Codes ist auf der Folie oben zu sehen:

```

VARIABLE b_phone_no VARCHAR2(15)
EXECUTE :b_phone_no := '8006330575'
PRINT b_phone_no
EXECUTE format_phone (:b_phone_no)
PRINT b_phone_no

```

OUT-Parameter mithilfe der untergeordneten Routine **DBMS_OUTPUT.PUT_LINE anzeigen**

PL/SQL-Variablen verwenden, die mit Aufrufen der Prozedur DBMS_OUTPUT.PUT_LINE ausgegeben werden:

```
SET SERVEROUTPUT ON

DECLARE
    v_emp_name employees.last_name%TYPE;
    v_emp_sal   employees.salary%TYPE;
BEGIN
    query_emp(171, v_emp_name, v_emp_sal);
    DBMS_OUTPUT.PUT_LINE('Name: ' || v_emp_name);
    DBMS_OUTPUT.PUT_LINE('Salary: ' || v_emp_sal);
END;
```

```
anonymous block completed
Name: Smith
Salary: 7400
```

ORACLE

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

OUT-Parameter mithilfe der untergeordneten Routine DBMS_OUTPUT anzeigen

Das Beispiel auf der Folie veranschaulicht, wie Sie die von den OUT-Parametern zurückgegebenen Werte in SQL*Plus oder im SQL Developer-Worksheet anzeigen.

Die OUT-Parameterwerte können mit PL/SQL-Variablen in einem anonymen Block abgerufen werden. Die Prozedur DBMS_OUTPUT.PUT_LINE wird aufgerufen, um die in den PL/SQL-Variablen enthaltenen Werte auszugeben. SET SERVEROUTPUT muss auf ON festgelegt sein.

OUT-Parameter mithilfe von SQL*Plus-Hostvariablen anzeigen

- 1. SQL*Plus-Hostvariablen verwenden**
- 2. QUERY_EMP mithilfe von Hostvariablen ausführen**
- 3. Hostvariablen ausgeben**

```
VARIABLE b_name    VARCHAR2(25)
VARIABLE b_sal     NUMBER
EXECUTE query_emp(171, :b_name, :b_sal)
PRINT b_name b_sal
```

The screenshot shows the 'Script Output' window from Oracle SQL*Plus. It displays the output of an anonymous block. The output is as follows:

```
anonymous block completed
B_NAME
-----
Smith

B_SAL
-----
7400
```

ORACLE

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

Das Beispiel auf der Folie demonstriert die Verwendung von SQL*Plus-Hostvariablen, die mit dem Befehl VARIABLE erstellt werden. Die SQL*Plus-Variablen befinden sich außerhalb des PL/SQL-Blockes und werden als Host- oder Bind-Variablen bezeichnet. Um Hostvariablen aus einem PL/SQL-Block zu referenzieren, müssen Sie Ihrem Namen einen Doppelpunkt (:) voranstellen. Zur Anzeige der in den Hostvariablen gespeicherten Werte verwenden Sie den SQL*Plus-Befehl PRINT, gefolgt vom Namen der SQL*Plus-Variablen (ohne Doppelpunkt, da es sich nicht um einen PL/SQL-Befehl oder -Kontext handelt).

Hinweis: Details zum Befehl VARIABLE finden Sie im Dokument SQL*Plus Command Reference.

Verfügbare Notationen für die Übergabe von tatsächlichen Parametern

- Wenn Sie ein Unterprogramm aufrufen, können Sie die tatsächlichen Parameter mit den folgenden Notationen erstellen:
 - Positionale Notation: Tatsächliche Parameter werden in der gleichen Reihenfolge aufgelistet wie formale Parameter.
 - Benannte Notation: Tatsächliche Parameter werden in zufälliger Reihenfolge aufgelistet. Mit dem Zuweisungsoperator (`=>`) wird ein benannter formaler Parameter seinem tatsächlichen Parameter zugeordnet.
 - Gemischte Notation: Einige tatsächliche Parameter werden positional aufgelistet, andere benannt.
- Vor Oracle Database 11g wurde nur die positionale Notation in Aufrufen aus SQL unterstützt.
- Ab Oracle Database 11g kann die benannte und gemischte Notation zur Angabe von Argumenten in Aufrufen von untergeordneten PL/SQL-Routinen aus SQL-Anweisungen verwendet werden.

ORACLE

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

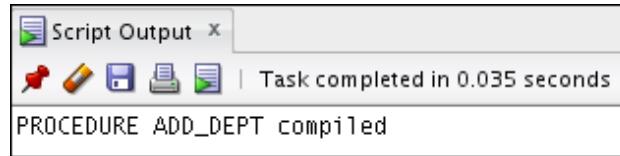
Parameter übergeben – Syntax

Wenn Sie ein Unterprogramm aufrufen, können Sie die tatsächlichen Parameter mit den folgenden Notationen erstellen:

- **Positionale Notation:** Sie listen die tatsächlichen Parameterwerte in derselben Reihenfolge auf, in der die formalen Parameter deklariert werden. Diese Notation ist kompakt. Wenn Sie jedoch die Parameter (insbesondere Literale) in der falschen Reihenfolge angeben, ist der Fehler unter Umständen schwer zu ermitteln. Sie müssen den Code ändern, wenn sich die Parameterliste der Prozedur ändert.
- **Benannte Notation:** Sie listen die tatsächlichen Werte in willkürlicher Reihenfolge auf. Mithilfe des Zuweisungsoperators ordnen Sie die einzelnen tatsächlichen Parameter anhand der Namen ihren formalen Parametern zu. Der PL/SQL-**Zuweisungsoperator** ist ein Gleichheitszeichen, dem direkt und ohne Leerzeichen ein Größer-als-Zeichen folgt: `=>`. Die Reihenfolge der Parameter hat keine Bedeutung. Diese Notation ist aufwändiger, der Code ist jedoch einfacher zu lesen und zu verwalten. Sie können manchmal Codeänderungen vermeiden, wenn sich die Parameterliste der Prozedur ändert, beispielsweise wenn die Parameter neu angeordnet werden oder ein neuer optionaler Parameter hinzukommt.
- **Gemischte Notation:** Sie listen die ersten Parameterwerte nach ihrer Position auf, die übrigen mit der speziellen Syntax der benannten Methode. Mit dieser Notation können Sie Prozeduren aufrufen, die einige obligatorische Parameter gefolgt von einigen optionalen Parametern aufweisen.

Tatsächliche Parameter übergeben – Prozedur add_dept erstellen

```
CREATE OR REPLACE PROCEDURE add_dept(
    p_name IN departments.department_name%TYPE,
    p_loc  IN departments.location_id%TYPE) IS
BEGIN
    INSERT INTO departments(department_id,
                           department_name, location_id)
    VALUES (departments_seq.NEXTVAL, p_name , p_loc );
END add_dept;
/
```



ORACLE

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

Parameter übergeben – Beispiele

Im Beispiel auf der Folie deklariert die Prozedur add_dept zwei formale IN-Parameter: p_name und p_loc. Die Werte dieser Parameter werden in der Anweisung INSERT verwendet, um die Spalten department_name und location_id festzulegen.

Tatsächliche Parameter übergeben – Beispiele

```
-- Passing parameters using the positional notation.
EXECUTE add_dept ('TRAINING', 2500)
```

Script Output X	
Task completed in 0.002 seconds	
PROCEDURE ADD_DEPT compiled anonymous block completed DEPARTMENT_ID DEPARTMENT_NAME	MANAGER_ID LOCATION_ID
----- 280 TRAINING	2500

```
-- Passing parameters using the named notation.
EXECUTE add_dept (p_loc=>2400, p_name=>'EDUCATION')
```

Script Output X	
Task completed in 0.002 seconds	
anonymous block completed DEPARTMENT_ID DEPARTMENT_NAME	MANAGER_ID LOCATION_ID
----- 290 EDUCATION	2400

ORACLE

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

Wie Sie tatsächliche Parameter anhand der Position übergeben, wird im ersten Codebeispiel auf der Folie im ersten Ausführungsaufruf von `add_dept` gezeigt. Der erste tatsächliche Parameter stellt den Wert `TRAINING` für den formalen Parameter `name` bereit. Der zweite tatsächliche Parameterwert `2500` wird dem formalen Parameter `loc` anhand seiner Position zugeordnet.

Die Parameterübergabe mithilfe der benannten Notation wird im zweiten Codebeispiel auf der Folie gezeigt. Der tatsächliche Parameter `loc` wird als zweiter formaler Parameter deklariert und anhand des Namens im Aufruf referenziert, wo er mit dem tatsächlichen Wert `2400` verknüpft wird. Der Namensparameter wird mit dem Wert `EDUCATION` verknüpft. Die Reihenfolge der tatsächlichen Parameter ist unerheblich, wenn alle Parameterwerte angegeben werden.

Hinweis: Sie müssen für alle Parameter einen Wert bereitstellen, es sei denn, der formale Parameter ist einem Standardwert zugeordnet. Im Folgenden wird erörtert, wie Sie Standardwerte für formale Parameter angeben.

Option **DEFAULT** für Parameter

- **Definiert Standardwerte für Parameter**
- **Bietet Flexibilität durch Kombination der positionalen und benannten Syntax zur Parameterübergabe**

```
CREATE OR REPLACE PROCEDURE add_dept(
    p_name departments.department_name%TYPE := 'Unknown',
    p_loc   departments.location_id%TYPE DEFAULT 1700)
IS
BEGIN
    INSERT INTO departments (department_id,
                           department_name, location_id)
    VALUES (departments_seq.NEXTVAL, p_name, p_loc);
END add_dept;
```

```
EXECUTE add_dept
EXECUTE add_dept ('ADVERTISING', p_loc => 1200)
EXECUTE add_dept (p_loc => 1200)
```

ORACLE

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

Ein Standardwert lässt sich einem **IN**-Parameter wie folgt zuweisen:

- Mit dem Zuweisungsoperator (**:=**), wie für den Parameter **name** auf der Folie gezeigt
- Mit der Option **DEFAULT**, wie für den Parameter **p_loc** auf der Folie gezeigt

Wenn den formalen Parametern Standardwerte zugeordnet werden, können Sie die Prozedur aufrufen, ohne einen tatsächlichen Parameterwert für den Parameter bereitzustellen. Sie können einem Unterprogramm eine unterschiedliche Anzahl tatsächlicher Parameter übergeben, indem Sie die Standardwerte je nach Erfordernis annehmen oder außer Kraft setzen. Es wird empfohlen, die Parameter zunächst ohne Standardwerte zu deklarieren. Anschließend können Sie formale Parameter mit Standardwerten hinzufügen, ohne alle Aufrufe der Prozedur ändern zu müssen.

Hinweis: Den Parametern **OUT** und **IN OUT** können keine Standardwerte zugeordnet werden.

Das zweite Codefeld auf der Folie zeigt drei Möglichkeiten zum Aufrufen der Prozedur **add_dept**:

- Im ersten Beispiel werden die Standardwerte für die einzelnen Parameter zugewiesen.
- Im zweiten Beispiel wird eine Kombination aus positionaler und benannter Notation zur Zuweisung von Werten gezeigt. In diesem Fall wird die benannte Notation als Beispiel vorgestellt.
- Im letzten Beispiel werden der Standardwert für den Parameter **name** (**Unknown**) und der bereitgestellte Wert für den Parameter **p_loc** verwendet.

Im Folgenden ist das Ergebnis des zweiten Codebeispiels aus der vorherigen Folie aufgeführt:

DEPARTMENT_ID	DEPARTMENT_NAME	MANAGER_ID	LOCATION_ID
280	TRAINING	2500	
290	EDUCATION	2400	
300	Unknown	1700	
310	ADVERTISING	1200	
320	Unknown	1200	
10	Administration	200	1700
20	Marketing	201	1800
30	Purchasing	114	1700
40	Human Resources	203	2400
50	Shipping	121	1500
60	IT	103	1400
70	Public Relations	204	2700
80	Sales	145	2500
90	Executive	100	1700
100	Finance	108	1700
110	Accounting	205	1700
120	Treasury		1700
130	Corporate Tax		1700
140	Control And Credit		1700
150	Shareholder Services		1700
160	Benefits		1700
170	Manufacturing		1700
180	Construction		1700
190	Contracting		1700
200	Operations		1700
210	IT Support		1700
220	NOC		1700
230	IT Helpdesk		1700
240	Government Sales		1700
250	Retail Sales		1700
260	Recruiting		1700
270	Payroll		1700

32 rows selected

In der Regel können Sie mit der benannten Notation die Standardwerte der formalen Parameter außer Kraft setzen. Sie müssen jedoch einen tatsächlichen Parameter übergeben, wenn für einen formalen Parameter kein Standardwert bereitgestellt wird.

Hinweis: Alle positionalen Parameter müssen in einem Unterprogrammaufruf vor den benannten Parametern stehen. Andernfalls erhalten Sie eine Fehlermeldung wie im folgenden Beispiel:

```
EXECUTE add_dept(p_name=>'new dept', 'new location')
```

```
Error starting at line 1 in command:
EXECUTE add_dept(p_name=>'new dept', 'new location')
Error report:
ORA-06550: Line 1, column 36:
PLS-00312: a positional parameter association may not follow a named association
ORA-06550: Line 1, column 7:
PL/SQL: Statement ignored
06550. 00000 - "line %s, column %s:\n%s"
*Cause: Usually a PL/SQL compilation error.
*Action:
```

Prozeduren aufrufen

- Sie können Prozeduren mit anonymen Blöcken, anderen Prozeduren oder Packages aufrufen.
- Sie müssen der Eigentümer der Prozedur sein oder die Berechtigung `EXECUTE` besitzen.

```
CREATE OR REPLACE PROCEDURE process_employees
IS
  CURSOR cur_emp_cursor IS
    SELECT employee_id
      FROM employees;
BEGIN
  FOR emp_rec IN cur_emp_cursor
  LOOP
    raise_salary(emp_rec.employee_id, 10);
  END LOOP;
  COMMIT;
END process_employees;
/
```

PROCEDURE PROCESS_EMPLOYEES compiled

ORACLE

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

Sie können Prozeduren aufrufen mit:

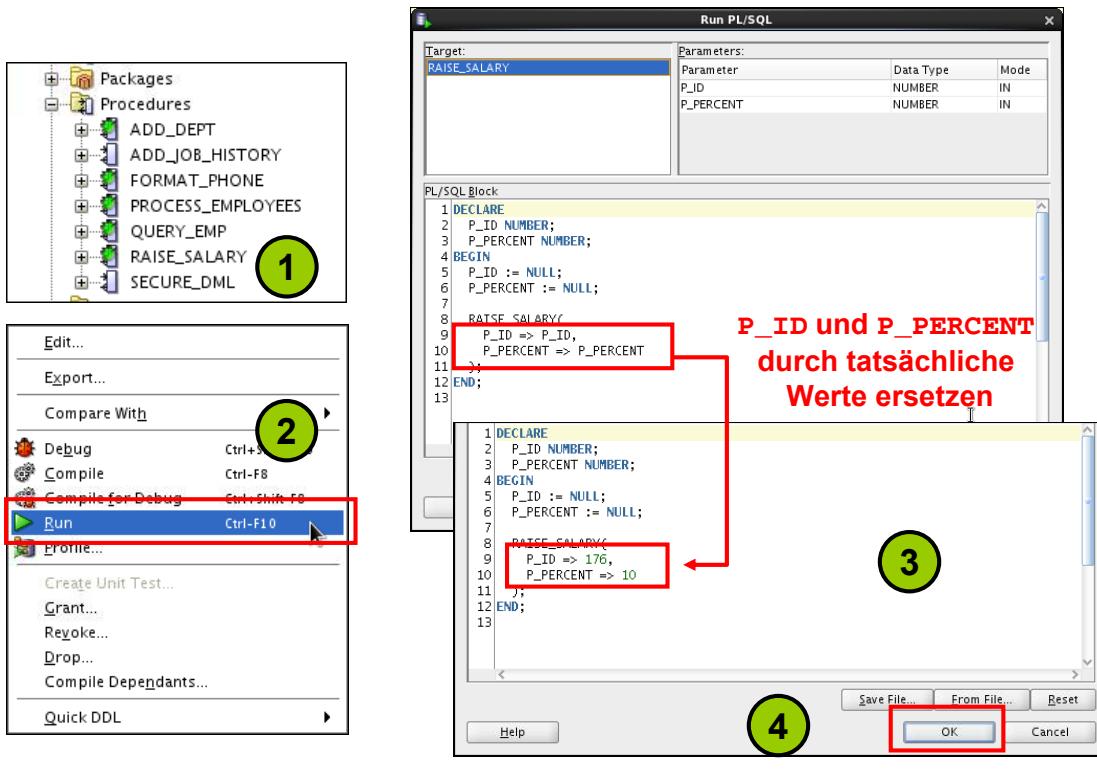
- anonymen Blöcken
- anderen Prozeduren oder PL/SQL-Unterprogrammen

In den Beispielen auf den vorherigen Seiten wurde die Verwendung von anonymen Blöcken (oder des Befehls `EXECUTE` in SQL Developer oder SQL*Plus) veranschaulicht.

Das Beispiel auf der Folie verdeutlicht, wie eine Prozedur aus einer anderen Stored Procedure aufgerufen wird. Die Stored Procedure `PROCESS_EMPLOYEES` verwendet einen Cursor, um alle Records in der Tabelle `EMPLOYEES` zu verarbeiten, und übergibt die Nummern der einzelnen Mitarbeiter an die Prozedur `RAISE_SALARY`. Dies bewirkt unternehmensweit eine Gehalts erhöhung von 10 %.

Hinweis: Sie müssen der Eigentümer der Prozedur sein oder die Berechtigung `EXECUTE` besitzen.

Prozeduren mit SQL Developer aufrufen



ORACLE

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

Im Beispiel auf der Folie wird die Prozedur `raise_salary` aufgerufen, um das tatsächliche Gehalt von Mitarbeiter 176 (8.600 \$) um 10 Prozent zu erhöhen. Dies geschieht wie folgt:

1. Klicken Sie mit der rechten Maustaste im Knoten **Procedures** auf den Prozedurnamen, und klicken Sie dann auf **Run**. Das Dialogfeld **Run PL/SQL** wird angezeigt.
2. Ändern Sie im Bereich **PL/SQL Block** die angezeigten Spezifikationen der formalen Parameter `IN` und `IN/OUT`, die nach dem Zuweisungsoperator "`=>`" angezeigt werden, in die *tatsächlichen* Werte, die Sie zum Ausführen oder Debuggen der Funktion oder Prozedur verwenden möchten. Beispiel: Um das tatsächliche Gehalt (8.600) von Mitarbeiter 176 um 10 Prozent zu erhöhen, können Sie die Prozedur `raise_salary` aufrufen, wie auf der Folie gezeigt. Stellen Sie die Werte für die Eingabeparameter `P_ID` und `P_PERCENT` bereit, d. h. 176 bzw. 10. Dazu ändern Sie die angezeigte Zeichenfolge `P_ID => P_ID` in `P_ID => 176` bzw. `P_PERCENT => P_PERCENT` in `P_PERCENT => 10`.
3. Klicken Sie auf **OK**. SQL Developer führt die Prozedur aus. Das aktualisierte Gehalt von 9.460 wird unten gezeigt:

Results		
	EMPLOYEE_ID	SALARY
1	176	9460

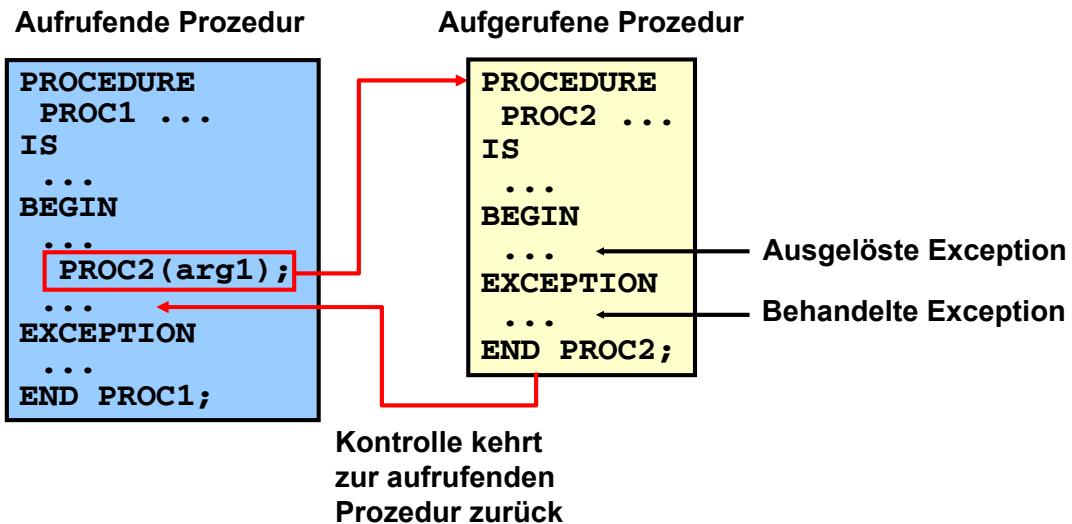
Lektionsagenda

- Modularisierte Unterprogramme sowie Unterprogramme mit Schichten verwenden und Vorteile von Unterprogrammen bestimmen
- Mit Prozeduren arbeiten:
 - Prozeduren erstellen und aufrufen
 - Verfügbare Modi für die Parameterübergabe bestimmen
 - Formale und tatsächliche Parameter verwenden
 - Positionale, benannte oder gemischte Notation verwenden
- Exceptions in Prozeduren behandeln, Prozeduren entfernen und Prozedurinformationen anzeigen
- Reine PL/SQL-Typen binden

ORACLE

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

Behandelte Exceptions



ORACLE

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

Bei der Entwicklung von Prozeduren, die aus anderen Prozeduren aufgerufen werden, müssen Sie wissen, wie sich behandelte und nicht behandelte Exceptions auf die Transaktion und die aufrufende Prozedur auswirken.

Wenn eine Exception in einer aufgerufenen Prozedur ausgelöst wird, geht die Kontrolle sofort an den Exception-Bereich des jeweiligen Blockes über. Exceptions gelten als behandelt, wenn im Exception-Bereich ein Handler für die ausgelöste Exception bereitgestellt ist.

Wenn eine Exception auftritt und behandelt wird, findet der folgende Codeablauf statt:

1. Die Exception wird ausgelöst.
2. Die Kontrolle wird an den Exception Handler übergeben.
3. Der Block wird beendet.
4. Das aufrufende Programm bzw. der aufrufende Block wird weiterhin ordnungsgemäß ausgeführt.

Wenn eine Transaktion gestartet wurde (also Data Manipulation Language-(DML-)Anweisungen vor der Prozedur mit der ausgelösten Exception ausgeführt wurden), bleibt die Transaktion davon unbetroffen. DML-Vorgänge werden zurückgerollt, wenn sie innerhalb der Prozedur vor der Exception ausgeführt wurden.

Hinweis: Sie können Transaktionen explizit beenden, indem Sie im Exception-Bereich ein COMMIT oder ROLLBACK ausführen.

Behandelte Exceptions – Beispiel

```

CREATE PROCEDURE add_department(
    p_name VARCHAR2, p_mgr NUMBER, p_loc NUMBER) IS
BEGIN
    INSERT INTO DEPARTMENTS (department_id,
        department_name, manager_id, location_id)
    VALUES (DEPARTMENTS_SEQ.NEXTVAL, p_name, p_mgr, p_loc);
    DBMS_OUTPUT.PUT_LINE('Added Dept: '|| p_name);
EXCEPTION
    WHEN OTHERS THEN
        DBMS_OUTPUT.PUT_LINE('Err: adding dept: '|| p_name);
END;

```

```

CREATE PROCEDURE create_departments IS
BEGIN
    add_department('Media', 100, 1800);
    → add_department('Editing', 99, 1800); ✗
    add_department('Advertising', 101, 1800); ✓
END;

```

ORACLE

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

Auf der Folie werden die folgenden zwei Prozeduren gezeigt:

- Die Prozedur `add_department` erstellt einen neuen Abteilungs-Record. Hierfür wird von einer Oracle-Sequence eine neue Abteilungsnummer zugewiesen. Außerdem werden die Spaltenwerte von `department_name`, `manager_id` und `location_id` mithilfe der jeweiligen Parameter `p_name`, `p_mgr` und `p_loc` eingestellt.
- Die Prozedur `create_departments` erstellt mehrere Abteilungen mithilfe von Aufrufen der Prozedur `add_department`.

Die Prozedur `add_department` fängt alle ausgelösten Exceptions in ihrem eigenen Handler ab. Bei Ausführung von `create_departments` wird Folgendes ausgegeben:

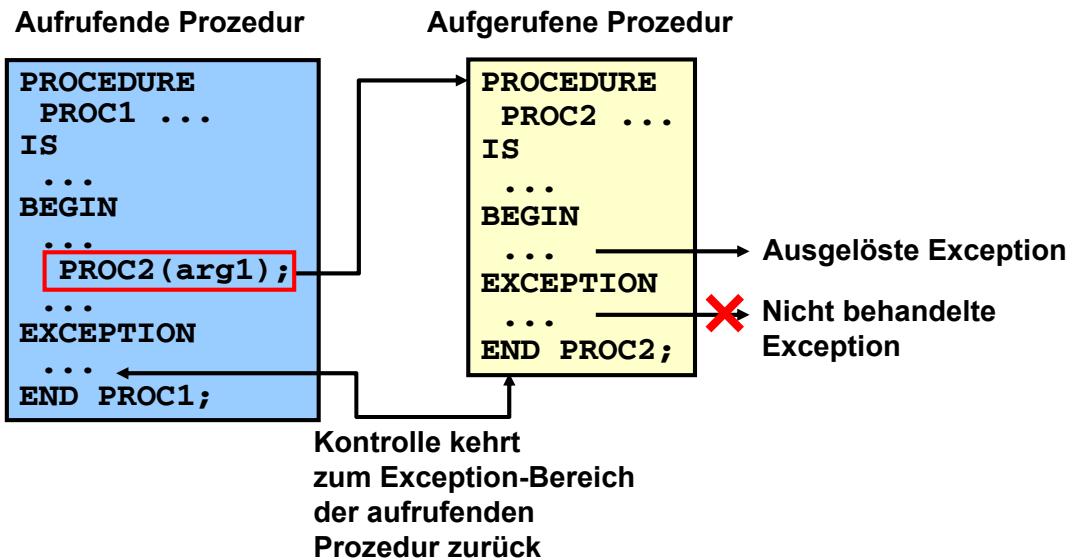
```

Added Dept: Media
Err: adding dept: Editing
Added Dept: Advertising

```

Die Abteilung `Editing` mit der Managernummer (`manager_id`) 99 wird nicht eingefügt, da eine Verletzung des Fremdschlüsselintegritäts-Constraints von `manager_id` gewährleistet, dass kein Manager die Nummer 99 hat. Da die Exception in der Prozedur `add_department` behandelt wurde, kann die Prozedur `create_department` weiter ausgeführt werden. Eine Abfrage an der Tabelle `DEPARTMENTS`, bei der 1800 als Standortnummer (`location_id`) verwendet wird, zeigt, dass `Media` und `Advertising` hinzugefügt werden, der Record `Editing` hingegen nicht.

Nicht behandelte Exceptions



ORACLE

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

Wie bereits erwähnt, geht die Kontrolle sofort an den Exception-Bereich des jeweiligen Blockes über, wenn in einer aufgerufenen Prozedur eine Exception ausgelöst wird. Falls der Exception-Bereich keinen Handler für die ausgelöste Exception bereitstellt, wird sie nicht behandelt. Es kommt zu folgendem Codeablauf:

1. Die Exception wird ausgelöst.
2. Der Block wird beendet, da kein Exception Handler vorhanden ist. Alle innerhalb der Prozedur ausgeführten DML-Vorgänge werden zurückgerollt.
3. Die Exception wird zum Exception-Bereich der aufrufenden Prozedur propagiert. Die Kontrolle geht also an den Exception-Bereich des aufrufenden Blockes zurück, sofern ein solcher vorhanden ist.

Bei nicht behandelten Exceptions werden alle DML-Anweisungen in der aufrufenden und der aufgerufenen Prozedur zusammen mit eventuellen Änderungen von Hostvariablen zurückgerollt. Davon nicht betroffen sind DML-Anweisungen, die ausgeführt wurden, bevor der PL/SQL-Code mit den nicht behandelten Exceptions aufgerufen wurde.

Nicht behandelte Exceptions – Beispiel

```
SET SERVEROUTPUT ON
CREATE PROCEDURE add_department_noex(
    p_name VARCHAR2, p_mgr NUMBER, p_loc NUMBER) IS
BEGIN
    INSERT INTO DEPARTMENTS (department_id,
        department_name, manager_id, location_id)
    VALUES (DEPARTMENTS_SEQ.NEXTVAL, p_name, p_mgr, p_loc);
    DBMS_OUTPUT.PUT_LINE('Added Dept: '|| p_name);
END;
```

```
CREATE PROCEDURE create_departments_noex IS
BEGIN
    add_department_noex('Media', 100, 1800);
    add_department_noex('Editing', 99, 1800);
    add_department_noex('Advertising', 101, 1800);
END;
```

ORACLE

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

Das Codebeispiel auf der Folie zeigt die Prozedur `add_department_noex`. Sie beinhaltet keinen Exception-Bereich. In diesem Beispiel wird eine Exception ausgelöst, wenn die Abteilung `Editing` hinzugefügt wird. Aufgrund der fehlenden Exception-Behandlung in einem der Unterprogramme werden keine neuen Abteilungs-Records in die Tabelle `DEPARTMENTS` eingefügt. Wenn Sie die Prozedur `create_departments_noex` ausführen, sieht das Ergebnis ungefähr wie folgt aus:

```
PROCEDURE CREATE_DEPARTMENTS_NOEX compiled

Error starting at line 8 in command:
EXECUTE create_departments_noex
Error report:
ORA-02291: integrity constraint (ORA61.DEPT_MGR_FK) violated - parent key not found
ORA-06512: at "ORA61.ADD_DEPARTMENT_NOEX", line 4
ORA-06512: at "ORA61.CREATE_DEPARTMENTS_NOEX", line 4
ORA-06512: at line 1
02291. 00000 - "integrity constraint (%s.%s) violated - parent key not found"
*Cause: A foreign key value has no matching primary key value.
*Action: Delete the foreign key or add a matching primary key.
Added Dept: Media
```

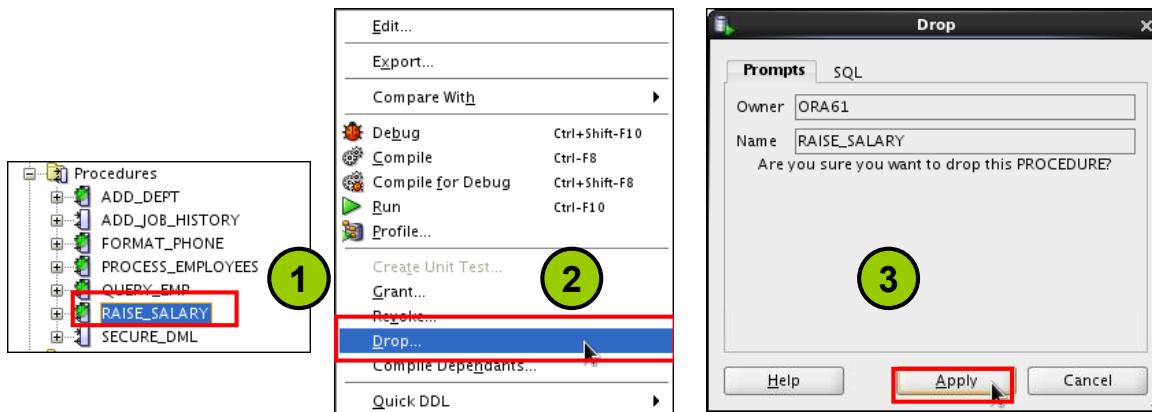
Die Abteilung `Media` wurde hinzugefügt. Der Vorgang wird jedoch zurückgerollt, weil die Exception in keinem der aufgerufenen Unterprogramme behandelt wurde.

Prozeduren mithilfe der SQL-Anweisung DROP oder mit SQL Developer entfernen

- Mit der **DROP**-Anweisung:

```
DROP PROCEDURE raise_salary;
```

- Mit **SQL Developer**:



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

Wenn eine Stored Procedure nicht mehr benötigt wird, können Sie sie mit der SQL-Anweisung **DROP PROCEDURE** gefolgt vom Namen der Prozedur wie folgt löschen:

```
DROP PROCEDURE procedure_name
```

Mit SQL Developer lässt sich eine Stored Procedure ebenfalls löschen. Gehen Sie dazu wie folgt vor:

1. Klicken Sie mit der rechten Maustaste im Knoten **Procedures** auf den Prozedurnamen, und klicken Sie dann auf **Drop**. Das Dialogfeld **Drop** wird angezeigt.
2. Um die Prozedur zu löschen, klicken Sie auf **Apply**.

Hinweis:

- Bei Ausführung eines Data Definition Language-(DDL-)Befehls wie **DROP PROCEDURE** werden alle ausstehenden Transaktionen, die nicht zurückgerollt werden können, festgeschrieben. Dabei ist es unerheblich, ob die Ausführung des Befehls erfolgreich war oder nicht.
- Eventuell müssen Sie den Knoten **Procedures** aktualisieren, damit die Ergebnisse des Löschevorgangs angezeigt werden. Um den Knoten **Procedures** zu aktualisieren, klicken Sie erst mit der rechten Maustaste im Knoten **Procedures** auf den Prozedurnamen und klicken dann auf **Refresh**.

Prozedurinformationen mithilfe von Data Dictionary Views anzeigen

```
DESCRIBE user_source
```

```
DESCRIBE user_source
Name Null Type
-----
NAME      VARCHAR2(128)
TYPE      VARCHAR2(12)
LINE      NUMBER
TEXT      VARCHAR2(4000)
```

```
SELECT text
FROM   user_source
WHERE  name = 'ADD_DEPT' AND type = 'PROCEDURE'
ORDER BY line;
```

```
TEXT
1 PROCEDURE add_dept(
2   p_name departments.department_name%TYPE:='Unknown',
3   p_loc  departments.location_id%TYPE DEFAULT 1700) IS
4
5 BEGIN
6   INSERT INTO departments (department_id, department_name, location_id)
7   VALUES (departments_seq.NEXTVAL, p_name, p_loc);
8 END add_dept;
```

ORACLE

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

Prozeduren im Data Dictionary anzeigen

Der Quellcode für PL/SQL-Unterprogramme wird in den Data Dictionary-Tabellen gespeichert. PL/SQL-Prozeduren können auf den Quellcode zugreifen, unabhängig davon, ob ihre Kompilierung erfolgreich war oder nicht. Um den im Data Dictionary gespeicherten PL/SQL-Quellcode anzuzeigen, führen Sie für die folgenden Tabellen eine SELECT-Anweisung aus:

- Für die Tabelle USER_SOURCE, um PL/SQL-Code anzuzeigen, dessen Eigentümer Sie sind
- Für die Tabelle ALL_SOURCE, um PL/SQL-Code anzuzeigen, für den der Eigentümer des entsprechenden Unterprogrammcodes Ihnen die Berechtigung EXECUTE erteilt hat

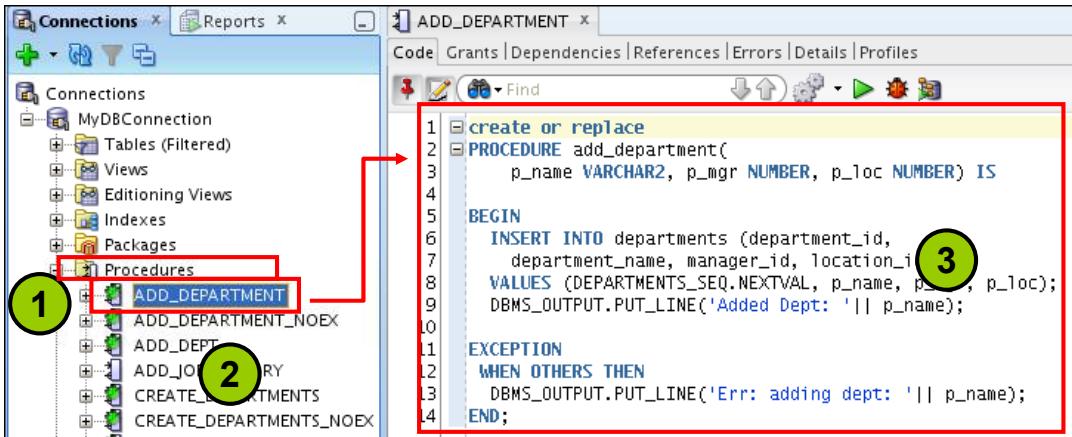
Das Abfragebeispiel zeigt alle Spalten, die in der Tabelle USER_SOURCE verfügbar sind:

- Die Spalte TEXT enthält eine Zeile mit PL/SQL-Quellcode.
- Die Spalte NAME enthält den Namen des Unterprogramms in Großbuchstaben.
- Die Spalte TYPE enthält den Unterprogrammtyp wie PROCEDURE oder FUNCTION.
- Die Spalte LINE speichert die Zeilennummer für die einzelnen Quellcodezeilen.

Die Tabelle ALL_SOURCE stellt neben den vorgenannten Spalten auch die Spalte OWNER bereit.

Hinweis: Die Anzeige des Quellcodes für PL/SQL-Built-In-Packages von Oracle oder für PL/SQL, dessen Quellcode mithilfe eines WRAP-Utilitys gewrapped wurde, ist nicht möglich. Das Utility WRAP konvertiert den PL/SQL-Quellcode in ein Format, das von Unbefugten nicht entschlüsselt werden kann.

Prozedurinformationen mit SQL Developer anzeigen



ORACLE

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

Um den Code einer Prozedur in SQL Developer anzuzeigen, gehen Sie wie folgt vor:

1. Klicken Sie in der Registerkarte **Connections** auf den Knoten **Procedures**.
2. Klicken Sie auf den Namen der Prozedur.
3. Der Prozedurcode wird in der Registerkarte **Code** angezeigt (siehe Folie).

Lektionsagenda

- Modularisierte Unterprogramme sowie Unterprogramme mit Schichten verwenden und Vorteile von Unterprogrammen bestimmen
- Mit Prozeduren arbeiten:
 - Prozeduren erstellen und aufrufen
 - Verfügbare Modi für die Parameterübergabe bestimmen
 - Formale und tatsächliche Parameter verwenden
 - Positionale, benannte oder gemischte Notation verwenden
- Exceptions in Prozeduren behandeln, Prozeduren entfernen und Prozedurinformationen anzeigen
- Reine PL/SQL-Typen binden

ORACLE

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

PL/SQL-Bind-Typen

Ein alterer PL/SQL-Block, die SQL-Anweisung CALL oder eine SQL-Abfrage können eine PL/SQL-Funktion aufrufen, die Parameter der folgenden Typen besitzt:

- **BOOLEAN**
- **In einer Packagespezifikation deklarierter Record**
- **In einer Packagespezifikation deklarierte Collection**



ORACLE

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

Wenn SQL in Oracle Database 11g von PL/SQL aufgerufen wurde, konnte kein Wert mit einem BOOLEAN-, Collection- oder Record-Datentyp gebunden werden, der in einer Packagespezifikation deklariert wurde. Die Einschränkung galt auch, wenn es sich beim aufgerufenen SQL um einen anonymen PL/SQL-Block handelte. Somit konnte PL/SQL kein PL/SQL-Unterprogramm dynamisch aufrufen, das einen formalen Parameter mit einem BOOLEAN-, Collection- oder Record-Datentyp besaß, der in einer Packagespezifikation deklariert wurde.

In Oracle Database 12c wurden diese Einschränkungen aufgehoben, um die Verwendungsmöglichkeiten und die Performance im Zusammenhang mit dieser PL/SQL-Nutzung zu optimieren.

Unterprogramme mit BOOLEAN-Parametern

```
CREATE OR REPLACE PROCEDURE p (x BOOLEAN) AUTHID
    CURRENT_USER AS
BEGIN
    IF x THEN
        DBMS_OUTPUT.PUT_LINE('x is true');
    END IF;
END;
/
DECLARE
    b BOOLEAN := TRUE;
BEGIN
    p(b);
END;
/
```

```
PROCEDURE P compiled
anonymous block completed
x is true
```

ORACLE

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

Im Codebeispiel auf der Folie ist der dynamische PL/SQL-Block ein **anonymer PL/SQL-Block**, der ein Unterprogramm mit einem formalen Parameter des PL/SQL-Datentyps **BOOLEAN** aufruft.

Quiz

Formale Parameter sind Literalwerte, Variablen oder Ausdrücke, die in der Parameterliste des aufrufenden Unterprogramms verwendet werden.

- a. Richtig
- b. Falsch

ORACLE

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

Richtige Antwort: b

Formale und tatsächliche Parameter (oder Argumente)

- **Formale Parameter:** Lokale Variablen, die in der Parameterliste einer Unterprogramm-spezifikation deklariert werden
- **Tatsächliche Parameter:** Literalwerte, Variablen oder Ausdrücke, die in der Parameterliste des aufrufenden Unterprogramms verwendet werden

Zusammenfassung

In dieser Lektion haben Sie Folgendes gelernt:

- **Vorteile von modularisierten Unterprogrammen und Unterprogrammen mit Schichten bestimmen**
- **Prozeduren erstellen und aufrufen**
- **Formale und tatsächliche Parameter verwenden**
- **Parameter mithilfe positionaler, benannter oder gemischter Notation übergeben**
- **Verfügbare Modi für die Parameterübergabe bestimmen**
- **Exceptions in Prozeduren behandeln**
- **Prozeduren entfernen**
- **Prozedurinformationen anzeigen**

ORACLE

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

Übungen zu Lektion 2 – Überblick: Prozeduren erstellen, kompilieren und aufrufen

Diese Übungen behandeln folgende Themen:

- **Stored Procedures für folgende Zwecke erstellen:**
 - Neue Zeilen mithilfe von bereitgestellten Parameterwerten in eine Tabelle einfügen
 - Tabellendaten für Zeilen aktualisieren, die mit den bereitgestellten Parameterwerten übereinstimmen
 - Zeilen, die mit bereitgestellten Parameterwerten übereinstimmen, aus einer Tabelle löschen
 - Tabelle abfragen und Daten auf der Grundlage bereitgestellter Parameterwerte abrufen
- **Exceptions in Prozeduren behandeln**
- **Prozeduren kompilieren und aufrufen**



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

Übungen zu Lektion 2 – Überblick

In diesen Übungen werden Prozeduren erstellt, kompiliert und aufgerufen, die DML- und Abfragebefehle absetzen. Außerdem lernen Sie, wie Exceptions in Prozeduren behandelt werden.

Wenn Sie beim Ausführen von Prozeduren auf Kompilierungsfehler stoßen, können Sie die Registerkarte **Compiler - Log** in SQL Developer verwenden.

Hinweis: Für diese Übung wird SQL Developer empfohlen.

Wichtig

Bei allen Übungen und Lösungen in diesem Kurs wird davon ausgegangen, dass Sie den SQL Worksheet-Bereich in SQL Developer zum Erstellen von Objekten wie Prozeduren oder Funktionen verwenden. Wenn Sie ein Objekt im SQL Worksheet-Bereich erstellen, müssen Sie den Objektnamen aktualisieren, damit das neue Objekt im Object Navigator-Baum angezeigt wird. Um das neu erstellte Objekt zu kompilieren, können Sie im Object Navigator-Baum mit der rechten Maustaste auf den Objektnamen klicken und dann im Kontextmenü **Compile** wählen. Beispiel: Nachdem Sie den Code zur Erstellung einer Prozedur in den SQL Worksheet-Bereich eingegeben haben, klicken Sie auf das Symbol **Run Script** (oder drücken F5), um den Code auszuführen. Damit wird die Prozedur erstellt und kompiliert.

Alternativ können Sie Objekte wie Prozeduren mithilfe des Knotens **PROCEDURES** im Object Navigator-Baum erstellen und dann die Prozedur kompilieren. Wenn Sie Objekte mit dem Object Navigator-Baum erstellen, wird das neu erstellte Objekt automatisch angezeigt.

3

Funktionen erstellen und Unterprogramme debuggen

ORACLE

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

Ziele

Nach Ablauf dieser Lektion haben Sie folgende Ziele erreicht:

- **Prozeduren und Funktionen unterscheiden**
- **Verwendungsmöglichkeiten von Funktionen beschreiben**
- **Stored Functions erstellen**
- **Funktionen aufrufen**
- **Funktionen entfernen**
- **Grundlegende Funktionalität des SQL Developer-Debuggers kennenlernen**

ORACLE

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

In dieser Lektion wird beschrieben, wie Sie Funktionen erstellen, aufrufen und verwalten.

Lektionsagenda

- **Mit Funktionen arbeiten:**
 - Prozeduren und Funktionen unterscheiden
 - Verwendungsmöglichkeiten von Funktionen beschreiben
 - Stored Functions erstellen, aufrufen und entfernen
- **SQL Developer-Debugger – Einführung**

ORACLE

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

Stored Functions – Überblick

Eine Funktion:

- ist ein benannter PL/SQL-Block, der einen Wert zurückgibt
- kann für die wiederholte Ausführung als Schemaobjekt in der Datenbank gespeichert werden
- wird als Teil eines Ausdrucks aufgerufen oder stellt einen Parameterwert für ein anderes Unterprogramm bereit
- kann in PL/SQL-Packages gruppiert werden



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

Eine Funktion ist ein benannter PL/SQL-Block, der Parameter annehmen, aufgerufen werden und Werte zurückgeben kann. Im Allgemeinen dienen Funktionen zum Berechnen von Werten. Funktionen und Prozeduren sind ähnlich strukturiert. Funktionen müssen einen Wert an die aufrufende Umgebung zurückgeben. Prozeduren können null oder mehr Werte an die aufrufende Umgebung zurückgeben. Wie Prozeduren bestehen auch Funktionen aus einem Header, einem deklarativen Bereich, einem ausführbaren Bereich und einem optionalen Bereich zur Exception-Behandlung. Funktionen müssen eine RETURN-Klausel im Header und mindestens eine RETURN-Anweisung im ausführbaren Bereich enthalten.

Funktionen können für die wiederholte Ausführung als Schemaobjekte in der Datenbank gespeichert werden. Eine in der Datenbank gespeicherte Funktion wird als Stored Function bezeichnet. Funktionen lassen sich auch in clientseitigen Anwendungen erstellen.

Funktionen fördern die Wiederverwendbarkeit und Verwaltbarkeit. Wenn sie validiert sind, können sie in beliebig vielen Anwendungen verwendet werden. Falls sich die Verarbeitungsanforderungen ändern, müssen Sie lediglich die jeweilige Funktion aktualisieren.

Funktionen können auch als Teil eines SQL- oder PL/SQL-Ausdrucks aufgerufen werden. In einem SQL-Ausdruck müssen Funktionen bestimmte Regeln einhalten, um Seiteneffekte auszuschalten. In einem PL/SQL-Ausdruck dient der Funktionsbezeichner als Variable. Ihr Wert hängt von dem Parameter ab, der an sie übergeben wird.

Funktionen (und Prozeduren) lassen sich in PL/SQL-Packages gruppieren. Packages können die Wiederverwendbarkeit und Verwaltbarkeit von Code weiter verbessern. Packages werden in den Lektionen "Packages erstellen" und "Mit Packages arbeiten" behandelt.

Funktionen erstellen

Der PL/SQL-Block muss mindestens eine RETURN-Anweisung enthalten.

```
CREATE [OR REPLACE] FUNCTION function_name
  [(parameter1 [mode1] datatype1, . . .)]
  RETURN datatype IS|AS
    [local_variable_declarations;
     . . .]
  BEGIN
    -- actions;
    RETURN expression;
  END [function_name];
```

PL/SQL-Block

ORACLE

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

Funktionen erstellen – Syntax

Eine Funktion ist ein PL/SQL-Block, der einen Wert zurückgibt. Zu diesem Zweck muss eine RETURN-Anweisung bereitgestellt werden. Sie gibt einen Wert zurück, dessen Datentyp mit der Funktionsdeklaration konsistent ist.

Neue Funktionen erstellen Sie mit der Anweisung CREATE FUNCTION. Diese Anweisung gibt einen Wert zurück und definiert die vom PL/SQL-Standardblock auszuführenden Aktionen. Sie kann außerdem eine Parameterliste deklarieren.

Beachten Sie bei der Anweisung CREATE FUNCTION folgende Punkte:

- Die Option REPLACE gibt an, dass die vorhandene Funktion gelöscht und durch die von der Anweisung erstellte neue Version ersetzt wird.
- Der Datentyp RETURN darf keine Größenangabe enthalten.
- Der PL/SQL-Block beginnt nach der Deklaration der lokalen Variablen mit BEGIN und endet mit END. Danach kann optional der Funktionsname (*function_name*) folgen.
- Es muss mindestens eine RETURN *expression*-Anweisung enthalten sein.
- Sie können im PL/SQL-Block einer Stored Function keine Host- oder Bind-Variablen referenzieren.

Hinweis: Zwar können die Parametermodi OUT und IN OUT in Funktionen verwendet werden, doch ist dies keine sinnvolle Vorgehensweise bei der Programmierung. Wenn eine Funktion allerdings mehrere Werte zurückgeben soll, empfiehlt sich eine zusammengesetzte Datenstruktur wie ein PL/SQL-Record oder eine PL/SQL-Tabelle.

Prozeduren und Funktionen – Unterschiede

Prozeduren	Funktionen
Werden als PL/SQL-Anweisung ausgeführt	Werden als Teil eines Ausdrucks aufgerufen
Enthalten keine RETURN-Klausel im Header	Müssen eine RETURN-Klausel im Header enthalten
Können Werte (falls vorhanden) in Ausgabeparametern übergeben	Müssen einen einzigen Wert zurückgeben
Können eine RETURN-Anweisung ohne Wert enthalten	Müssen mindestens eine RETURN-Anweisung enthalten

ORACLE

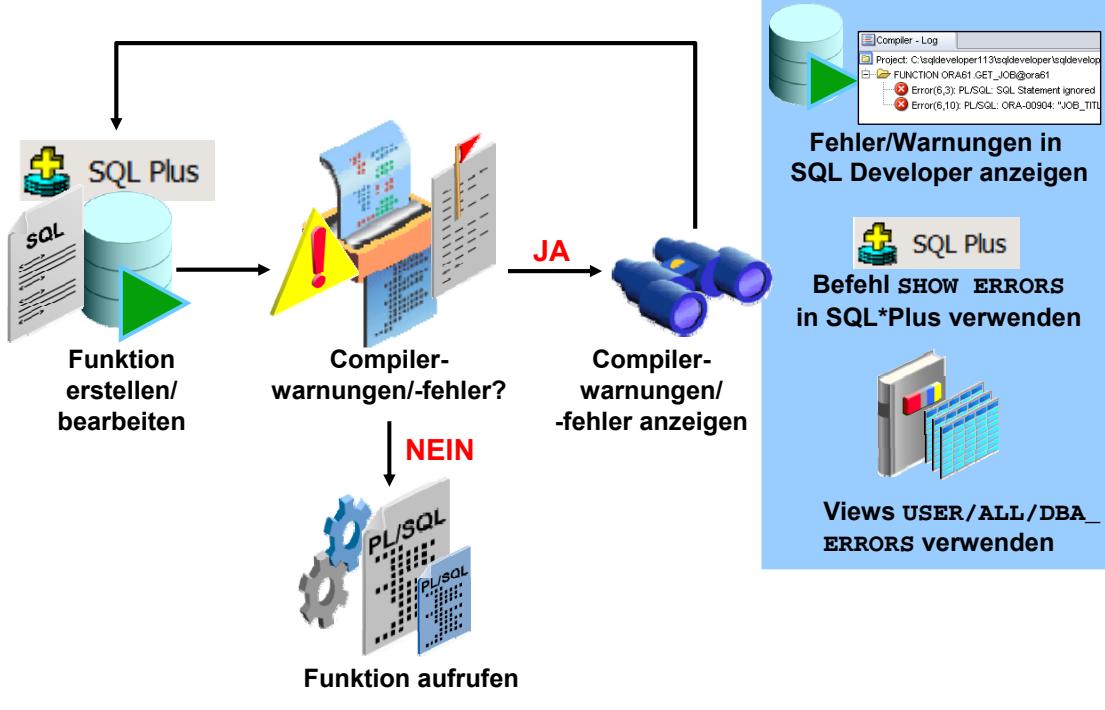
Copyright © 2013, Oracle und/oder verbundene Unternehmen. All rights reserved. Alle Rechte vorbehalten.

Sie erstellen eine Prozedur, um eine Reihe von Aktionen zu speichern, die später ausgeführt werden sollen. Prozeduren können null oder mehr Parameter enthalten, die an die aufrufende oder von der aufrufenden Umgebung übertragen werden. Sie müssen jedoch keinen Wert zurückgeben. Prozeduren können zur Unterstützung ihrer Aktionen Funktionen aufrufen.

Hinweis: Es empfiehlt sich, Prozeduren mit einem einzigen OUT-Parameter als Funktion umzuschreiben, die den Wert zurückgibt.

Sie erstellen eine Funktion, wenn Sie einen Wert berechnen möchten, der an die aufrufende Umgebung zurückgegeben werden muss. Funktionen können null oder mehr Parameter enthalten, die von der aufrufenden Umgebung übertragen werden. Sie geben in der Regel nur einen einzigen Wert zurück und zwar mithilfe einer RETURN-Anweisung. Funktionen, die in SQL-Anweisungen verwendet werden, dürfen keine Parameter vom Typ OUT oder IN OUT enthalten. Funktionen mit Ausgabeparametern können in PL/SQL-Prozeduren oder -Blöcken verwendet werden, nicht jedoch in SQL-Anweisungen.

Funktionen erstellen und ausführen – Überblick



ORACLE

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

Das Diagramm auf der Folie veranschaulicht die Hauptschritte beim Erstellen und Ausführen einer Funktion:

1. Erstellen Sie die Funktion mithilfe des Object Navigator-Baumes oder des SQL Worksheet-Bereichs von SQL Developer.
2. Komplizieren Sie die Funktion. Die Funktion wird in der Datenbank erstellt. Die Anweisung `CREATE FUNCTION` erstellt und speichert den Quellcode sowie den kompilierten *m*-Code in der Datenbank. Um die Funktion zu komplizieren, klicken Sie mit der rechten Maustaste auf den Namen der Funktion im Object Navigator-Baum und klicken anschließend auf **Compile**.
3. Wenn Komplizierungswarnungen oder -fehler aufgetreten sind, können Sie diese mit einer der folgenden Methoden anzeigen (und dann korrigieren):
 - a. SQL Developer-Benutzeroberfläche (Registerkarte **Compiler – Log**)
 - b. SQL*Plus-Befehl `SHOW ERRORS`
 - c. Views `USER/ALL/DBA_ERRORS`
4. Um den gewünschten Wert zurückzugeben, rufen Sie nach erfolgreicher Komplizierung die Funktion auf.

Stored Functions mit CREATE FUNCTION-Anweisungen erstellen und aufrufen – Beispiel

```
CREATE OR REPLACE FUNCTION get_sal
(p_id employees.employee_id%TYPE) RETURN NUMBER IS
  v_sal employees.salary%TYPE := 0;
BEGIN
  SELECT salary
  INTO   v_sal
  FROM   employees
  WHERE  employee_id = p_id;
  RETURN v_sal;
END get_sal;
/
```

FUNCTION GET_SAL compiled

```
-- Invoke the function as an expression or as
-- a parameter value.
```

```
EXECUTE dbms_output.put_line(get_sal(100))
```

anonymous block completed
24000

ORACLE

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

Stored Functions – Beispiel

Die Funktion `get_sal` wird mit einem einzigen Eingabeparameter erstellt und gibt das Gehalt in Form einer Zahl zurück. Führen Sie den Befehl wie oben gezeigt aus, oder speichern Sie ihn als Skriptdatei, und führen Sie das Skript zur Erstellung der Funktion `get_sal` aus.

Die Funktion `get_sal` folgt die gängige Vorgehensweise bei der Programmierung und verwendet eine einzige `RETURN`-Anweisung, um einen Wert zurückzugeben, der einer lokalen Variablen zugeordnet ist. Wenn die Funktion über einen Exception-Bereich verfügt, kann sie auch eine `RETURN`-Anweisung enthalten.

Rufen Sie eine Funktion als Teil eines PL/SQL-Ausdrucks auf, da die Funktion einen Wert an die aufrufende Umgebung zurückgibt. Im zweiten Codefeld wird mit dem SQL*Plus-Befehl `EXECUTE` die Prozedur `DBMS_OUTPUT.PUT_LINE` aufgerufen, deren Argument der Rückgabewert der Funktion `get_sal` ist. In diesem Fall wird `get_sal` zunächst zur Berechnung des Gehalts des Mitarbeiters mit der Personalnummer 100 aufgerufen. Das zurückgegebene Gehalt wird als Wert des Parameters `DBMS_OUTPUT.PUT_LINE` bereitgestellt, der das Ergebnis anzeigt (falls Sie `SET SERVEROUTPUT ON` ausgeführt haben).

Hinweis: Funktionen müssen stets einen Wert zurückgeben. Das Beispiel gibt keinen Wert zurück, wenn für eine angegebene Nummer (`id`) keine Zeile gefunden wird. Idealerweise erstellen Sie einen Exception Handler, um auch in diesen Fällen einen Wert zurückzugeben.

Funktionen mithilfe verschiedener Methoden ausführen

```
-- As a PL/SQL expression, get the results using host variables
```

```
VARIABLE b_salary NUMBER
EXECUTE :b_salary := get_sal(100)
```

```
anonymous block completed
B_SALARY
-----
24000
```

```
-- As a PL/SQL expression, get the results using a local
-- variable
```

```
SET SERVEROUTPUT ON
DECLARE
    sal employees.salary%type;
BEGIN
    sal := get_sal(100);
    DBMS_OUTPUT.PUT_LINE('The salary is: '|| sal);
END;
/
```

```
anonymous block completed
The salary is: 24000
```

ORACLE

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

Gut konzipierte Funktionen können sehr effizient sein. Funktionen lassen sich wie folgt aufrufen:

- **Als Teil von PL/SQL-Ausdrücken:** Der zurückgegebene Wert einer Funktion kann in Hostvariablen oder lokale Variablen aufgenommen werden. Im ersten Beispiel auf der Folie wird eine Hostvariable verwendet, im zweiten eine lokale Variable in einem anonymen Block.

Hinweis: Die Vorteile und Einschränkungen von Funktionen in einer SQL-Anweisung werden auf den nächsten Seiten erläutert.

Funktionen mithilfe verschiedener Methoden ausführen

```
-- Use as a parameter to another subprogram
```

```
EXECUTE dbms_output.put_line(get_sal(100))
```

```
anonymous block completed
24000
```

```
-- Use in a SQL statement (subject to restrictions)
```

```
SELECT job_id, get_sal(employee_id)
FROM employees;
```

JOB_ID	GET_SAL(EMPLOYEE_ID)
AC_ACCOUNT	8300
AC_MGR	12008
AD_ASST	4400
AD_PRES	24000
...	

ST_MAN	6500
ST_MAN	5800

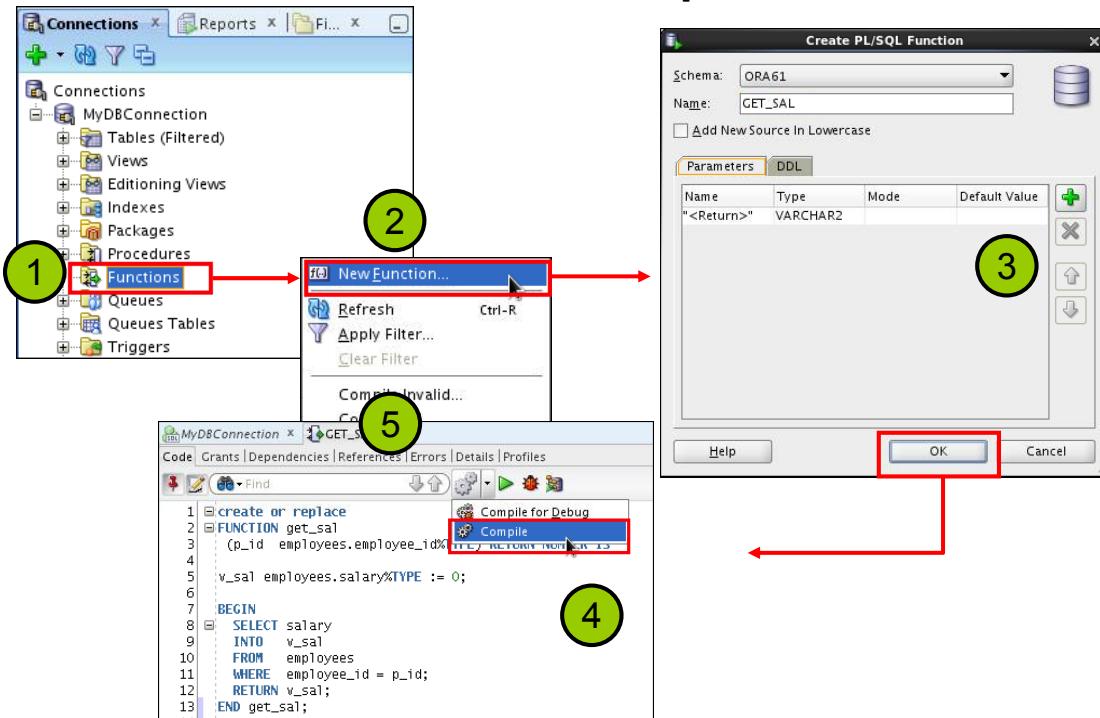
107 rows selected

ORACLE

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

- **Als Parameter für ein anderes Unterprogramm:** Diese Verwendung wird im ersten Beispiel auf der Folie veranschaulicht. Die Funktion `get_sal` ist mit all ihren Argumenten in dem Parameter verschachtelt, der für die Prozedur `DBMS_OUTPUT.PUT_LINE` benötigt wird. Dies basiert auf dem Konzept der verschachtelten Funktionen, wie im Kurs *Oracle Database: SQL Fundamentals I* erläutert.
- **Als Ausdruck in einer SQL-Anweisung:** Das zweite Beispiel auf der Folie zeigt, wie eine Funktion als Single Row-Funktion in einer SQL-Anweisung verwendet werden kann.

Funktionen mit SQL Developer erstellen und kompilieren



ORACLE

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

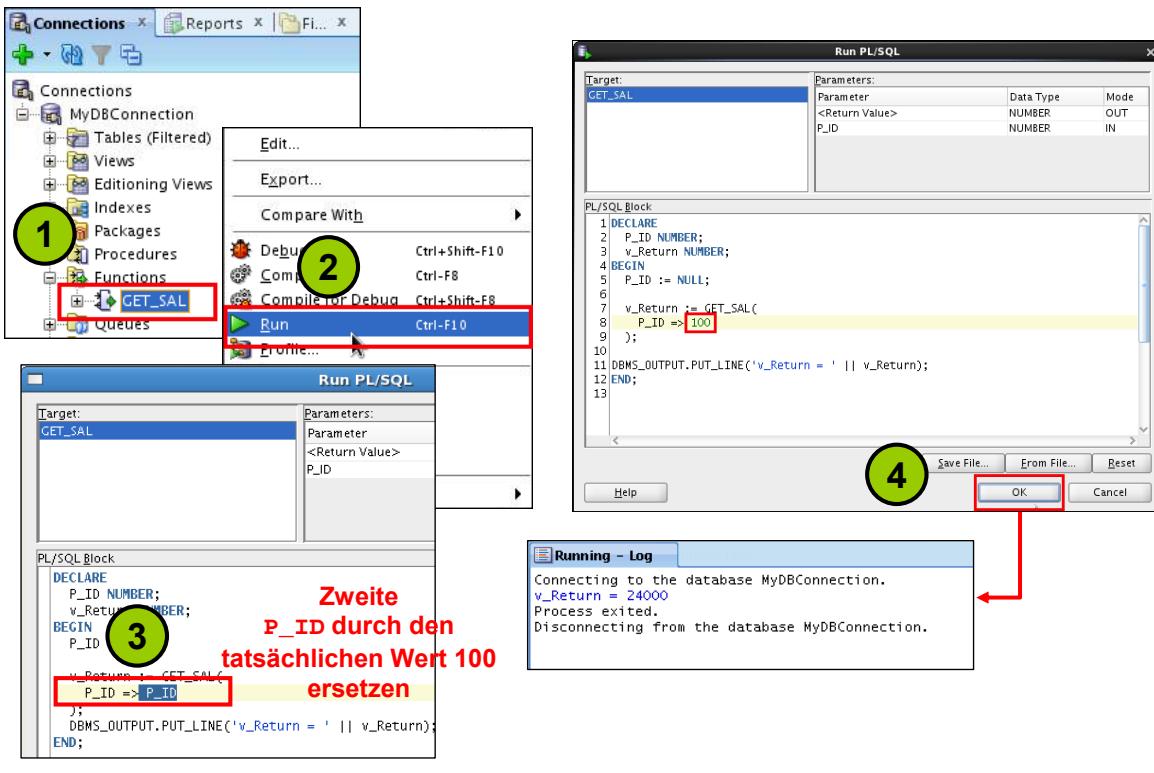
Um in SQL Developer eine neue Funktion zu erstellen, gehen Sie wie folgt vor:

1. Klicken Sie mit der rechten Maustaste auf den Knoten **Functions**.
2. Wählen Sie im Kontextmenü die Option **New Function**. Das Dialogfeld **Create PL/SQL Function** wird angezeigt.
3. Wählen Sie das Schema, den Funktionsnamen und die Parameterliste (mithilfe des Symbols +), und klicken Sie dann auf **OK**. Der Code Editor für die Funktion wird angezeigt.
4. Geben Sie den Code der Funktion ein.
5. Um die Funktion zu kompilieren, klicken Sie auf das Symbol **Compile**.

Hinweis:

- Um eine neue Funktion in SQL Developer zu erstellen, können Sie den Code auch in das SQL Worksheet eingeben und dann auf das Symbol **Run Script** klicken.
- Weitere Informationen zum Erstellen von Funktionen in SQL Developer finden Sie im entsprechenden Onlinehilfethema "Create PL/SQL Subprogram Function or Procedure".

Funktionen mit SQL Developer ausführen



ORACLE

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

Um eine Funktion in SQL Developer auszuführen, gehen Sie wie folgt vor:

1. Klicken Sie auf den Knoten **Functions**.
2. Klicken Sie mit der rechten Maustaste auf den Namen der Funktion, und wählen Sie **Run**. Das Dialogfeld **Run PL/SQL** wird angezeigt.
3. Ersetzen Sie den zweiten Parameternamen durch den tatsächlichen Parameterwert, wie im Beispiel auf der Folie gezeigt.
4. Klicken Sie auf **OK**.

Hinweis: Weitere Informationen zum Ausführen von Funktionen in SQL Developer finden Sie im Onlinehilfethema "Running and Debugging Functions and Procedures".

Benutzerdefinierte Funktionen in SQL-Anweisungen – Vorteile

- Können SQL erweitern, wenn Aktivitäten in SQL zu komplex, zu umständlich oder nicht verfügbar sind
- Können die Effizienz steigern, indem Daten in der Klausel WHERE und nicht in der Anwendung gefiltert werden
- Können Datenwerte bearbeiten

ORACLE

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

Benutzerdefinierte PL/SQL-Funktionen können in SQL-Anweisungen an allen Positionen referenziert werden, an denen SQL-Ausdrücke zulässig sind. Beispielsweise können benutzerdefinierte Funktionen an allen Positionen verwendet werden, an denen eine Built-In-SQL-Funktion wie UPPER() eingefügt werden kann.

Vorteile

- Benutzerdefinierte Funktionen ermöglichen Berechnungen, die mit SQL zu komplex, zu umständlich oder nicht verfügbar sind. Funktionen erhöhen die Unabhängigkeit der Daten, da komplexe Datenanalysen im Oracle-Server verarbeitet werden und die Daten nicht in eine Anwendung abgerufen werden müssen.
- Sie steigern die Effizienz von Abfragen, da Funktionen in der Abfrage und nicht in der Anwendung ausgeführt werden
- Sie bearbeiten neue Datentypen (z. B. LATITUDE und LONGITUDE), indem Zeichenfolgen codiert und mit Funktionen bearbeitet werden.

Funktionen in SQL-Ausdrücken – Beispiel

```

CREATE OR REPLACE FUNCTION tax(p_value IN NUMBER)
  RETURN NUMBER IS
BEGIN
  RETURN (p_value * 0.08);
END tax;
/
SELECT employee_id, last_name, salary, tax(salary)
FROM   employees
WHERE  department_id = 100;

```

FUNCTION TAX compiled	EMPLOYEE_ID LAST_NAME	SALARY	TAX(SALARY)
	108 Greenberg	12008	960.64
	109 Faviet	9000	720
	110 Chen	8200	656
	111 Sciarra	7700	616
	112 Urman	7800	624
	113 Popp	6900	552
6 rows selected			

ORACLE

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

Funktionen in SQL-Ausdrücken – Beispiel

Das Beispiel auf der Folie zeigt die Berechnung der Einkommensteuer mit einer `tax`-Funktion. Die Funktion nimmt einen `NUMBER`-Parameter an und gibt die auf der Basis eines einfachen Einheitssteuersatzes von 8 % berechnete Einkommensteuer zurück.

Um den Code der Folie in SQL Developer auszuführen, geben Sie ihn in das SQL Worksheet ein und klicken dann auf das Symbol **Run Script**. Die Funktion `tax` wird in der Klausel `SELECT` als Ausdruck zusammen mit Personalnummer, Nachname und Gehalt von Mitarbeitern aus der Abteilung 100 aufgerufen. Das Rückgabergebnis der Funktion `tax` wird mit der normalen Ausgabe der Abfrage angezeigt.

Benutzerdefinierte Funktionen in SQL-Anweisungen aufrufen

Benutzerdefinierte Funktionen fungieren wie Built-In Single Row-Funktionen in:

- **SELECT-Listen oder -Klauseln von Abfragen**
- **bedingten Ausdrücken von WHERE- und HAVING-Klauseln**
- **CONNECT BY-, START WITH-, ORDER BY- und GROUP BY-Klauseln von Abfragen**
- **VALUES-Klauseln von INSERT-Anweisungen**
- **SET-Klauseln von UPDATE-Anweisungen**

ORACLE

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

Benutzerdefinierte PL/SQL-Funktionen können aus allen SQL-Ausdrücken aufgerufen werden, aus denen eine Built-In Single Row-Funktion aufgerufen werden kann, wie im folgenden Beispiel gezeigt:

```
SELECT employee_id, tax(salary)
  FROM employees
 WHERE tax(salary) > (SELECT MAX(tax(salary))
                           FROM employees
                          WHERE department_id = 30)
 ORDER BY tax(salary) DESC;
```

EMPLOYEE_ID	TAX(SALARY)
100	1920
101	1360
102	1360
145	1120
146	1080
201	1040
205	960.64
108	960.64
147	960
168	920

10 rows selected

Funktionen aus SQL-Ausdrücken aufrufen – Einschränkungen

- **Damit benutzerdefinierte Funktionen aus SQL-Ausdrücken aufgerufen werden können, müssen sie:**
 - in der Datenbank gespeichert sein
 - ausschließlich `IN`-Parameter mit gültigen SQL-Datentypen und PL/SQL-spezifischen Datentypen annehmen
 - gültige SQL-Datentypen und PL/SQL-spezifische Datentypen zurückgeben
- **Für den Aufruf von Funktionen in SQL-Anweisungen:**
 - müssen Sie der Eigentümer der Funktion sein oder die Berechtigung `EXECUTE` besitzen
 - müssen Sie gegebenenfalls das Schlüsselwort `PARALLEL_ENABLE` aktivieren, um eine parallele Ausführung der SQL-Anweisung zu ermöglichen

ORACLE

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

Damit benutzerdefinierte PL/SQL-Funktionen aus SQL-Ausdrücken aufgerufen werden können, müssen sie folgende Voraussetzungen erfüllen:

- Sie müssen in der Datenbank gespeichert sein.
- Ihre Parameter müssen `IN`-Parameter mit gültigen SQL-Datentypen und PL/SQL-spezifischen Datentypen sein.
- Sie müssen Datentypen zurückgeben, die gültige SQL-Datentypen und PL/SQL-spezifische Datentypen sind, wie `BOOLEAN`, `RECORD` oder `TABLE`.

Beim Aufruf von Funktionen in einer SQL-Anweisung gelten folgende Einschränkungen:

- Parameter müssen die positionale Notation oder die benannte Notation verwenden.
- Sie müssen der Eigentümer der Funktion sein oder die Berechtigung `EXECUTE` besitzen.
- Sie müssen gegebenenfalls das Schlüsselwort `PARALLEL_ENABLE` aktivieren, um eine parallele Ausführung der SQL-Anweisung mithilfe der Funktion zu ermöglichen. Jeder parallele Unterprozess verfügt über private Kopien der lokalen Variablen der Funktion.

Weitere Einschränkungen für benutzerdefinierte Funktionen: Sie können nicht aus der `CHECK` Constraint-Klausel einer `CREATE TABLE`- oder `ALTER TABLE`-Anweisung aufgerufen werden. Darüber hinaus können sie nicht verwendet werden, um Standardwerte für Spalten anzugeben. Nur Stored Functions können aus SQL-Anweisungen aufgerufen werden. Stored Procedures lassen sich nur aus Funktionen aufrufen, die die oben genannten Anforderungen erfüllen.

Funktionen aus SQL-Ausdrücken aufrufen – Seiteneffekte ausschalten

Funktionen, die aus:

- **SELECT-Anweisungen aufgerufen werden, dürfen keine DML-Anweisungen enthalten**
- **UPDATE- oder DELETE-Anweisungen für eine T-Tabelle aufgerufen werden, dürfen in derselben T-Tabelle kein DML abfragen oder enthalten**
- **SQL-Anweisungen aufgerufen werden, dürfen Transaktionen nicht beenden (d. h. kein COMMIT oder ROLLBACK ausführen)**

Hinweis: Aufrufe von Unterprogrammen, die gegen diese Einschränkungen verstößen, sind in Funktionen ebenfalls nicht zulässig.

ORACLE

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

Um eine SQL-Anweisung auszuführen, die eine Stored Function auuftut, muss der Oracle-Server wissen, ob die Funktion frei von bestimmten Seiteneffekten ist. Seiteneffekte sind unzulässige Änderungen an Datenbanktabellen.

Für den Aufruf von Funktionen in Ausdrücken von SQL-Anweisungen gelten zusätzliche Einschränkungen:

- Funktionen, die aus einer SELECT-Anweisung oder einer parallelen UPDATE- oder DELETE-Anweisung aufgerufen werden, können keine Datenbanktabellen ändern.
- Funktionen, die aus einer UPDATE- oder DELETE-Anweisung aufgerufen werden, können keine Datenbanktabellen abfragen oder ändern, die durch diese Anweisung geändert werden.
- Funktionen, die aus einer SELECT-, INSERT-, UPDATE- oder DELETE-Anweisung aufgerufen wird, können weder direkt noch indirekt über andere Unterprogramme oder folgende SQL-Steueranweisungen für Transaktionen ausgeführt werden:
 - COMMIT- oder ROLLBACK-Anweisungen
 - Steueranweisungen für Sessions (z. B. SET ROLE)
 - Steueranweisungen für das System (z. B. ALTER SYSTEM)
 - DDL-Anweisungen (z. B. CREATE), da ein automatisches Commit folgt

Funktionen aus SQL aufrufen – Einschränkungen: Beispiel

```
CREATE OR REPLACE FUNCTION dml_call_sql(p_sal NUMBER)
  RETURN NUMBER IS
BEGIN
  INSERT INTO employees(employee_id, last_name,
                        email, hire_date, job_id, salary)
  VALUES(1, 'Frost', 'jfrost@company.com',
         SYSDATE, 'SA_MAN', p_sal);
  RETURN (p_sal + 100);
END;
```

```
UPDATE employees
  SET salary = dml_call_sql(2000)
 WHERE employee_id = 170;
```

```
FUNCTION DML_CALL_SQL compiled
Error starting at line 127 in command:
UPDATE employees
  SET salary = dml_call_sql(2000)
 WHERE employee_id = 170
Error report:
SQL Error: ORA-04091: table ORA$1.EMPLOYEES is mutating, trigger/function may not see it
ORA-06512: at "ORA$1.DML_CALL_SQL", line 4
04091. 00000 -  "table %s.%s is mutating, trigger/function may not see it"
*Cause: A trigger (or a user defined plsql function that is referenced in
this statement) attempted to look at (or modify) a table that was
in the middle of being modified by the statement which fired it.
*Action: Rewrite the trigger (or function) so it does not read that table.
```

ORACLE

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

Die Funktion `dml_call_sql` auf der Folie enthält eine `INSERT`-Anweisung, mit der ein neuer Record in die Tabelle `EMPLOYEES` eingefügt und der Eingabewert für das Gehalt in 100er-Schritten zurückgegeben wird. Diese Funktion wird in der Anweisung `UPDATE` aufgerufen, mit der das Gehalt des Mitarbeiters 170 in den von der Funktion zurückgegebenen Wert geändert wird. Die Anweisung `UPDATE` wird nicht erfolgreich ausgeführt. In einer Fehlermeldung wird angezeigt, dass sich die Tabelle verändert (d. h., es werden bereits Änderungen an derselben Tabelle vorgenommen). Im folgenden Beispiel fragt die Funktion `query_call_sql` die Spalte `SALARY` der Tabelle `EMPLOYEES` ab:

```
CREATE OR REPLACE FUNCTION query_call_sql(p_a NUMBER)
  RETURN NUMBER IS
  v_s NUMBER;
BEGIN
  SELECT salary INTO v_s FROM employees
  WHERE employee_id = 170;
  RETURN (v_s + p_a);
END;
```

Beim Aufruf aus der folgenden `UPDATE`-Anweisung wird eine ähnliche Fehlermeldung wie auf der Folie zurückgegeben:

```
UPDATE employees SET salary = query_call_sql(100)
WHERE employee_id = 170;
```

Benannte und gemischte Notation aus SQL

- In PL/SQL können Argumente in Aufrufen von untergeordneten Routinen mit der positionalen, benannten oder gemischten Notation angegeben werden.
- Vor Oracle Database 11g wurde in Aufrufen aus SQL nur die positionale Notation unterstützt.
- Ab Oracle Database 11g kann die benannte und gemischte Notation zur Angabe von Argumenten in Aufrufen von untergeordneten PL/SQL-Routinen aus SQL-Anweisungen verwendet werden.
- In langen Parameterlisten, die hauptsächlich Standardwerte enthalten, können Sie Werte aus den optionalen Parametern weglassen.
- Der Standardwert des optionalen Parameters muss nicht bei jeder Aufrufsite dupliziert werden.

ORACLE

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

Benannte und gemischte Notation aus SQL – Beispiel

```
CREATE OR REPLACE FUNCTION f(
    p_parameter_1 IN NUMBER DEFAULT 1,
    p_parameter_5 IN NUMBER DEFAULT 5)
RETURN NUMBER
IS
    v_var number;
BEGIN
    v_var := p_parameter_1 + (p_parameter_5 * 2);
    RETURN v_var;
END f;
/
```

```
SELECT f(p_parameter_5 => 10) FROM DUAL;
```

```
FUNCTION F compiled
F(P_PARAMETER_5=>10)
-----
21
```

ORACLE

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

Benannte und gemischte Notation in SQL-Anweisungen – Beispiel

Im Beispiel auf der Folie wird im Aufruf der Funktion `f` in der SQL-Anweisung `SELECT` die benannte Notation verwendet. Vor Oracle Database 11g stand die benannte oder gemischte Notation für die Parameterübergabe an eine Funktion aus einer SQL-Anweisung nicht zur Verfügung. Vor Oracle Database 11g wurde der folgende Fehler ausgegeben:

```
SELECT f(p_parameter_5 => 10) FROM DUAL;
```

ORA-00907: missing right parenthesis

Funktionen mit Data Dictionary Views anzeigen

```
DESCRIBE USER_SOURCE
```

```
DESCRIBE user_source
Name Null Type
-----
NAME      VARCHAR2(128)
TYPE      VARCHAR2(12)
LINE      NUMBER
TEXT      VARCHAR2(4000)
```

```
SELECT text
FROM user_source
WHERE type = 'FUNCTION'
ORDER BY line;
```

TEXT
1 FUNCTION dml_call_sql(p_sal NUMBER) 2 FUNCTION tax(p_value IN NUMBER) 3 FUNCTION query_call_sql(p_a NUMBER) RETURN NUMBER IS 4 FUNCTION get_sal 5 RETURN NUMBER IS 6 RETURN NUMBER IS 7 (p_id employees.employee_id%TYPE) RETURN NUMBER IS 8 v_s NUMBER;

...

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

Der Quellcode für PL/SQL-Funktionen wird in den Data Dictionary-Tabellen gespeichert. PL/SQL-Funktionen können auf den Quellcode unabhängig davon zugreifen, ob ihre Kompilierung erfolgreich war oder nicht. Um den im Data Dictionary gespeicherten Code der PL/SQL-Funktionen anzuzeigen, führen Sie für folgende Tabellen eine SELECT-Anweisung aus (die Spalte TYPE hat den Wert FUNCTION):

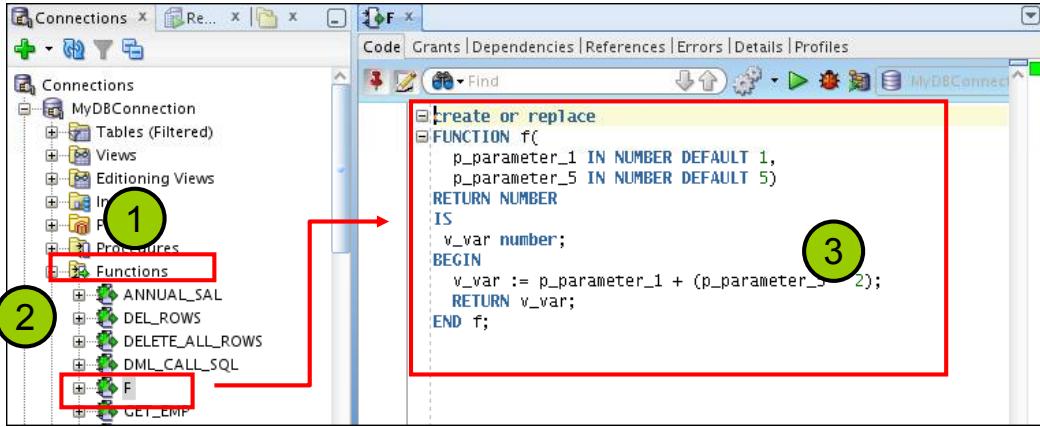
- Tabelle USER_SOURCE zur Anzeige des PL/SQL-Codes, dessen Eigentümer Sie sind
- Tabelle ALL_SOURCE zur Anzeige des PL/SQL-Codes, für den Ihnen der Eigentümer des entsprechenden Unterprogrammcodes die Berechtigung EXECUTE erteilt hat

Im zweiten Beispiel auf der Folie wird der Quellcode für alle Funktionen in Ihrem Schema aus der Tabelle USER_SOURCE abgerufen.

Sie können auch die Data Dictionary View USER_OBJECTS verwenden, um eine Liste der Funktionsnamen anzuzeigen.

Hinweis: Die Ausgabe des zweiten Codebeispiels auf der Folie wurde mithilfe des Symbols **Execute Statement** (F9) in der Symbolleiste generiert, um die Formatierung der Ausgabe zu optimieren.

Funktionsinformationen mit SQL Developer anzeigen



ORACLE

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

Um den Code einer Funktion in SQL Developer anzuzeigen, gehen Sie wie folgt vor:

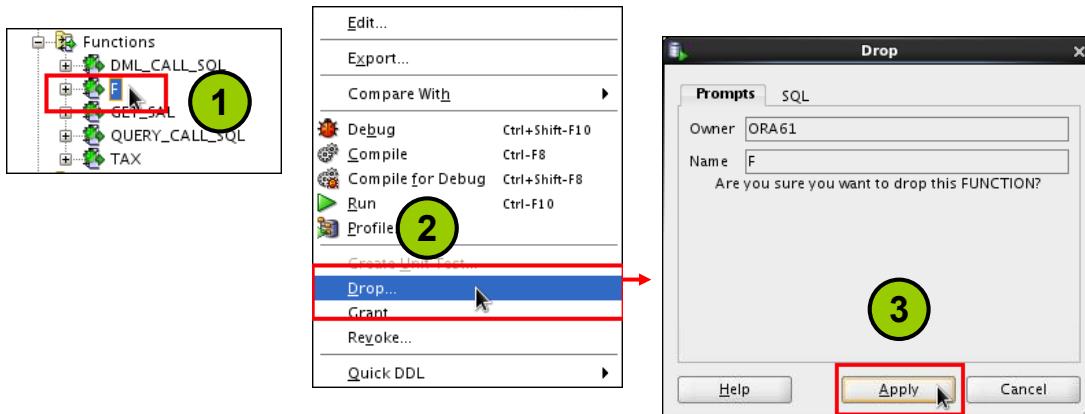
1. Klicken Sie in der Registerkarte **Connections** auf den Knoten **Functions**.
2. Klicken Sie auf den Namen der Funktion.
3. Der Funktionscode wird in der Registerkarte **Code** angezeigt (siehe Folie).

Funktionen mit der SQL-Anweisung **DROP** oder mit SQL Developer entfernen

- Mit der Anweisung **DROP**:

```
DROP FUNCTION f;
```

- Mit SQL Developer:



ORACLE

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

Funktionen entfernen

Mit der Anweisung **DROP**

Wird eine Stored Function nicht mehr benötigt, können Sie sie mit einer SQL-Anweisung in SQL*Plus löschen. Zum Entfernen einer Stored Function mit SQL*Plus führen Sie den SQL-Befehl **DROP FUNCTION** aus.

Mit **CREATE OR REPLACE** im Vergleich zu **DROP** und **CREATE**

Die Klausel **REPLACE** in der Syntax **CREATE OR REPLACE** entspricht dem Löschen und Neuerstellen einer Funktion. Bei Verwendung der Syntax **CREATE OR REPLACE** bleiben die Berechtigungen erhalten, die anderen Benutzern für dieses Objekt erteilt wurden. Wenn Sie eine Funktion mit dem Befehl **DROP** löschen und anschließend neu erstellen, werden alle für diese Funktion erteilten Berechtigungen automatisch entzogen.

Mit SQL Developer

Um eine Funktion in SQL Developer zu löschen, klicken Sie im Knoten **Functions** mit der rechten Maustaste auf den Namen der Funktion und wählen dann **Drop**. Das Dialogfeld **Drop** wird angezeigt. Um die Funktion zu löschen, klicken Sie auf **Apply**.

Quiz

Eine PL/SQL-Stored Function:

- a. kann als Teil eines Ausdrucks aufgerufen werden
- b. muss eine RETURN-Klausel im Header enthalten
- c. muss einen einzigen Wert zurückgeben
- d. muss mindestens eine RETURN-Anweisung enthalten
- e. enthält keine RETURN-Klausel im Header

ORACLE

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

Richtige Antworten: a, b, c, d

Übung 1 zu Lektion 3 – Überblick

Diese Übungen behandeln folgende Themen:

- **Stored Functions für folgende Zwecke erstellen:**
 - Datenbanktabelle abfragen und bestimmte Werte zurückgeben
 - In einer SQL-Anweisung verwenden
 - Eine neue Zeile mit angegebenen Parameterwerten in eine Datenbanktabelle einfügen
 - Standardparameterwerte verwenden
- **Stored Function aus einer SQL-Anweisung aufrufen**
- **Stored Function aus einer Stored Procedure aufrufen**

ORACLE

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

Für diese Übung wird SQL Developer empfohlen.

Lektionsagenda

- **Mit Funktionen arbeiten:**
 - Prozeduren und Funktionen unterscheiden
 - Verwendungsmöglichkeiten von Funktionen beschreiben
 - Stored Functions erstellen, aufrufen und entfernen
- **SQL Developer-Debugger – Einführung**

ORACLE

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

PL/SQL-Unterprogramme mit SQL Developer-Debugger debuggen

- Sie können die Ausführung des PL/SQL-Programms mit dem Debugger steuern.
- Zum Debuggen eines PL/SQL-Unterprogramms müssen dem Anwendungsentwickler die folgenden Berechtigungen vom **Sicherheitsadministrator** erteilt werden:
 - DEBUG ANY PROCEDURE
 - DEBUG CONNECT SESSION

```
GRANT DEBUG ANY PROCEDURE TO ora61;
GRANT DEBUG CONNECT SESSION TO ora61;
```

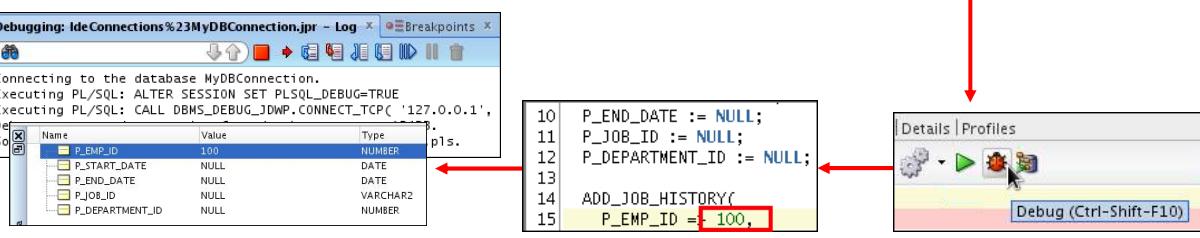
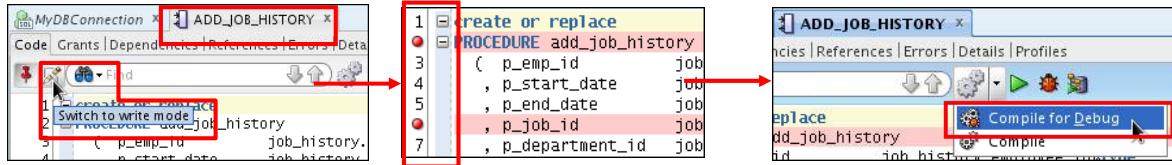
ORACLE

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

Code beim Debugging schrittweise ablaufen lassen

Mit dem SQL Developer-Debugger lässt sich die Ausführung des Programms steuern. So können Sie bestimmen, ob das Programm eine einzige Codezeile, ein gesamtes Unterprogramm (Prozedur oder Funktion) oder einen gesamten Programmblock ausführt. Wenn Sie manuell steuern, wann das Programm ausgeführt und wann es angehalten werden soll, können Sie die bekanntermaßen korrekt funktionierenden Bereiche schnell übergehen und Ihr Augenmerk auf die Bereiche lenken, die Probleme verursachen.

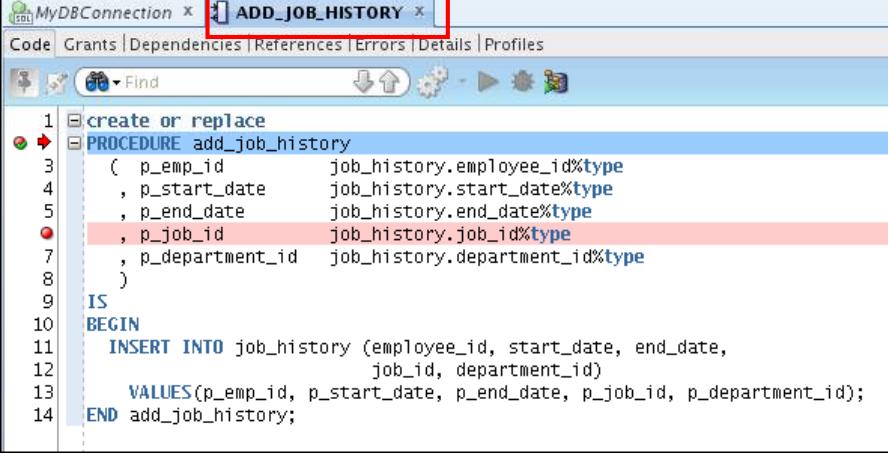
Unterprogramme debuggen – Überblick



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

ORACLE

Registerkarte zur Bearbeitung von Prozedur- oder Funktionscode



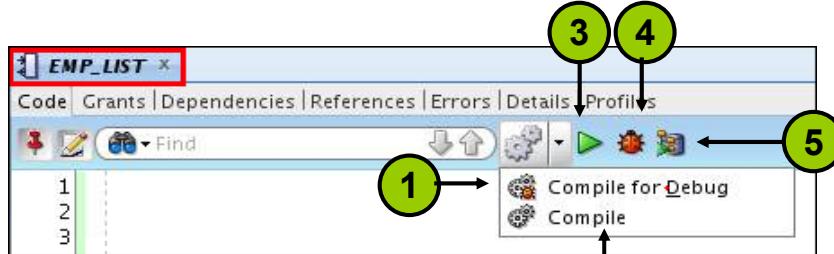
```
MyDBConnection x ADD_JOB_HISTORY x
Code Grants Dependencies References Errors Details Profiles
Find
1 create or replace
2 PROCEDURE add_job_history
3   ( p_emp_id          job_history.employee_id%type
4     , p_start_date    job_history.start_date%type
5     , p_end_date      job_history.end_date%type
6     , p_job_id        job_history.job_id%type
7     , p_department_id job_history.department_id%type
8   )
9   IS
10  BEGIN
11    INSERT INTO job_history (employee_id, start_date, end_date,
12                           job_id, department_id)
13      VALUES(p_emp_id, p_start_date, p_end_date, p_job_id, p_department_id);
14  END add_job_history;
```

ORACLE

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

In dieser Registerkarte wird neben einer Symbolleiste der bearbeitbare Text des Unterprogramms angezeigt. Sie können Breakpoints für das Debugging festlegen und aufheben. Klicken Sie dazu neben jeder Anweisung, der Sie einen Breakpoint zuordnen möchten, links neben die dünne, vertikale Linie. (Wenn ein Breakpoint festgelegt ist, wird ein roter Kreis angezeigt.)

Registerkarte für Prozeduren oder Funktionen – Symbolleiste

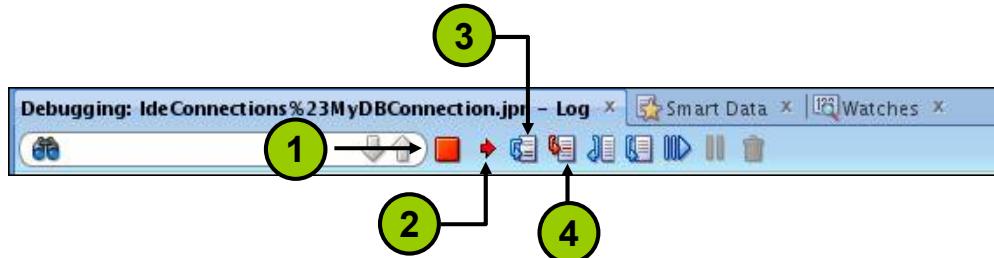


Symbol	Beschreibung
1. Compile for Debug	Kompliiert das Unterprogramm für das Debugging
2. Compile	Kompliiert das Unterprogramm
3. Run	Startet die normale Ausführung der Funktion oder Prozedur und zeigt die Ergebnisse in der Registerkarte "Running – Log" an
4. Debug	Führt das Unterprogramm im Debugmodus aus und zeigt die Registerkarte "Debugging – Log" mit der Debuggingsymbolleiste zum Steuern der Ausführung an
5. Profile	Zeigt das Fenster "Profile" an, mit dem Sie Parameterwerte zum Ausführen, Debuggen oder Profilieren einer PL/SQL-Funktion oder -Prozedur angeben

ORACLE

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

Registerkarte "Debugging – Log" – Symbolleiste



Symbol	Beschreibung
1. Terminate	Unterbricht und beendet die Ausführung
2. Find Execution Point	Geht zum nächsten Ausführungspunkt
3. Step Over	Übergeht das nächste Unterprogramm und geht zur nächsten Anweisung nach dem Unterprogramm
4. Step Into	Führt jeweils eine einzelne Programmanweisung aus. Wenn sich der Ausführungspunkt am Aufruf eines Unterprogramms befindet, geht der Befehl in die erste Anweisung dieses Unterprogramms hinein.

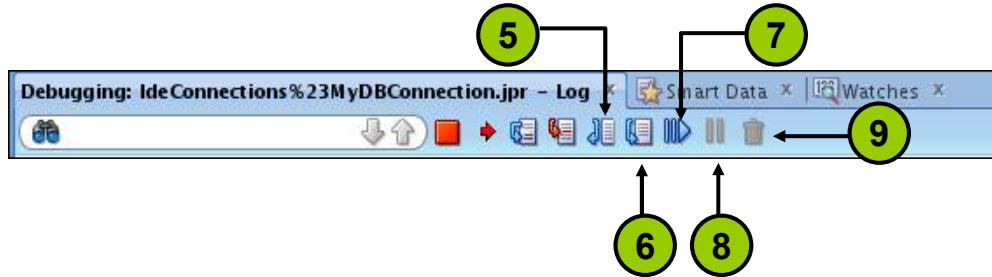
ORACLE

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

Die Registerkarte **Debugging – Log** enthält die Debuggingsymbolleiste und die Informationsmeldungen.

1. **Terminate:** Unterbricht und beendet die Ausführung. Sie können die Ausführung von diesem Punkt nicht wieder aufnehmen. Um die Ausführung oder das Debugging vom Anfang des Unterprogramms zu starten, klicken Sie in der Symbolleiste der Registerkarte **Source** auf das Symbol **Run** oder **Debug**.
2. **Find Execution Point:** Geht zum nächsten Ausführungspunkt (zur nächsten vom Debugger auszuführenden Quellcodezeile)
3. **Step Over:** Übergeht das nächste Unterprogramm (sofern es keinen Breakpoint aufweist) und geht zur nächsten Anweisung nach dem Unterprogramm. Falls sich der Ausführungspunkt an einem Unterprogrammaufruf befindet, führt **Step Over** dieses Unterprogramm ohne Unterbrechung aus und geht nicht in das Unterprogramm hinein. Der Ausführungspunkt wird dann an die Anweisung gesetzt, die dem Aufruf folgt. Befindet sich der Ausführungspunkt an der letzten Anweisung eines Unterprogramms, setzt **Step Over** den Ausführungspunkt an die Codezeile, die auf den Aufruf des Unterprogramms folgt.
4. **Step Into:** Führt jeweils eine einzelne Programmanweisung aus. Wenn sich der Ausführungspunkt am Aufruf eines Unterprogramms befindet, geht **Step Into** in dieses Unterprogramm hinein und setzt den Ausführungspunkt an die erste Anweisung. Befindet sich der Ausführungspunkt an der letzten Anweisung eines Unterprogramms, setzt **Step Into** den Ausführungspunkt an die Codezeile, die auf den Aufruf des Unterprogramms folgt.

Registerkarte "Debugging – Log" – Symbolleiste



Symbol	Beschreibung
5. Step Out	Verlässt das aktuelle Unterprogramm und geht zur nächsten Anweisung mit einem Breakpoint
6. Step to End of Method	Geht zur letzten Anweisung des aktuellen Unterprogramms
7. Resume	Setzt die Ausführung fort
8. Pause	Unterbricht die Ausführung, beendet sie jedoch nicht
9. Garbage Collect	Entfernt ungültige Objekte aus dem Cache

ORACLE

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

5. **Step Out:** Verlässt das aktuelle Unterprogramm und geht zur nächsten Anweisung
6. **Step to End of Method:** Geht zur letzten Anweisung des aktuellen Unterprogramms
7. **Resume:** Setzt die Ausführung fort
8. **Pause:** Unterbricht die Ausführung, beendet sie jedoch nicht. Sie können die Ausführung wieder aufnehmen.
9. **Garbage Collect:** Entfernt ungültige Objekte aus dem Cache, um Platz für gültige Objekte zu schaffen, auf die häufiger zugegriffen wird

Weitere Registerkarten

Debugging - Log	Breakpoints	Smart Data	Data	Watches
Name	Value	Type		
P_MAXROWS	100	NUMBER		
REC_EMP		Rowtype		
EMP_TAB	indexed table	EMP_TAB_TYPE		
I	1	NUMBER		
V_CITY	NULL	VARCHAR2(30)		

Registerkarte	Beschreibung
Breakpoints	Zeigt system- und benutzerdefinierte Breakpoints an
Smart Data	Zeigt Informationen zu Variablen an. Sie können diese Voreinstellungen festlegen, indem Sie mit der rechten Maustaste auf das Fenster "Smart Data" klicken und "Preferences" wählen.
Data	Befindet sich unter dem Codetextbereich; zeigt Informationen zu allen Variablen an
Watches	Befindet sich unter dem Codetextbereich; zeigt Informationen zu Überwachungspunkten an

ORACLE

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

Überwachungsausdrücke festlegen

Mithilfe von Überwachungsausdrücken können Sie während der Programmausführung veränderliche Werte von Variablen oder Ausdrücken verfolgen. Nachdem Sie einen Überwachungsausdruck eingegeben haben, wird im Fenster **Watches** der aktuelle Wert des Ausdrucks angezeigt. Im Laufe der Programmausführung ändert sich der Wert des Überwachungsausdrucks, da das Programm die Werte der Variablen im Überwachungsausdruck aktualisiert.

Ein Überwachungsausdruck wertet einen Ausdruck auf Grundlage des aktuellen Kontexts aus, der durch die Auswahl im Fenster **Stack** gesteuert wird. Wenn Sie zu einem neuen Kontext gehen, wird der Ausdruck für diesen erneut ausgewertet. Wird der Ausführungspunkt an eine Stelle verschoben, an der eine Variable des Überwachungsausdruck nicht definiert ist, gilt der gesamte Überwachungsausdruck als nicht definiert. Kehrt der Ausführungspunkt zu einer Stelle zurück, an der der Überwachungsausdruck ausgewertet werden kann, ist im Fenster **Watches** wieder der Wert des Überwachungsausdrucks zu sehen.

Zum Öffnen des Fensters **Watches** klicken Sie nacheinander auf **View**, **Debugger** und **Watches**.

Um einen Überwachungsausdruck hinzuzufügen, klicken Sie mit der rechten Maustaste auf das Fenster **Watches** und wählen **Add Watch**. Um einen Überwachungsausdruck zu bearbeiten, klicken Sie mit der rechten Maustaste auf das Fenster **Watches** und wählen **Edit Watch**.

Hinweis: Falls einige der in dieser Lektion beschriebenen Debuggingregisterkarten nicht sichtbar sind, können Sie sie mit **View > Debugger** erneut anzeigen.

Prozeduren debuggen – Beispiel: Neue Prozedur emp_list erstellen

```

1 CREATE OR REPLACE PROCEDURE emp_list(pmaxrows IN NUMBER) AS
2 CURSOR emp_cursor IS
3 SELECT d.department_name,
4       e.employee_id,
5       e.last_name,
6       e.salary,
7       e.commission_pct
8 FROM departments d,
9      employees e
10 WHERE d.department_id = e.department_id;
11 emp_record emp_cursor % rowtype;
12 type emp_tab_type IS TABLE OF emp_cursor % rowtype INDEX BY binary_integer;
13 emp_tab emp_tab_type;
14 i NUMBER := 1;
15 v_city VARCHAR2(30);
16 BEGIN
17
18   OPEN emp_cursor;
19   FETCH emp_cursor
20   INTO emp_record;
21   emp_tab(i) := emp_record;
22   WHILE (emp_cursor % FOUND)
23     AND (i <= pmaxrows)
24   LOOP
25     i := i + 1;
26     FETCH emp_cursor
27     INTO emp_record;
28     emp_tab(i) := emp_record;
29     v_city := get_location(emp_record.department_name);
30     DBMS_OUTPUT.PUT_LINE('Employee ' || emp_record.last_name || ' works in ' || v_city);
31   END LOOP;
32
33   CLOSE emp_cursor;
34   FOR j IN REVERSE 1 .. i
35   LOOP
36     DBMS_OUTPUT.PUT_LINE(emp_tab(j).last_name);
37   END LOOP;
38 END emp_list;

```

ORACLE

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

Die Prozedur `emp_list` ruft Informationen wie den Abteilungsnamen, die Personalnummer, den Namen, das Gehalt und den Provisionsprozentsatz des Mitarbeiters ab. Die Prozedur erstellt einen Record, um die Informationen des Mitarbeiters zu speichern, sowie eine Tabelle, die mehrere Records von Mitarbeitern aufnehmen kann. "i" ist ein Zähler.

Der Code öffnet den Cursor und ruft die Mitarbeiter-Records ab. Darüber hinaus prüft der Code, ob noch mehr Records abzurufen sind und ob die Anzahl der bisher abgerufenen Records niedriger ist als die von Ihnen angegebene Anzahl von Records. Schließlich gibt der Code die Mitarbeiterinformationen aus. Außerdem ruft die Prozedur die Funktion `get_location` auf, die den Namen der Stadt zurückgibt, in der ein Mitarbeiter arbeitet.

Hinweis: Stellen Sie sicher, dass Sie den Prozedurcode im Bearbeitungsmodus anzeigen. Um den Prozedurcode zu bearbeiten, klicken Sie in der Symbolleiste der Prozedur auf das Symbol **Edit**.

Prozeduren debuggen – Beispiel: Neue Funktion get_location erstellen

```
1 | CREATE OR REPLACE FUNCTION get_location(p_deptname IN VARCHAR2) RETURN VARCHAR2 AS
2 |   v_loc_id NUMBER;
3 |   v_city VARCHAR2(30);
4 | BEGIN
5 |   SELECT d.location_id,
6 |         l.city
7 |   INTO v_loc_id,
8 |         v_city
9 |   FROM departments d,
10 |        locations l
11 |  WHERE UPPER(department_name) = UPPER(p_deptname)
12 |    AND d.location_id = l.location_id;
13 |  RETURN v_city;
14 | END get_location;
```

ORACLE

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

Prozeduren debuggen – Beispiel: Neue Prozeduren erstellen

Diese Funktion gibt die Stadt zurück, in der ein Mitarbeiter arbeitet. Sie wird aus der Prozedur emp_list aufgerufen.

Breakpoints festlegen und emp_list für Debugmodus komplizieren

The screenshot shows the Oracle SQL Developer interface. In the top tab bar, 'MyDBConnection' and 'EMP_LIST' are visible. The main area is the PL/SQL Editor with the following code:

```

1  create or replace
2  PROCEDURE emp_list
3  (p_maxrows IN NUMBER)
4  IS
5    CURSOR cur_emp IS
6      SELECT d.department_name, e.employee_id, e.last_name,
7             e.salary, e.commission_pct
8      FROM departments d, employees e
9     WHERE d.department_id = e.department_id;
10   rec_emp cur_emp%ROWTYPE;
11   TYPE emp_tab_type IS TABLE OF cur_emp%ROWTYPE INDEX BY BINARY_INTEGER;
12   emp_tab emp_tab_type;
13   i NUMBER := 1;
14   v_city VARCHAR2(30);
15  BEGIN
16    OPEN cur_emp;
17    FETCH cur_emp INTO rec_emp;
18    emp_tab(i) := rec_emp;
19    WHILE (cur_emp%FOUND) AND (i <= p_maxrows) LOOP
20      i := i + 1;
21      FETCH cur_emp INTO rec_emp;
22      emp_tab(i) := rec_emp;
23      v_city := get_location (rec_emp.department_name);
24      dbms_output.put_line('Employee ' || rec_emp.last_name || 
25                           ' works in ' || v_city );
26    END LOOP;
27    CLOSE cur_emp;
28    FOR j IN REVERSE 1..i LOOP
29      DBMS_OUTPUT.PUT_LINE(emp_tab(j).last_name);
30    END LOOP;
31  END emp_list;

```

Four red boxes highlight specific parts of the code: the procedure header 'PROCEDURE emp_list', the cursor declaration 'CURSOR cur_emp IS', the loop condition 'WHILE (cur_emp%FOUND) AND (i <= p_maxrows) LOOP', and the assignment statement 'v_city := get_location (rec_emp.department_name);'. A fifth red box highlights the 'Messages - Log' tab at the bottom.

ORACLE

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

Im Beispiel auf der Folie wird die Prozedur `emp_list` im Bearbeitungsmodus angezeigt. Vier Breakpoints wurden an verschiedenen Stellen im Code hinzugefügt. Um die Prozedur für das Debugging zu komplizieren, klicken Sie mit der rechten Maustaste auf den Code und wählen dann im Kontextmenü die Option **Compile for Debug**. Die Registerkarte **Messages - Log** zeigt in einer Meldung an, dass die Prozedur kompiliert wurde.

Funktion get_location für Debugmodus kompilieren

The screenshot shows the Oracle SQL Developer interface. A PL/SQL function named 'get_location' is displayed in the code editor. The function takes a department name as input and returns the city associated with that department. The code is highlighted in blue and black. A red box highlights the tab bar at the top where 'GET_LOCATION' is selected. Another red box highlights the toolbar above the code editor, specifically the 'Compile for Debug' button, which is highlighted with a cursor. A third red box highlights the 'Messages - Log' tab at the bottom, which displays the message 'Compiled'.

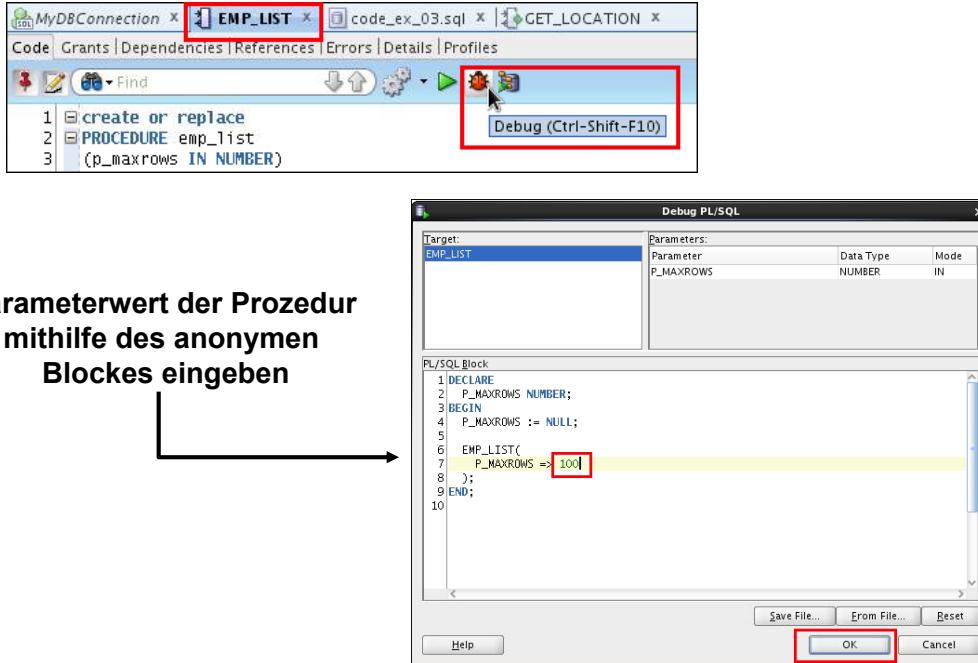
```
1 create or replace
2 FUNCTION get_location
3   ( p_deptname IN VARCHAR2) RETURN VARCHAR2;
4   AS
5     v_loc_id NUMBER;
6     v_city  VARCHAR2(30);
7   BEGIN
8     SELECT d.location_id, l.city INTO v_loc_id, v_city
9     FROM departments d, locations l
10    WHERE upper(department_name) = upper(p_deptname)
11      and d.location_id = l.location_id;
12    RETURN v_city;
13  END GET_LOCATION;
```

ORACLE

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

Im Beispiel auf der Folie wird die Funktion `get_location` im Bearbeitungsmodus angezeigt. Um die Funktion für das Debugging zu kompilieren, klicken Sie mit der rechten Maustaste auf den Code und wählen dann im Kontextmenü die Option **Compile for Debug**. Die Registerkarte **Messages - Log** zeigt in einer Meldung an, dass die Funktion kompiliert wurde.

emp_list debuggen und Werte für Parameter PMAXROWS eingeben



Parameterwert der Prozedur mithilfe des anonymen Blockes eingeben

Als nächsten Schritt im Debuggingprozess debuggen Sie die Prozedur mit einer der verschiedenen bereits erwähnten verfügbaren Methoden, beispielsweise durch Klicken auf das Symbol **Debug** in der Symbolleiste der Prozedur. Es wird ein alterer Block angezeigt, in dem Sie zur Eingabe der Parameter für diese Prozedur aufgefordert werden. `emp_list` verfügt über einen Parameter `PMAXROWS`, der die Anzahl der zurückzugebenden Records angibt. Ersetzen Sie den zweiten Parameter `PMAXROWS` durch eine Zahl wie 100, und klicken Sie dann auf **OK**.

emp_list debuggen – In Code gehen (F7)

Programmsteuerung hält am ersten Breakpoint an.

```

1 CREATE OR REPLACE PROCEDURE emp_list
2 (p_maxrows IN NUMBER)
3 IS
4 CURSOR cur_emp IS
5   SELECT d.department_name, e.employee_id, e.last_name,
6         e.salary, e.commission_pct
7   FROM departments d, employees e
8   WHERE d.department_id = e.department_id;
9   TYPE emp_tab_type IS TABLE OF cur_emp%ROWTYPE INDEX BY BINARY_INTEGER;
10  emp_tab emp_tab_type;
11  i NUMBER := 1;
12  v_city VARCHAR2(30);
13
14 BEGIN
15   OPEN cur_emp;
16   FETCH cur_emp INTO rec_emp;
17   emp_tab(i) := rec_emp;
18   WHILE (cur_emp%FOUND) AND (i <= p_maxrows) LOOP
19     i := i + 1;
20     FETCH cur_emp INTO rec_emp;
21     emp_tab(i) := rec_emp;
22     v_city := get_location(rec_emp.department_name);
23     dbms_output.put_line('Employee ' || rec_emp.last_name ||
24                           ' works in ' || v_city );
25   END LOOP;
26   CLOSE cur_emp;
27   FOR j IN REVERSE 1..1 LOOP
28     DBMS_OUTPUT.PUT_LINE(emp_tab(j).last_name);
29   END LOOP;
30 END emp_list;
31

```

Debugging: IdeConnections%23MyDBConnection.jpr - Log Smart Data Watches

Connecting to the database MyDBConnection.
Executing PL/SQL: ALTER SESSION SET PLSQL_DEBUG=TRUE
Executing PL/SQL: CALL DBMS_DEBUG_JDWP.CONNECT_TCP('127.0.0.1', '15290')
Debugger accepted connection from database on port 15290.
Source breakpoint occurred at line 16 of EMP_LIST.p1s.

ORACLE

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

Debuggingtool "Step Into"

Der Befehl **Step Into** führt die Programmanweisungen einzeln nacheinander aus. Wenn sich der Ausführungspunkt am Aufruf eines Unterprogramms befindet, geht der Befehl **Step Into** in dieses Unterprogramm hinein und setzt den Ausführungspunkt an die erste Anweisung des Unterprogramms. Wenn sich der Ausführungspunkt bei Auswahl von **Step Into** an der letzten Anweisung eines Unterprogramms befindet, kehrt der Debugger aus dem Unterprogramm zurück und setzt den Ausführungspunkt an die Codezeile, die dem Aufruf des Unterprogramms folgt. Der Begriff *Single Stepping* bezieht sich auf die Verwendung des Befehls **Step Into**, mit dem die Anweisungen im Programmcode nacheinander abgearbeitet werden. Sie können mit einer der folgenden Methoden in ein Unterprogramm hineingehen: Befehlsfolge **Debug > Step Into**, Funktionstaste F7 oder Symbol **Step Into** in der Symbolleiste von **Debugging – Log**.

Im Beispiel auf der Folie wird die Programmsteuerung am ersten Breakpoint im Code gestoppt. Der Pfeil neben dem Breakpoint zeigt an, dass dies die Codezeile ist, die als Nächstes ausgeführt wird. Beachten Sie die verschiedenen Registerkarten unten im Fenster.

Hinweis: Die Befehle **Step Into** und **Step Over** sind das einfachste Verfahren zum schrittweisen Durchlaufen von Programmcode. Die beiden Befehle sind zwar sehr ähnlich, bieten jedoch ein unterschiedliches Verfahren zur Steuerung der Codeausführung.

emp_list debuggen – In Code gehen (F7)

The screenshot shows the Oracle SQL Developer interface with the 'emp_list' procedure open. A red arrow points from the explanatory text to the cursor code in the main pane. A green circle labeled '1' highlights the first line of the cursor. A green circle labeled '2' highlights the 'OPEN cur_emp;' line. A green circle labeled '3' highlights the cursor select statement.

```

create or replace
PROCEDURE emp_list
(p_maxrows IN NUMBER)
IS
CURSOR cur
SELECT d.department_name, e.employee_id, e.last_name,
       e.salary, e.commission_pct
  FROM departments d, employees e
 WHERE d.department_id = e.department_id;
rec_emp cur%ROWTYPE;
TYPE emp_tab_type IS TABLE OF cur%ROWTYPE INDEX BY BINARY_INTEGER;
emp_tab emp_tab_type;
i NUMBER := 1;
v_city VARCHAR2(30);
BEGIN
  OPEN cur_emp;
  FETCH cur_emp INTO rec_emp;
  emp_tab(i) := rec_emp;
  WHILE (cur_emp%FOUND) AND (i <= p_maxrows) LOOP
    i := i + 1;
    FETCH cur_emp INTO rec_emp;
    emp_tab(i) := rec_emp;
    v_city := get_location (rec_emp.department_name);
    dbms_output.put_line('Employee ' || rec_emp.last_name ||
      ' works in ' || v_city );
  END LOOP;
  CLOSE cur_emp;
  FOR j IN REVERSE 1..i LOOP
    DBMS_OUTPUT.PUT_LINE(emp_tab(j).last_name);
  END LOOP;
END emp_list;
  
```

Debugging: IdeConnections%23MyDBConnection.jpr - Log x Breakpoints x Smart Data x Data

Connecting to the database HyDBConnection.
Executing PL/SQL: ALTER SESSION SET PLSQL_DEBUG=TRUE
Executing PL/SQL: CALL DBMS_DEBUG_JDWP.CONNECT_TCP('127.0.0.1', '63138')
Debugger accepted connection from database on port 63138.
source breakpoint occurred at line 16 of EMP_LIST.pls.

"Step Into" (F7):
Geht in den
Cursorcode und
führt ihn aus

emp_list debuggen – In Code gehen

Mit dem Befehl **Step Into** werden die Programmanweisungen einzeln nacheinander ausgeführt. Wenn sich der Ausführungspunkt am Aufruf eines Unterprogramms befindet, geht der Befehl **Step Into** in dieses Unterprogramm hinein und setzt die Ausführung an die erste Anweisung des Unterprogramms.

Im Beispiel auf der Folie wird durch Drücken von F7 die Codezeile am ersten Breakpoint ausgeführt.

Daten anzeigen

The screenshot illustrates the Oracle SQL Developer interface during a debug session. It shows two code snippets and their corresponding variable states in the Data tab.

Code Snippet 1:

```

18 OPEN emp_cursor;
19
20 FETCH emp_cursor

```

Code Snippet 2:

```

16 OPEN cur_emp;
17   FETCH cur_emp INTO rec_emp;
18   emp_tab(i) := rec_emp;
19 WHILE (cur_emp%FOUND) AND (i <= p_maxrows) LOOP

```

Data Tab Content (Step 18):

Name	Value	Type
P_MAXROWS	100	NUMBER
REC_EMP	Rowtype	
DEPARTMENT_NAME	NULL	VARCHAR2(30)
EMPLOYEE_ID	NULL	NUMBER(6,0)
LAST_NAME	NULL	VARCHAR2(25)
SALARY	NULL	NUMBER(8,2)
COMMISSION_PCT	NULL	NUMBER(2,2)
EMP_TAB	Indexed table	
_index	EMP_TAB_TYPE	EMP_TAB_TYPE elemen...
_values	1	NUMBER
I	NULL	VARCHAR2(30)
V_CITY		

Data Tab Content (Step 19):

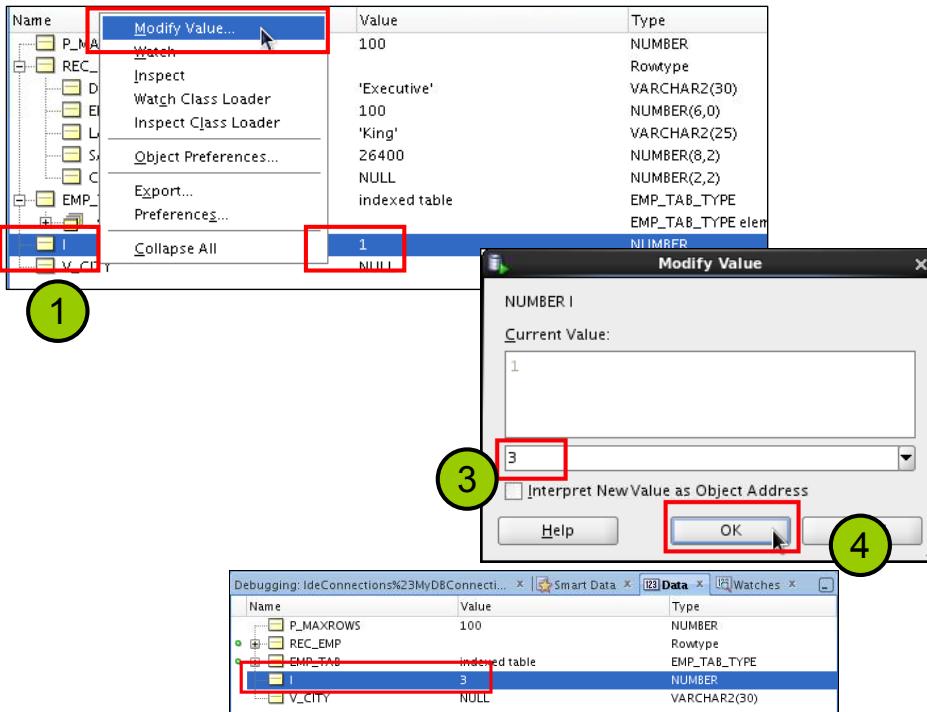
Name	Value	Type
P_MAXROWS	100	NUMBER
REC_EMP	Rowtype	
DEPARTMENT_NAME	'Executive'	VARCHAR2(30)
EMPLOYEE_ID	100	NUMBER(6,0)
LAST_NAME	'King'	VARCHAR2(25)
SALARY	26400	NUMBER(8,2)
COMMISSION_PCT	NULL	NUMBER(2,2)
EMP_TAB	Indexed table	
_index	EMP_TAB_TYPE	EMP_TAB_TYPE elemen...
_values	1	NUMBER
I	NULL	VARCHAR2(30)
V_CITY		

ORACLE

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

Beim Debuggen des Codes können Sie in der Registerkarte **Data** die Variablen anzeigen und ändern. Es lassen sich auch Überwachungspunkte festlegen, um eine Teilmenge der in der Registerkarte **Data** angezeigten Variablen zu überwachen. Um die Registerkarten **Data**, **Smart Data** und **Watch** ein- oder auszublenden, wählen Sie **View > Debugger** und dann die ein- oder auszublenden Registerkarten.

Variablen beim Debugging des Codes ändern



ORACLE

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

Um den Wert einer Variablen in der Registerkarte **Data** zu ändern, klicken Sie mit der rechten Maustaste auf den Variablennamen und wählen dann im Kontextmenü die Option **Modify Value**. Das Fenster **Modify Value** wird angezeigt. Der aktuelle Wert für die Variable wird angezeigt. Sie können im zweiten Textfeld einen neuen Wert eingeben und dann auf **OK** klicken.

emp_list debuggen – Code übergehen

```

14 BEGIN
15   OPEN cur_emp;
16   FETCH cur_emp INTO rec_emp;
17   emp_tab(i) := rec_emp;
18   WHILE (cur_emp%FOUND) AND (i <= p_maxrows) LOOP
19     i := i + 1;
20     FETCH cur_emp INTO rec_emp;
21     emp_tab(i) := rec_emp;
22     v_city := get_location (rec_emp.department_name);
23     dbms_output.put_line('Employee ' || rec_emp.last_name ||
24                               ' works in ' || v_city );

```

1 F8

```

14 BEGIN
15   OPEN cur_emp;
16   FETCH cur_emp INTO rec_emp;
17   emp_tab(i) := rec_emp;
18   WHILE (cur_emp%FOUND) AND (i <= p_maxrows) LOOP
19     i := i + 1;
20     FETCH cur_emp INTO rec_emp;
21     emp_tab(i) := rec_emp;
22     v_city := get_location (rec_emp.department_name);
23     dbms_output.put_line('Employee ' || rec_emp.last_name ||
24                               ' works in ' || v_city );

```

2 F8

```

14 BEGIN
15   OPEN cur_emp;
16   FETCH cur_emp INTO rec_emp;
17   emp_tab(i) := rec_emp;
18   WHILE (cur_emp%FOUND) AND (i <= p_maxrows) LOOP
19     i := i + 1;
20     FETCH cur_emp INTO rec_emp;
21     emp_tab(i) := rec_emp;
22     v_city := get_location (rec_emp.department_name);
23     dbms_output.put_line('Employee ' || rec_emp.last_name ||
24                               ' works in ' || v_city );

```

3 F8

"Step Over" (F8):
Führt den Cursor aus
(wie F7), die Steuerung
wird jedoch nicht
an Open Cursor-
Code übergeben.

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

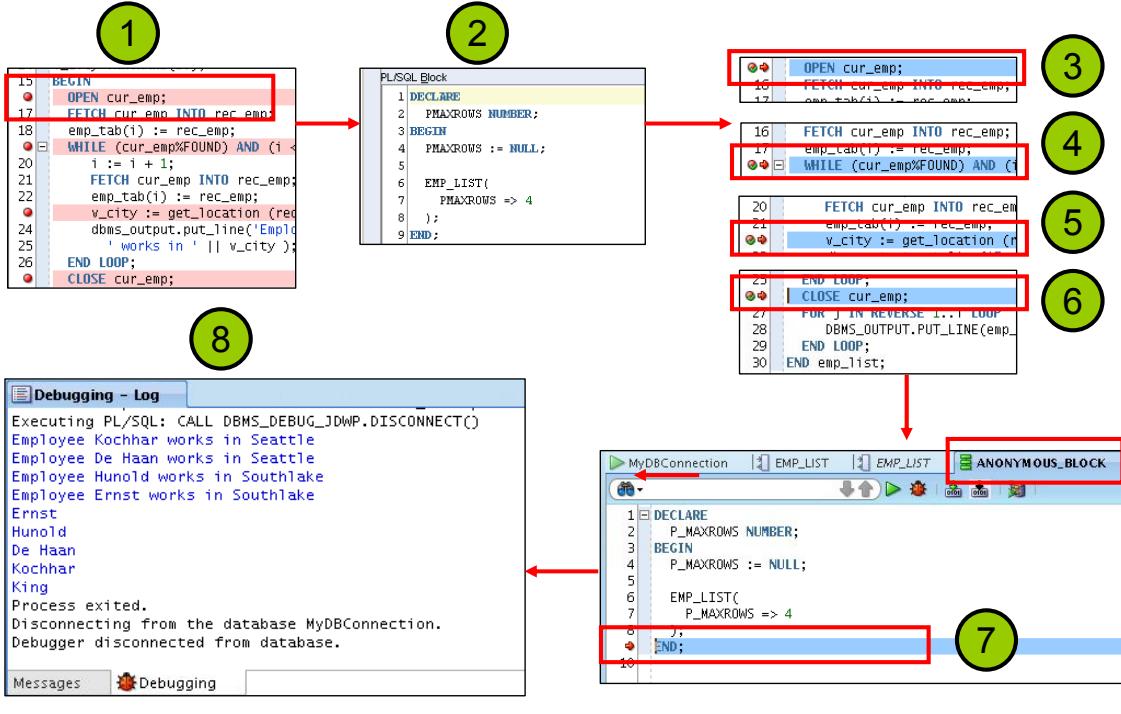
ORACLE

Debuggingtool "Step Over"

Mit dem Befehl **Step Over** werden wie mit **Step Into** die Programmanweisungen einzeln nacheinander ausgeführt. Wenn Sie den Befehl **Step Over** absetzen und sich der Ausführungspunkt an einem Unterprogrammaufruf befindet, geht der Debugger jedoch nicht in dieses Unterprogramm hinein, sondern führt es ohne Unterbrechung aus. Anschließend setzt er den Ausführungspunkt an die Anweisung, die dem Unterprogrammaufruf folgt. Wenn sich der Ausführungspunkt an der letzten Anweisung eines Unterprogramms befindet und **Step Over** gewählt wird, kehrt der Debugger aus dem Unterprogramm zurück und setzt den Ausführungspunkt an die Codezeile, die dem Aufruf des Unterprogramms folgt. Ein Unterprogramm können Sie mit einer der folgenden Methoden übergehen: Befehlsfolge **Debug > Step Over**, Funktionstaste F8 oder Symbol **Step Over** in der Symboleiste von **Debugging - Log**.

Im Beispiel auf der Folie wird die geöffnete Cursorzeile ausgeführt, ohne dass die Programmsteuerung an die Cursordefinition übergeben wird, wie dies beim Beispiel für die Option **Step Into** der Fall war.

emp_list debuggen – Code verlassen (UMSCHALT+F7)



ORACLE

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

Die Option **Step Out** verlässt das aktuelle Unterprogramm und geht zum Unterprogramm der nächsten Anweisung. Wenn Sie im Beispiel auf der Folie (beginnend beim 3. Schritt) UMSCHALT+F7 drücken, geht die Programmsteuerung zum nächsten Breakpoint. Wenn Sie im 6. Schritt UMSCHALT + F7 drücken, verlässt die Programmsteuerung die Prozedur und geht zur nächsten Anweisung im anonymen Block.

emp_list debuggen – Code bis Cursor ausführen (F4)

```

10    emp_record emp_cursor%ROWTYPE;
11    TYPE emp_tab_type IS TABLE OF emp_cursor%ROWTYPE INDEX BY BINARY_INTEGER;
12    emp_tab emp_tab_type;
13    i NUMBER := 1;
14    v_city VARCHAR2(30);
15
16    BEGIN
17        OPEN emp_cursor;
18        FETCH emp_cursor INTO emp_record;
19        emp_tab(i) := emp_record;
20        WHILE (emp_cursor%FOUND) AND (i <= pMaxRows) LOOP
21            i := i + 1;
22            FETCH emp_cursor INTO emp_record;
23            emp_tab(i) := emp_record;
24            v_city := get_location (emp_record.department_name);
25            dbms_output.put_line('Employee ' || emp_record.last_name ||
26                ' works in ' || v_city );
27        END LOOP;
28        CLOSE emp_cursor;
29        FOR j IN REVERSE 1..i LOOP
30            DBMS_OUTPUT.PUT_LINE(emp_tab(j).last_name);
31        END LOOP;
32    END emp_list;

```

Connecting to the database debugging.
Executing PL/SQL: ALTER SESSION SET PLSQL_DEBUG=TRUE
Executing PL/SQL: CALL DEMS_DEBUG_JDWP.CONNECT_TCP('127.0.0.1:1521')
Debugger accepted connection from database on port 1521
Processing 59 classes that have already been prepared..
Finished processing prepared classes.
Source breakpoint occurred at line 16 of EMP_LIST.pls.

**"Run to Cursor" (F4):
Ausführung bis zur
Cursorposition
ohne Einzelschritte
oder Festlegen
eines Breakpoints**

Name	Value	Type
PMAXROWS	5	NUMBER
EMP_RECORD	Rowtype	
DEPARTMENT_NAME	'Administration'	VARCHAR2(14)
EMPLOYEE_ID	200	NUMBER(6,0)
LAST_NAME	'Vhalen'	VARCHAR2(12)
SALARY	4400	NUMBER(8,2)
COMMISSION_PCT	NULL	NUMBER(2,2)
EMP_TAB	indexed table	EMP_TAB...
I	1	NUMBER
V_CITY	NULL	VARCHAR2(30)

ORACLE

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

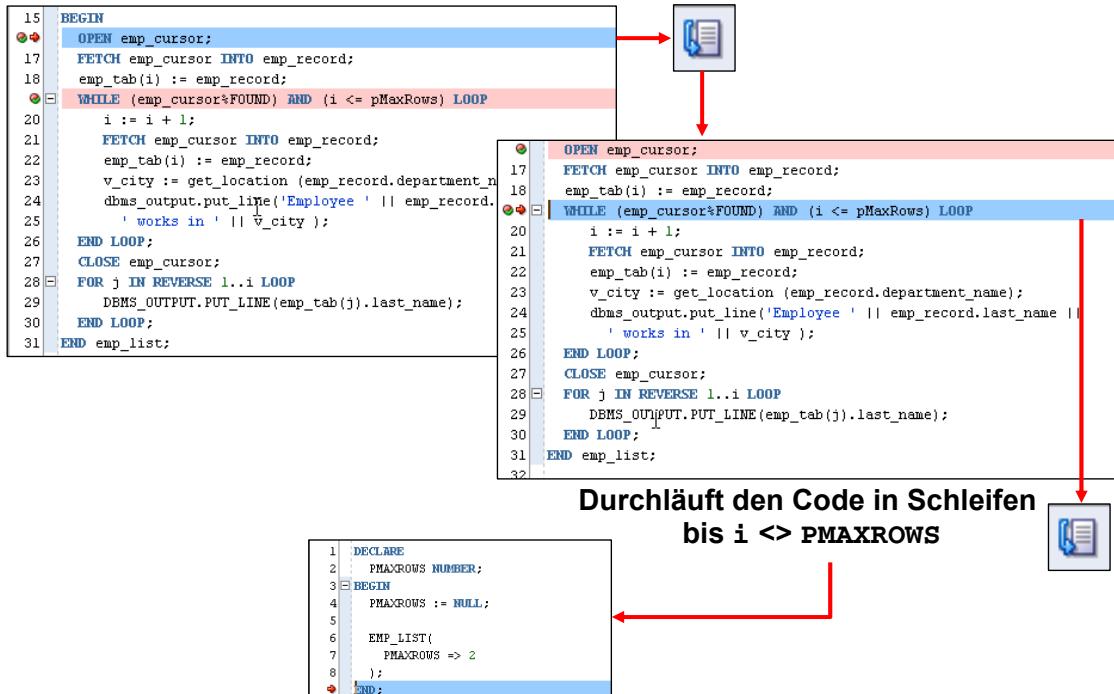
Code bis zur Cursorposition ausführen

Beim Durchlaufen des Anwendungscodes im Debugger haben Sie die Möglichkeit, den Code bis zu einer bestimmten Programmposition durchgehend und ohne Einzelschritte oder Breakpoints auszuführen. Setzen Sie hierfür den Textcursor in einem Unterprogrammeditor in die Codezeile, in der der Debugger anhalten soll. Die Codeausführung bis zur Cursorposition können Sie mit einer der folgenden Methoden starten: Klicken Sie im Procedure Editor mit der rechten Maustaste, und wählen Sie **Run to Cursor**. Wählen Sie im Hauptmenü die Option **Debug > Run to Cursor**, oder drücken Sie F4.

Die folgenden Ergebnisse sind möglich:

- Das Programm wird ohne Unterbrechung ausgeführt, bis die vom Textcursor im Source Editor markierte Position erreicht ist.
- Der Textcursor befindet sich an einer Codezeile, die das Programm niemals ausführt. In diesem Fall wird das Programm bis zum nächsten Breakpoint oder bis zum Ende ausgeführt.

emp_list debuggen – Zum Methodenende gehen



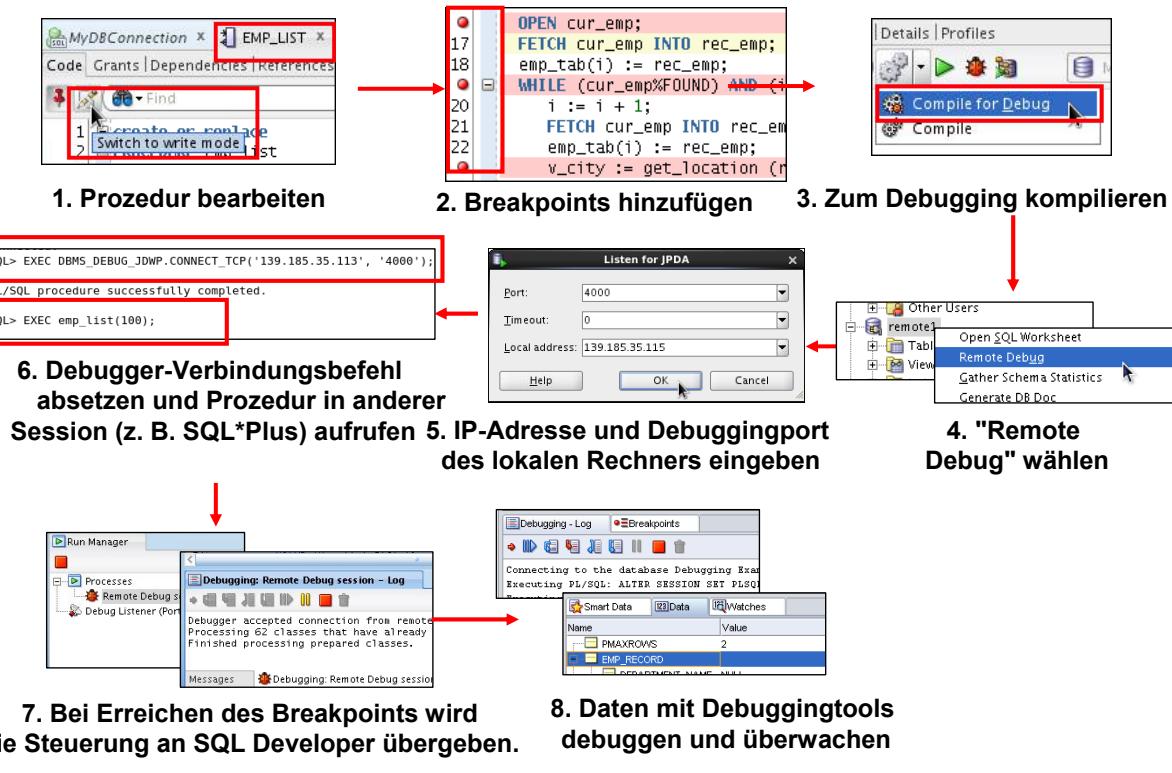
ORACLE

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

Step to End of Method geht zur letzten Anweisung im aktuellen Unterprogramm oder zum nächsten Breakpoint, falls im aktuellen Unterprogramm vorhanden.

Im Beispiel auf der Folie wird das Debuggingtool **Step to End of Method** verwendet. Da ein zweiter Breakpoint vorhanden ist, wird bei Wahl von **Step to End of Method** die Steuerung an diesen Breakpoint übertragen. Wenn Sie **Step to End of Method** erneut wählen, werden zuerst die Iterationen der WHILE-Schleife durchlaufen. Dann wird die Programmsteuerung an die nächste ausführbare Anweisung im anonymen Block übertragen.

Unterprogramme remote debuggen – Überblick



ORACLE

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

Remote-Debugging

Mit Remote-Debugging können Sie PL/SQL-Code in jeder Datenbank unabhängig vom Speicherort aufrufen und debuggen. Sie benötigen lediglich Zugriff auf die Datenbank und müssen eine Verbindung herstellen. Beim Remote-Debugging debuggen Sie eine Prozedur, die nicht von Ihnen als Entwickler, sondern von anderer Seite gestartet wird. Remote-Debugging und lokales Debugging haben viele Schritte gemeinsam. Gehen Sie beim Remote-Debugging wie folgt vor:

1. Bearbeiten Sie das Unterprogramm, das Sie debuggen möchten.
2. Fügen Sie dem Unterprogramm die Breakpoints hinzu.
3. Klicken Sie in der Symbolleiste auf das Symbol **Compile for Debug**.
4. Klicken Sie mit der rechten Maustaste auf die Verbindung zur Remote-Datenbank, und wählen Sie **Remote Debug**.
5. Geben Sie im Dialogfeld **Debugger - Attach to JPA** die IP-Adresse und die Portnummer des lokalen Rechners ein, beispielsweise 4000, und klicken Sie dann auf **OK**.
6. Setzen Sie den Debugger-Verbindungsbefehl in einer anderen Session (z. B. SQL*Plus) ab, und rufen Sie dann die Prozedur in dieser Session auf.
7. Beim Erreichen eines Breakpoints wird die Steuerung an die ursprüngliche SQL Developer-Session zurückgegeben und die Debugger-Symbolleiste angezeigt.
8. Debuggen Sie das Unterprogramm, und überwachen Sie die Daten. Verwenden Sie dafür die bereits besprochenen Debuggingtools.

Übung 2 zu Lektion 3 – Überblick: SQL Developer-Debugger – Einführung

Diese Übungen behandeln folgende Themen:

- **Prozedur und Funktion erstellen**
- **Breakpoints in die Prozedur einfügen**
- **Prozedur und Funktion für den Debugmodus kompilieren**
- **Prozedur debuggen und in Code gehen**
- **Variablen des Unterprogramms anzeigen und ändern**

ORACLE

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

Übung 2 zu Lektion 3 – Überblick

Für diese Übung müssen Sie SQL Developer verwenden. Sie erhalten eine Einführung in die grundlegende Funktionalität des SQL Developer-Debuggers.

Zusammenfassung

In dieser Lektion haben Sie Folgendes gelernt:

- **Prozeduren und Funktionen unterscheiden**
- **Verwendungsmöglichkeiten von Funktionen beschreiben**
- **Stored Functions erstellen**
- **Funktionen aufrufen**
- **Funktionen entfernen**
- **Grundlegende Funktionalität des SQL Developer-Debuggers kennenlernen**



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

Eine Funktion ist ein benannter PL/SQL-Block, der einen Wert zurückgeben muss. Im Allgemeinen erstellen Sie Funktionen, um einen Wert zu berechnen und zurückzugeben, und Prozeduren, um eine Aktion durchzuführen.

Funktionen können erstellt oder gelöscht werden.

Funktionen werden als Teil eines Ausdrucks aufgerufen.

Packages erstellen

4

ORACLE

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

Ziele

Nach Ablauf dieser Lektion haben Sie folgende Ziele erreicht:

- **Packages beschreiben und ihre Komponenten auflisten**
- **Packages erstellen, um zusammengehörige Variablen, Cursor, Konstanten, Exceptions, Prozeduren und Funktionen zu gruppieren**
- **Packagekonstrukte als öffentlich oder privat angeben**
- **Packagekonstrukte aufrufen**
- **Verwendungsmöglichkeiten von Packages ohne Body beschreiben**

ORACLE

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

In dieser Lektion wird beschrieben, was ein Package ist und aus welchen Komponenten es besteht. Außerdem lernen Sie, Packages zu erstellen und zu verwenden.

Lektionsagenda

- **Vorteile und Komponenten von Packages bestimmen**
- **Mit Packages arbeiten:**
 - Packagespezifikationen und Packagebodys erstellen
 - Packageunterprogramme aufrufen
 - Packageinformationen anzeigen
 - Packages entfernen

ORACLE

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

Was sind PL/SQL-Packages?

- Ein Package ist ein Schemaobjekt, das logisch zusammengehörige PL/SQL-Typen, Variablen und Unterprogramme gruppiert.
- Packages bestehen meist aus zwei Teilen:
 - Spezifikation
 - Body
- Die Spezifikation ist die Schnittstelle zum Package. In ihr sind die Typen, Variablen, Konstanten, Exceptions, Cursor und Unterprogramme deklariert, die von außerhalb des Packages referenziert werden können.
- Im Body sind die Abfragen für die Cursor und der Code für die Unterprogramme definiert.
- Mithilfe von Packages kann der Oracle-Server mehrere Objekte gleichzeitig in den Speicher lesen.

ORACLE

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

PL/SQL-Packages – Überblick

Mit PL/SQL-Packages können Sie zusammengehörige PL/SQL-Typen, Variablen, Datenstrukturen, Exceptions und Unterprogramme in einem Container bündeln. Beispiel: Ein Human Resource-Package kann Einstellungs- und Kündigungsprozeduren, Provisions- und Prämienfunktionen sowie Steuerbefreiungsvariablen enthalten.

Normalerweise bestehen Packages aus zwei Teilen, die separat in der Datenbank gespeichert werden:

- Spezifikation
- Body (optional)

Das Package selbst lässt sich nicht aufrufen, parametrisieren oder verschachteln. Nachdem der Inhalt erstellt und kompiliert wurde, kann er von mehreren Anwendungen gemeinsam verwendet werden.

Wenn Sie zum ersten Mal ein Konstrukt in einem PL/SQL-Package referenzieren, wird das gesamte Package in den Speicher geladen. Bei späteren Zugriffen auf Konstrukte im selben Package müssen keine Input/Output-(I/O-)Vorgänge mehr ausgeführt werden.

Packages – Vorteile

- **Modularität: Zusammengehörige Konstrukte kapseln**
- **Einfachere Verwaltung: Logisch zusammengehörige Funktionen zusammenhalten**
- **Einfacheres Anwendungsdesign: Spezifikation und Body separat codieren und kompilieren**
- **Informationen ausblenden:**
 - Nur die in der Packagespezifikation enthaltenen Deklarationen sind für die Anwendungen sichtbar und zugänglich.
 - Private Konstrukte im Packagebody sind ausgeblendet und nicht verfügbar.
 - Die gesamte Codierung ist im Packagebody ausgeblendet.

ORACLE

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

Packages bieten nicht nur eine Alternative zum Erstellen von Prozeduren und Funktionen als Standalone-Schemaobjekte, sondern auch verschiedene Vorteile.

- **Modularität und einfache Verwaltung:** Packages kapseln logisch zusammengehörige Programmstrukturen in einem benannten Modul. Die einzelnen Packages sind leicht verständlich. Die Schnittstelle zwischen den Packages ist einfach, deutlich und klar definiert.
- **Einfacheres Anwendungsdesign:** Am Anfang benötigen Sie nur die Schnittstelleninformationen in der Packagespezifikation. Spezifikationen lassen sich ohne zugehörigen Body codieren und kompilieren. Danach können auch Stored Subprograms kompiliert werden, die das Package referenzieren. Die Definition des Packagebodys kann warten, bis Sie die Anwendung fertigstellen möchten.
- **Informationen ausblenden:** Sie entscheiden, welche Konstrukte öffentlich (sichtbar und zugänglich) und welche privat (ausgeblendet und nicht verfügbar) sind. Die in der Packagespezifikation enthaltenen Deklarationen sind für die Anwendungen sichtbar und zugänglich. Der Packagebody blendet die Definition der privaten Konstrukte aus, sodass bei Definitionsänderungen nur das Package betroffen ist (und nicht die Anwendung oder ein aufrufendes Programm). Somit können Sie die Implementierung ändern, ohne die aufrufenden Programme rekomplizieren zu müssen. Mit dem Ausblenden der Implementierungsdetails vor den Benutzern schützen Sie auch die Integrität des Packages.

Packages – Vorteile

- **Zusätzliche Funktionalität: Persistenz der öffentlichen Variablen und Cursor**
- **Bessere Performance:**
 - Das gesamte Package wird bei der ersten Referenzierung in den Speicher geladen.
 - Im Speicher befindet sich nur eine Kopie für alle Benutzer.
 - Die Abhängigkeitshierarchie wurde vereinfacht.
- **Überladung: Mehrere gleichnamige Unterprogramme**

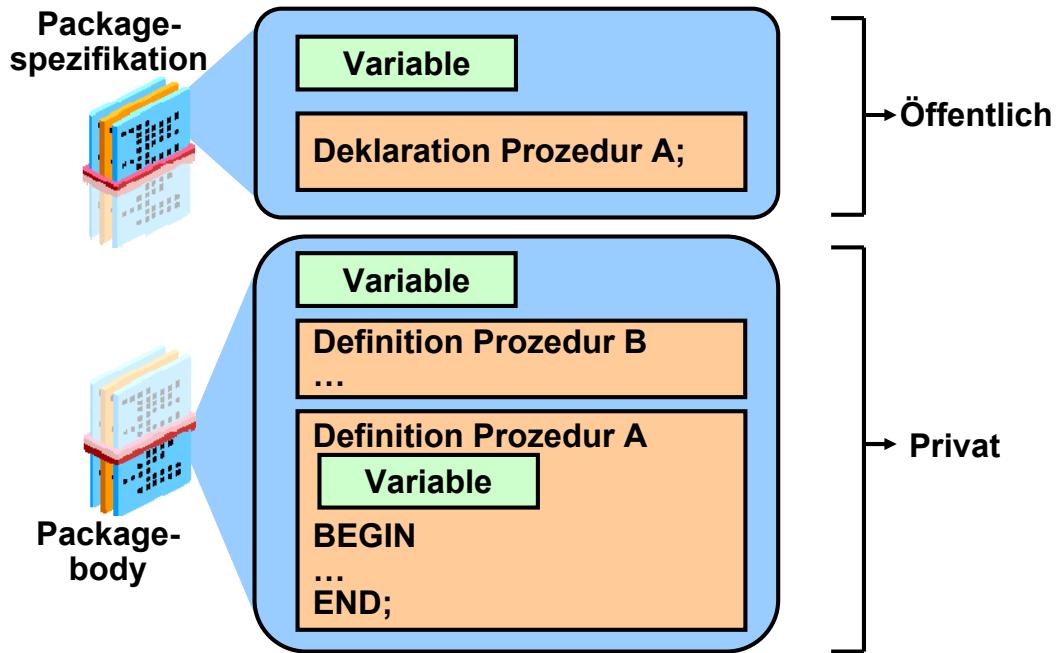
ORACLE

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

- **Zusätzliche Funktionalität:** Die in einem Package integrierten öffentlichen Variablen und Cursor persistieren für die Dauer einer Session. Daher können sie von allen in der Umgebung ausgeführten Unterprogrammen gemeinsam verwendet werden. Überdies ermöglichen sie die transaktionsübergreifende Verwaltung von Daten, ohne diese in der Datenbank speichern zu müssen. Private Konstrukte persistieren ebenfalls für die Dauer der Session, sind aber nur im Package zugänglich.
- **Bessere Performance:** Wenn Sie zum ersten Mal ein in ein Package integriertes Unterprogramm aufrufen, wird das gesamte Package in den Speicher geladen. Daher müssen bei späteren Aufrufen zugehöriger Unterprogramme im Package keine weiteren I/O-Vorgänge mehr ausgeführt werden. Die in ein Package integrierten Unterprogramme vermeiden auch kaskadierende Abhängigkeiten und somit unnötige Kompilierungen.
- **Überladung:** Mit Packages können Sie Prozeduren und Funktionen überladen. Dies bedeutet, dass Sie in einem Package mehrere Unterprogramme mit demselben Namen erstellen können, wobei jedes Unterprogramm Parameter unterschiedlicher Anzahl bzw. unterschiedlichen Datentyps verwendet.

Hinweis: Genauere Informationen zu Abhängigkeiten finden Sie in der Lektion "Abhängigkeiten verwalten".

PL/SQL-Packages – Komponenten



ORACLE

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

Sie erstellen ein Package in zwei Teilen:

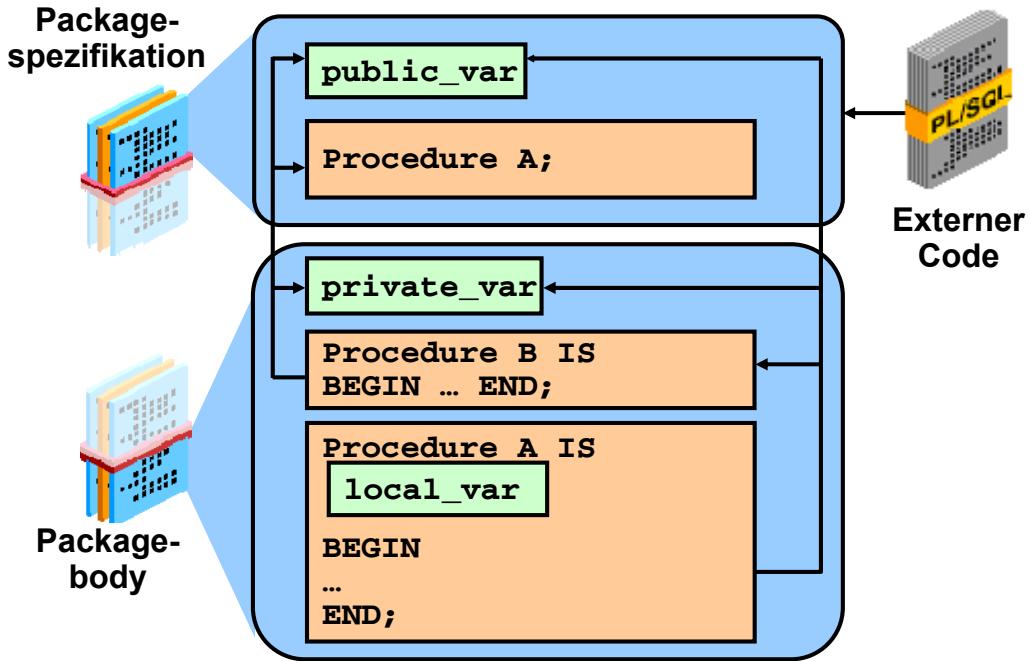
- Die **Packagespezifikation** ist die Schnittstelle zu den Anwendungen. Sie deklariert die verfügbaren öffentlichen Typen, Variablen, Konstanten, Exceptions, Cursor und Unterprogramme. Die Packagespezifikation kann auch PRAGMAS enthalten. Dabei handelt es sich um Anweisungen für den Compiler.
- Der **Packagebody** definiert seine eigenen Unterprogramme und muss die in der Spezifikation deklarierten Unterprogramme vollständig implementieren. Der Packagebody kann auch PL/SQL-Konstrukte wie Typen, Variablen, Konstanten, Exceptions und Cursor definieren.

Öffentliche Komponenten werden in der Packagespezifikation deklariert. Die Spezifikation definiert eine öffentliche Application Programming Interface (API) für Benutzer von Package-features und -funktionen. Öffentliche Komponenten können also aus jeder Oracle-Server-umgebung außerhalb des Packages referenziert werden.

Private Komponenten werden im Packagebody platziert und lassen sich nur von anderen Konstrukten innerhalb desselben Packagebodys referenzieren. Private Komponenten können die öffentlichen Komponenten eines Packages referenzieren.

Hinweis: Wenn eine Packagespezifikation keine Deklarationen von Unterprogrammen enthält, ist kein Packagebody erforderlich.

Packagekomponenten – Interne und externe Sichtbarkeit



ORACLE

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

Mit der Sichtbarkeit einer Komponente wird beschrieben, ob sie von anderen Komponenten oder Objekten gesehen, d. h. referenziert und verwendet werden kann. Die Sichtbarkeit von Komponenten hängt davon ab, ob sie *lokal* oder *global* deklariert werden.

Lokale Komponenten sind innerhalb der Struktur sichtbar, in der sie deklariert werden. Beispiele:

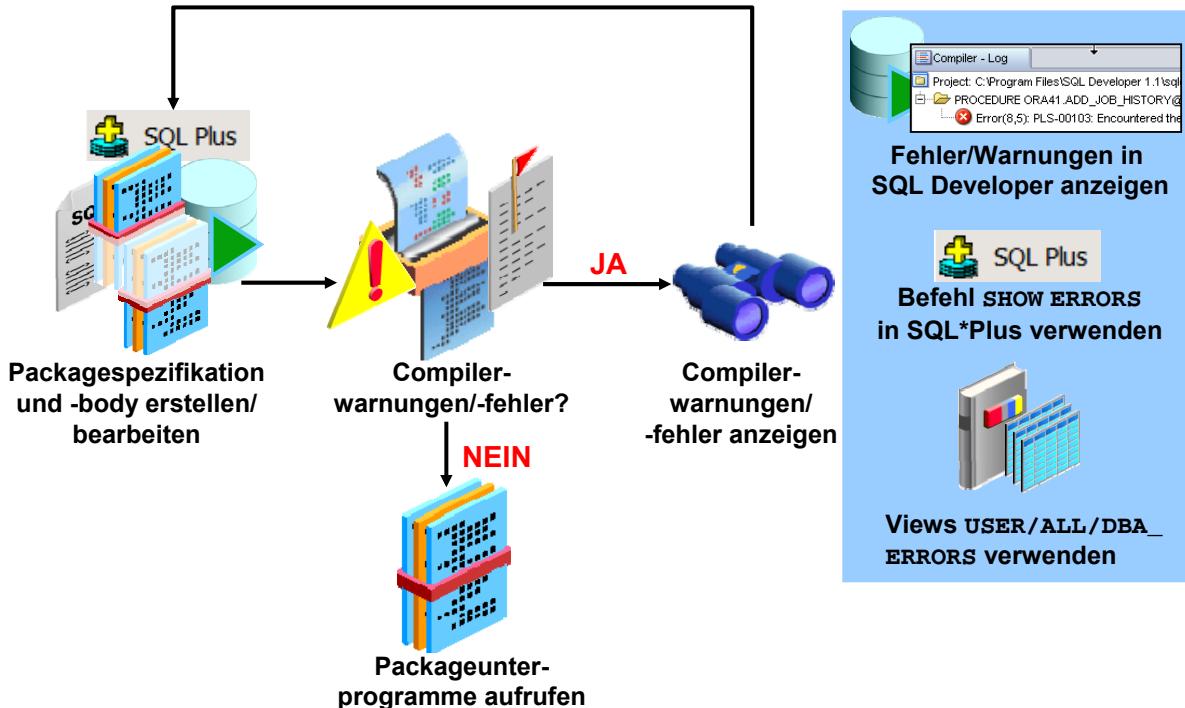
- In Unterprogrammen definierte Variablen können innerhalb des entsprechenden Unterprogramms referenziert werden und sind für externe Komponenten nicht sichtbar. Beispiel: `local_var` kann in Prozedur A verwendet werden.
- Private Packagevariablen, die in Packagebodys deklariert werden, können von anderen Komponenten im selben Packagebody referenziert werden. Sie sind für andere Unterprogramme oder Objekte außerhalb des Packages nicht sichtbar. Beispiel: `private_var` kann von den Prozeduren A und B innerhalb des Packagebodys verwendet werden, jedoch nicht außerhalb des Packages.

Global deklarierte Komponenten sind innerhalb und außerhalb der Packages sichtbar. Beispiele: Öffentliche Variablen, die in einer Packagespezifikation deklariert werden, können außerhalb des Packages referenziert und geändert werden. (Beispiel: `public_var` kann extern referenziert werden.)

- Packageunterprogramme in der Spezifikation können aus externen Codequellen aufgerufen werden. (Beispiel: Prozedur A kann aus einer Umgebung außerhalb des Packages aufgerufen werden.)

Hinweis: Private Unterprogramme wie Prozedur B lassen sich nur mit öffentlichen Unterprogrammen wie Prozedur A oder mit anderen privaten Packagekonstrukten aufrufen. Eine in der Packagespezifikation deklarierte öffentliche Variable ist eine globale Variable.

PL/SQL-Packages entwickeln – Überblick



ORACLE

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

PL/SQL-Packages entwickeln

Die Grafik auf der Folie veranschaulicht die Hauptschritte beim Entwickeln und Verwenden eines Packages:

1. Erstellen Sie die Prozedur im Object Navigator-Baum oder im SQL Worksheet-Bereich von SQL Developer.
2. Komplizieren Sie das Package. Das Package wird in der Datenbank erstellt. Die CREATE PACKAGE-Anweisung erstellt und speichert den Quellcode und den kompilierten *m*-Code in der Datenbank. Um das Package zu komplizieren, klicken Sie im Object Navigator-Baum mit der rechten Maustaste auf den Namen des Packages. Klicken Sie anschließend auf **Compile**.
3. Wenn keine Komplizierungswarnungen oder -fehler aufgetreten sind, führen Sie alle öffentlichen Konstrukte innerhalb der Packagespezifikation über eine Oracle-Serverumgebung aus.
4. Wenn Komplizierungswarnungen oder -fehler aufgetreten sind, können Sie diese mit einer der folgenden Methoden anzeigen (und dann korrigieren):
 - SQL Developer-Benutzeroberfläche (Registerkarte **Compiler – Log**)
 - SQL*Plus-Befehl `SHOW ERRORS`
 - Views `USER/ALL/DBA_ERRORS`

Lektionsagenda

- Vorteile und Komponenten von Packages bestimmen
- Mit Packages arbeiten:
 - Packagespezifikationen und Packagebodys erstellen
 - Packageunterprogramme aufrufen
 - Packageinformationen anzeigen
 - Packages entfernen

ORACLE

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

Packagespezifikationen mit CREATE PACKAGE-Anweisungen erstellen

```
CREATE [OR REPLACE] PACKAGE package_name IS|AS  
    public type and variable declarations  
    subprogram specifications  
END [package_name];
```

- Mit der Option OR REPLACE wird die Packagespezifikation gelöscht und neu erstellt.
- Die in der Packagespezifikation deklarierten Variablen werden standardmäßig mit NULL initialisiert.
- Alle in einer Packagespezifikation deklarierten Konstrukte sind für Benutzer sichtbar, denen Berechtigungen für das Package erteilt wurden.

ORACLE

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

Packagespezifikationen erstellen

Um Packages zu erstellen, deklarieren Sie alle öffentlichen Konstrukte in der Packagespezifikation.

- Geben Sie die Option OR REPLACE an, wenn eine vorhandene Packagespezifikation überschrieben werden soll.
- Initialisieren Sie eine Variable gegebenenfalls mit einem konstanten Wert oder einer Formel in der Deklaration. Andernfalls wird die Variable implizit mit NULL initialisiert.

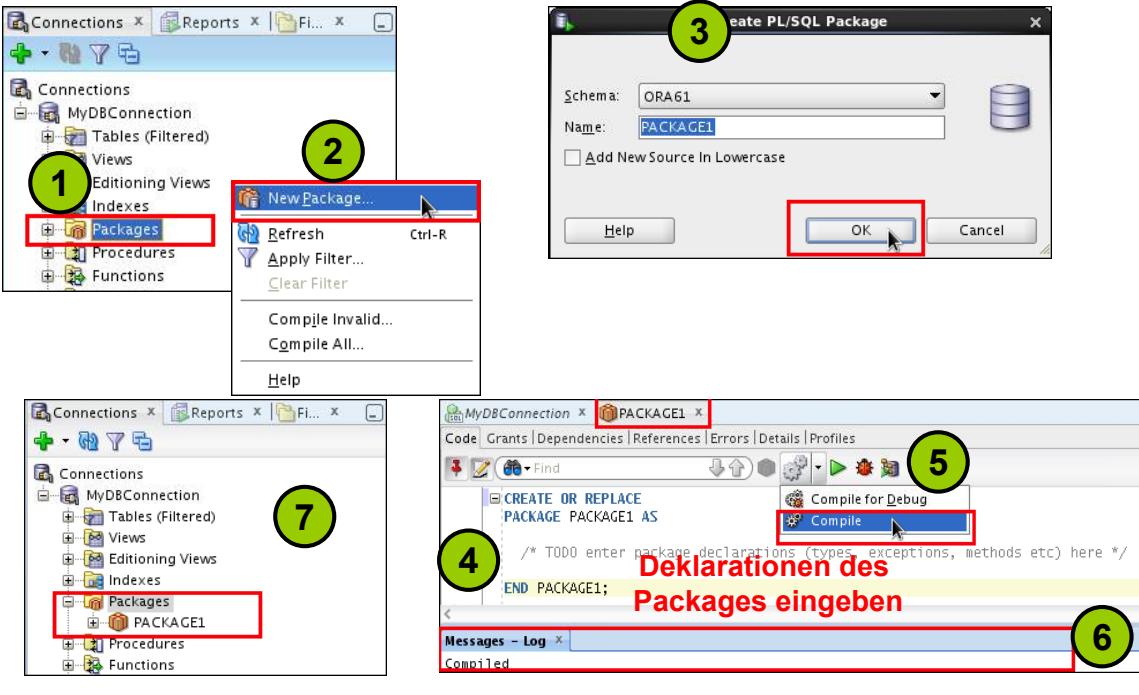
Die Elemente in der Packagesyntax haben folgende Bedeutung:

- **package_name** gibt einen Namen für das Package an, der unter den Objekten des Eigentümerschemas eindeutig sein muss. Nach dem Schlüsselwort END können Sie optional den Packagename aufnehmen.
- **public type and variable declarations** deklariert öffentliche Variablen, Konstanten, Cursor, Exceptions, benutzerdefinierte Typen und Subtypen.
- **subprogram specification** gibt die Deklarationen der öffentlichen Prozeduren oder Funktionen an.

Die Packagespezifikation muss die Überschriften der Prozeduren und Funktionen enthalten, abgeschlossen durch ein Semikolon, ohne das Schlüsselwort `IS` (oder `AS`) und seinen PL/SQL-Block. Die in Packagespezifikationen deklarierten Prozeduren oder Funktionen werden im Packagebody implementiert.

Die Oracle-Datenbank speichert die Spezifikation und den Body eines Packages separat. Auf diese Weise können Sie die Implementierung eines Programmkonstrukts im Packagebody ändern, ohne andere Schemaobjekte zu invalidieren, die das Programmkonstrukt aufrufen oder referenzieren.

Packagespezifikationen mithilfe von SQL Developer erstellen

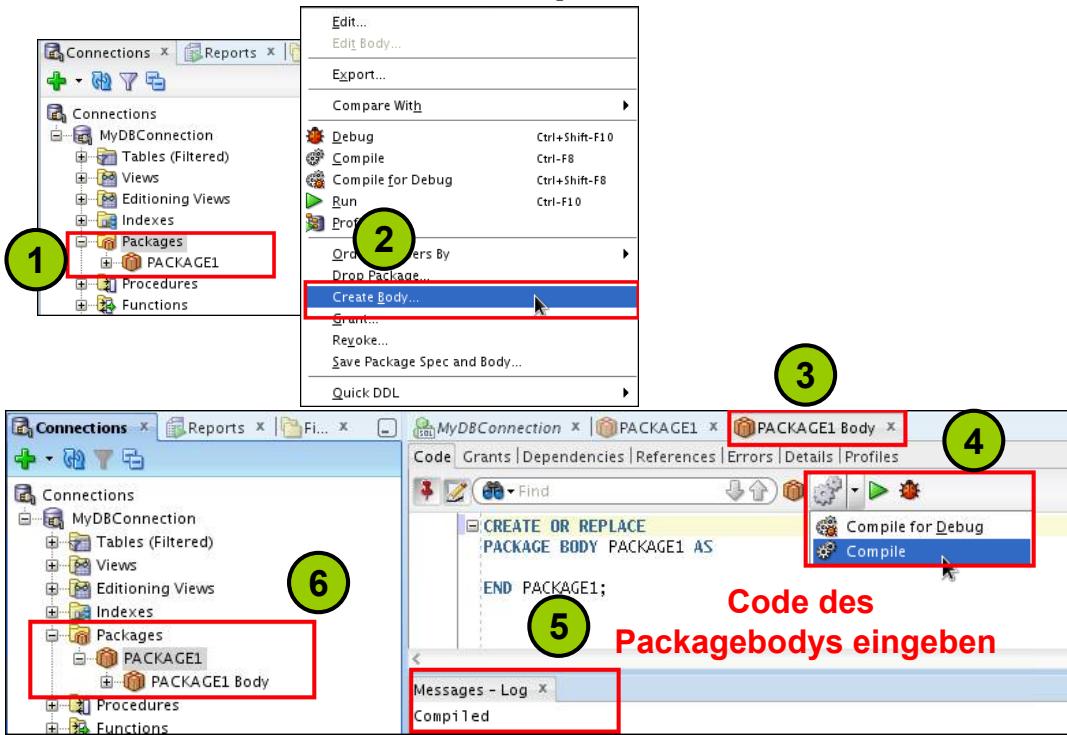


Copyright © 2013, Oracle und/oder verbundene Unternehmen. All rights reserved. Alle Rechte vorbehalten.

Mit SQL Developer können Sie die Packagespezifikation wie folgt erstellen:

1. Klicken Sie im Navigator-Baum **Connections** mit der rechten Maustaste auf den Knoten **Packages**.
2. Wählen Sie im Kontextmenü die Option **New Package**.
3. Wählen Sie im Fenster **Create PL/SQL Package** den Schemanamen, geben Sie den Namen für das neue Package ein, und klicken Sie dann auf **OK**. Für das neue Package werden eine Registerkarte und die Shell angezeigt.
4. Geben Sie den Code für das neue Package ein.
5. Komplizieren Sie das neue Package, oder speichern Sie es (mit dem Symbol **Save** in der Hauptsymbolleiste).
6. In der Registerkarte **Messages – Log** wird angezeigt, ob die Komplizierung erfolgreich verlaufen ist.
7. Das neu erstellte Package wird im Navigator-Baum **Connections** unter dem Knoten **Packages** angezeigt.

Packagebodys mithilfe von SQL Developer erstellen



ORACLE

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

Mit SQL Developer können Sie den Packagebody wie folgt erstellen:

1. Klicken Sie im Navigator-Baum **Connections** im Knoten **Packages** mit der rechten Maustaste auf den Namen des Packages, für das Sie einen Body erstellen.
2. Wählen Sie im Kontextmenü die Option **Create Body**. Für den neuen Packagebody werden eine Registerkarte und die Shell angezeigt.
3. Geben Sie den Code für den neuen Packagebody ein.
4. Komplizieren oder speichern Sie den Packagebody.
5. In der Registerkarte **Messages – Log** wird angezeigt, ob die Komplizierung erfolgreich verlaufen ist.
6. Der neu erstellte Packagebody wird im Navigator-Baum **Connections** unter dem Knoten **Packages** angezeigt.

Packagespezifikationen – Beispiel: comm_pkg

```
-- The package spec with a public variable and a
-- public procedure that are accessible from
-- outside the package.

CREATE OR REPLACE PACKAGE comm_pkg IS
    v_std_comm NUMBER := 0.10; --initialized to 0.10
    PROCEDURE reset_comm(p_new_comm NUMBER);
END comm_pkg;
/
```

- **V_STD_COMM ist eine öffentliche globale Variable, die mit 0.10 initialisiert wird.**
- **RESET_COMM ist eine öffentliche Prozedur zum Zurücksetzen der Standardprovision auf der Grundlage bestimmter Geschäftsregeln. Sie wird im Packagebody implementiert.**

ORACLE

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

Im Beispiel auf der Folie wird das Package comm_pkg zur Verwaltung der geschäftlichen Verarbeitungsregeln zur Provisionsberechnung erstellt.

Die öffentliche (globale) Variable v_std_comm wird so deklariert, dass sie den höchstmöglichen Prozentsatz für die Provision in der Benutzersession aufnimmt, und mit 0.10 (d. h. 10 %) initialisiert.

Die öffentliche Prozedur reset_comm wird so deklariert, dass sie einen neuen Provisionsprozentsatz annimmt, mit dem der standardmäßige Prozentsatz aktualisiert wird, wenn die Validierungsregeln für die Provision akzeptiert werden. Die Validierungsregeln zum Zurücksetzen der Provision werden weder veröffentlicht noch in der Packagespezifikation angezeigt. Die Verwaltung der Validierungsregeln erfolgt mit einer privaten Funktion im Packagebody.

Packagebodys erstellen

```
CREATE [OR REPLACE] PACKAGE BODY package_name IS | AS
    private type and variable declarations
    subprogram bodies
[BEGIN initialization statements]
END [package_name];
```

- **Mit der Option OR REPLACE wird der Packagebody gelöscht und neu erstellt.**
- **Die im Packagebody definierten Bezeichner sind privat und außerhalb des Packagebodys nicht sichtbar.**
- **Alle privaten Konstrukte müssen vor ihrer Referenzierung deklariert werden.**
- **Öffentliche Konstrukte sind für den Packagebody sichtbar.**

ORACLE

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

Erstellen Sie Packagebodys, um alle öffentlichen Unterprogramme und unterstützenden privaten Konstrukte zu definieren und zu implementieren. Gehen Sie zum Erstellen eines Packagebodys wie folgt vor:

- Um einen vorhandenen Packagebody zu überschreiben, geben Sie die Option OR REPLACE an.
- Definieren Sie die Unterprogramme in der korrekten Reihenfolge. Grundsätzlich müssen Sie Variablen oder Unterprogramme deklarieren, bevor sie von anderen Komponenten im selben Packagebody referenziert werden können. Üblicherweise werden im Packagebody zuerst alle privaten Variablen und Unterprogramme und zuletzt die öffentlichen Unterprogramme definiert.
- Schließen Sie die Implementierung für alle in der Packagespezifikation deklarierten Prozeduren oder Funktionen im Packagebody ab.

Die Elemente in der Packagebody-Syntax haben folgende Bedeutung:

- **package_name** gibt einen Namen für das Package an. Dieser muss mit dem Namen der Packagespezifikation übereinstimmen. Nach dem Schlüsselwort END können Sie optional den Packagennamen verwenden.
- **private type and variable declarations** deklariert private Variablen, Konstanten, Cursor, Exceptions, benutzerdefinierte Typen und Subtypen.
- **subprogram specification** gibt die vollständige Implementierung aller privaten und/oder öffentlichen Prozeduren oder Funktionen an.
- **[BEGIN initialization statements]** ist ein optionaler Block mit Initialisierungscode, der bei der ersten Packagerefenzierung ausgeführt wird.

Packagebodys – Beispiel: comm_pkg

```

CREATE OR REPLACE PACKAGE BODY comm_pkg IS
    FUNCTION validate(p_comm NUMBER) RETURN BOOLEAN IS
        v_max_comm employees.commission_pct%type;
    BEGIN
        SELECT MAX(commission_pct) INTO v_max_comm
        FROM employees;
        RETURN (p_comm BETWEEN 0.0 AND v_max_comm);
    END validate;

    PROCEDURE reset_comm (p_new_comm NUMBER) IS
    BEGIN
        IF validate(p_new_comm) THEN
            v_std_comm := p_new_comm; -- reset public var
        ELSE
            RAISE_APPLICATION_ERROR(
                -20210, 'Bad Commission');
        END IF;
    END reset_comm;
END comm_pkg;

```

ORACLE

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

Die Folie zeigt den gesamten Packagebody von comm_pkg mit der privaten Funktion validate, die auf gültige Provisionen prüft. Die Validierung setzt voraus, dass die Provision positiv und kleiner als die höchste Provision unter den vorhandenen Mitarbeitern ist. Die Prozedur reset_comm ruft die private Validierungsfunktion vor der Änderung der Standardprovision in v_std_comm auf. Beachten Sie im Beispiel Folgendes:

- Die in der Prozedur reset_comm referenzierte Variable v_std_comm ist eine öffentliche Variable. Die in der Packagespezifikation deklarierten Variablen wie v_std_comm können direkt ohne Qualifizierung referenziert werden.
- Die Prozedur reset_comm implementiert die öffentliche Definition in der Spezifikation.
- Im Body für comm_pkg ist validate eine private Funktion, die von der Prozedur reset_comm direkt ohne Qualifizierung referenziert wird.

Hinweis: Die Funktion validate wird vor der Prozedur reset_comm angezeigt, da die Prozedur reset_comm die Funktion validate referenziert. Sie können im Packagebody Vorwärts-deklarationen für Unterprogramme erstellen, wenn ihre Reihenfolge geändert werden muss. Wenn in einer Packagespezifikation nur Typen, Konstanten, Variablen und Exceptions ohne Unterprogrammspezifikationen deklariert werden, ist kein Packagebody erforderlich. Sie können jedoch mit dem Body die in der Packagespezifikation deklarierten Elemente initialisieren.

Packageunterprogramme aufrufen – Beispiele

```
-- Invoke a function within the same package:
CREATE OR REPLACE PACKAGE BODY comm_pkg IS ...
  PROCEDURE reset_comm(p_new_comm NUMBER) IS
  BEGIN
    IF validate(p_new_comm) THEN
      v_std_comm := p_new_comm;
    ELSE ...
    END IF;
  END reset_comm;
END comm_pkg;
```

```
-- Invoke a package procedure from SQL*Plus:
EXECUTE comm_pkg.reset_comm(0.15)
```

```
-- Invoke a package procedure in a different schema:
EXECUTE scott.comm_pkg.reset_comm(0.15)
```

ORACLE

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

Packageunterprogramme aufrufen

Nachdem Sie das Package in der Datenbank gespeichert haben, können Sie öffentliche oder private Unterprogramme innerhalb desselben Packages oder öffentliche Unterprogramme außerhalb des Packages aufrufen. Wenn ein Unterprogramm außerhalb des Packages aufgerufen wird, müssen Sie seinen Packagename vollständig angeben. Verwenden Sie dazu die Syntax `package_name.subprogram`.

Beim Aufrufen eines Unterprogramms innerhalb desselben Packages ist die vollständige Angabe des Namens optional.

1. Beispiel: Ruft die Funktion `validate` aus der Prozedur `reset_comm` innerhalb desselben Packages auf. Das Präfix des Packagennamens ist nicht obligatorisch, sondern optional.

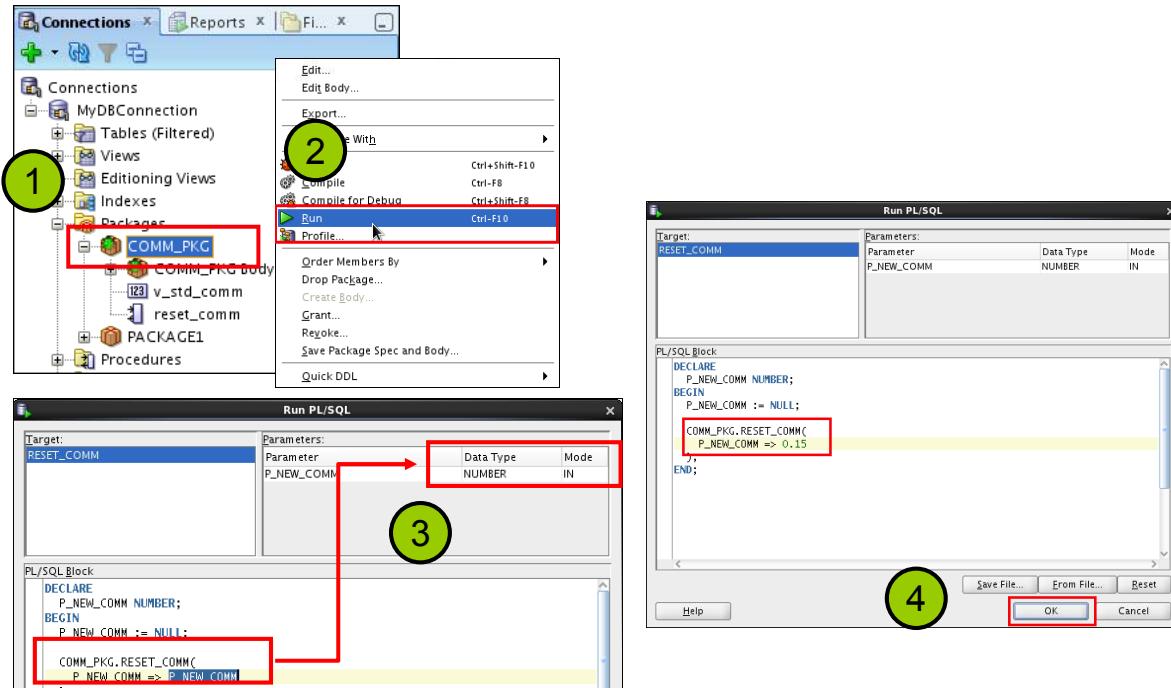
2. Beispiel: Ruft die Prozedur `reset_comm` aus SQL*Plus (einer Umgebung außerhalb des Packages) auf, um die geltende Provision für die Benutzersession auf 0.15 zurückzusetzen.

3. Beispiel: Ruft die Prozedur `reset_comm` auf, deren Eigentümer der Schemabenutzer SCOTT ist. In SQL*Plus wird der qualifizierten Packageprozedur der Schemaname vorangestellt. Sie können dies mit einem Synonym vereinfachen, das `schema.package_name` referenziert.

Beispiel: Für eine Remote-Datenbank, in der die Packageprozedur `reset_comm` erstellt wird, wurde der Datenbanklink NY erstellt. Rufen Sie die Remote-Prozedur wie folgt auf:

```
EXECUTE comm_pkg.reset_comm(0.15)
```

Packageunterprogramme mithilfe von SQL Developer aufrufen



ORACLE

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

Mit SQL Developer können Sie das Unterprogramm eines Packages wie folgt aufrufen:

1. Klicken Sie im Navigator-Baum unter dem Knoten **Packages** mit der rechten Maustaste auf den Namen des Packages.
2. Wählen Sie im verschiebbaren Menü die Option **Run**. Das Fenster **Run PL/SQL** wird angezeigt. Mit dem Fenster **Run PL/SQL** können Sie Parameterwerte zur Ausführung einer PL/SQL-Funktion oder -Prozedur angeben. (Wenn Sie ein Package angeben, wählen Sie eine Funktion oder Prozedur im Package.) Geben Sie folgende Werte an:
 - a. **Target:** Wählen Sie den Namen der auszuführenden Funktion oder Prozedur.
 - b. **Parameters:** In diesem Bereich werden alle Parameter für das angegebene Ziel aufgeführt. Die einzelnen Parameter können folgenden Modus besitzen: **IN** (der Wert wird übergeben), **OUT** (der Wert wird zurückgegeben) oder **IN/OUT** (der Wert wird übergeben, und das Ergebnis der Funktions- oder Prozeduraktion wird im Parameter gespeichert).
3. Ändern Sie im Bereich **PL/SQL Block** die Spezifikationen der formalen **IN**- und **IN/OUT**-Parameter in diesem Block in die konkreten Werte, die Sie für die Ausführung der Funktion oder Prozedur verwenden möchten. Beispiel: Um 0.15 als Wert für den Eingabeparameter **P_NEW_COMM** anzugeben, ändern Sie **P_NEW_COMM => P_NEW_COMM** in **P_NEW_COMM => 0.15**.
4. Klicken Sie auf **OK**. SQL Developer führt die Funktion oder Prozedur aus.

Packages ohne Body erstellen und verwenden

```
CREATE OR REPLACE PACKAGE global_consts IS
    c_mile_2_kilo CONSTANT NUMBER := 1.6093;
    c_kilo_2_mile CONSTANT NUMBER := 0.6214;
    c_yard_2_meter CONSTANT NUMBER := 0.9144;
    c_meter_2_yard CONSTANT NUMBER := 1.0936;
END global_consts;
```

```
SET SERVEROUTPUT ON
BEGIN
    DBMS_OUTPUT.PUT_LINE('20 miles = ' ||
        20 * global_consts.c_mile_2_kilo || ' km');
END;
```

```
SET SERVEROUTPUT ON
CREATE FUNCTION mtr2yrd(p_m NUMBER) RETURN NUMBER IS
BEGIN
    RETURN (p_m * global_consts.c_meter_2_yard);
END mtr2yrd;
/
EXECUTE DBMS_OUTPUT.PUT_LINE(mtr2yrd(1))
```

ORACLE

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

Die in Standalone-Unterprogrammen deklarierten Variablen und Konstanten sind nur während der Ausführung des Unterprogramms vorhanden. Zur Bereitstellung von Daten während der gesamten Benutzersession müssen Sie eine Packagespezifikation mit Deklarationen zu öffentlichen (globalen) Variablen und Konstanten erstellen. Erstellen Sie in diesem Fall eine Packagespezifikation ohne Packagebody. Wie bereits erwähnt, ist kein Packagebody erforderlich, wenn in einer Spezifikation nur Typen, Konstanten, Variablen und Exceptions deklariert werden.

Beispiele

Im ersten Codefeld auf der Folie wird eine Packagespezifikation ohne Body mit mehreren Konstanten erstellt, die als Umrechnungssätze dienen. Für die Unterstützung dieser Packagespezifikation ist kein Packagebody erforderlich. Es wird angenommen, dass die Anweisung SET SERVEROUTPUT ON vor Ausführung der Codebeispiele auf der Folie abgesetzt wurde.

Im zweiten Codefeld wird die Konstante `c_mile_2_kilo` im Package `global_consts` referenziert, indem der Packagename dem Konstantenbezeichner vorangestellt wird.

Im dritten Beispiel wird die Standalone-Funktion `c_mtr2yrd` zur Umrechnung von Metern in Yards erstellt. Als Umrechnungssatz wird die im Package `global_consts` deklarierte Konstante `c_meter_2_yard` verwendet. Die Funktion wird im Parameter `DBMS_OUTPUT.PUT_LINE` aufgerufen.

Regel: Wenn Sie Variablen, Cursor, Konstanten oder Exceptions von außerhalb des Packages referenzieren, müssen Sie den Packagennamen angeben.

Packages im Data Dictionary anzeigen

```
-- View the package specification.
SELECT text
FROM user_source
WHERE name = 'COMM_PKG' AND type = 'PACKAGE'
ORDER BY LINE;
```

TEXT
1 PACKAGE comm_pkg IS
2 v_std_comm NUMBER := 0.10; --initialized to 0.10
3 PROCEDURE reset_comm(p_new_comm NUMBER);
4 END comm_pkg;

```
-- View the package body.
SELECT text
FROM user_source
WHERE name = 'COMM_PKG' AND type = 'PACKAGE BODY'
ORDER BY LINE;
```

TEXT
1 PACKAGE BODY comm_pkg IS
2 FUNCTION validate(comm NUMBER) RETURN BOOLEAN IS
3 max_comm employees.commission_pct%type;
4 BEGIN
5 SELECT MAX(commission_pct) INTO max_comm
6 FROM employees;
7 RETURN (comm BETWEEN 0.0 AND max_comm);

ORACLE

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

Packages im Data Dictionary anzeigen

Der Quellcode für PL/SQL-Packages wird auch in den Data Dictionary Views `USER_SOURCE` und `ALL_SOURCE` gespeichert. Mit der Tabelle `USER_SOURCE` wird der PL/SQL-Code angezeigt, dessen Eigentümer Sie sind. Mit der Tabelle `ALL_SOURCE` zeigen Sie den PL/SQL-Code an, für den Ihnen der Eigentümer des jeweiligen Unterprogrammcodes die Berechtigung `EXECUTE` erteilt hat. Neben den vorhergehenden Spalten stellt die Tabelle die Spalte `OWNER` bereit.

Verwenden Sie bei der Abfrage von Packages eine Bedingung, in der die Spalte `TYPE`:

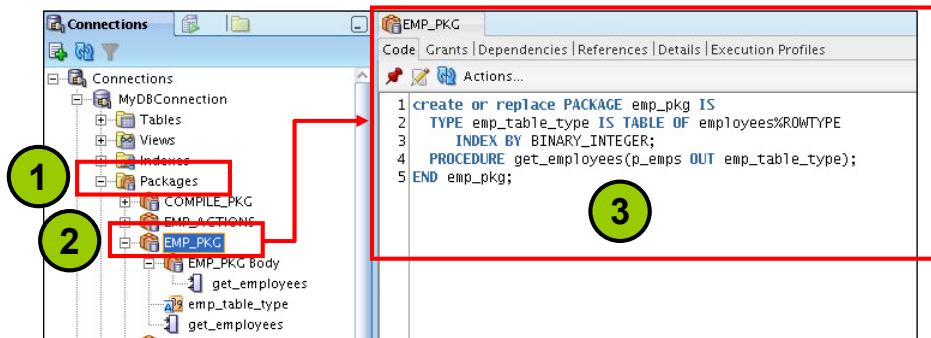
- gleich '`PACKAGE`' ist, um den Quellcode für die Packagespezifikation anzuzeigen
- gleich '`PACKAGE BODY`' ist, um den Quellcode für den Packagebody anzuzeigen

Sie können die Packagespezifikation und den Packagebody in SQL Developer auch über den Packagennamen im Knoten **Packages** anzeigen.

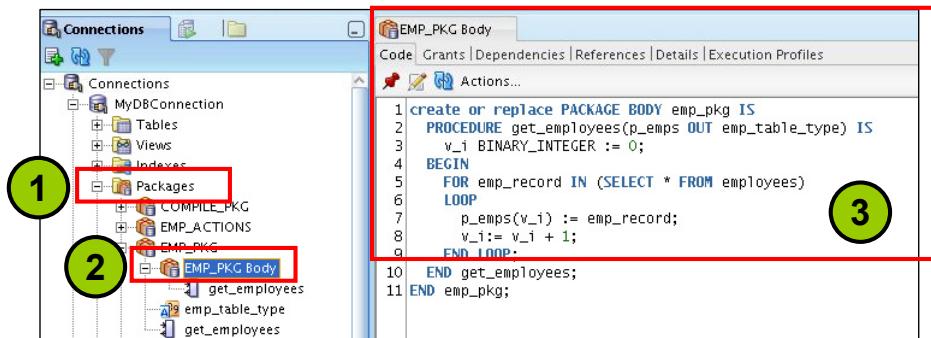
Hinweis: Die Anzeige des Quellcodes für PL/SQL-Built-In-Packages von Oracle oder für PL/SQL, dessen Quellcode mithilfe eines WRAP-Utilitys oder durch Obfuscation gewrapped wurde, ist nicht möglich. Obfuscation und das Wrappen von PL/SQL-Quellcode werden in einer späteren Lektion ausführlich behandelt. Wenn Sie in der SQL Worksheet-Symbolleiste auf das Symbol **Execute Statement** (F9) klicken (statt auf das Symbol **Run Script**), wird die Ausgabe in der Registerkarte **Results** unter Umständen besser formatiert angezeigt.

Packages in SQL Developer anzeigen

Zum Anzeigen der Packagespezifikation auf den Packagenamen klicken



Zum Anzeigen des Packagebodys auf den Packagebody klicken



ORACLE

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

Um die Spezifikation eines Packages in SQL Developer anzuzeigen, gehen Sie wie folgt vor:

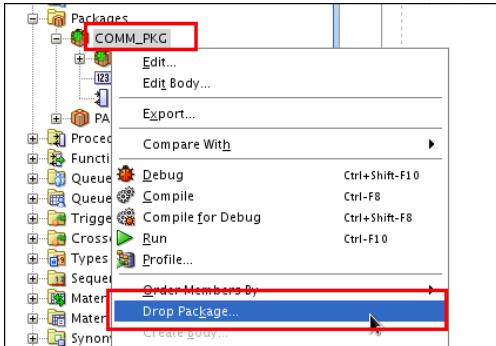
1. Klicken Sie in der Registerkarte **Connections** auf den Knoten **Packages**.
2. Klicken Sie auf den Namen des Packages.
3. Der Spezifikationscode des Packages wird in der Registerkarte **Code** angezeigt (siehe Folie).

Um den Body eines Packages in SQL Developer anzuzeigen, gehen Sie wie folgt vor:

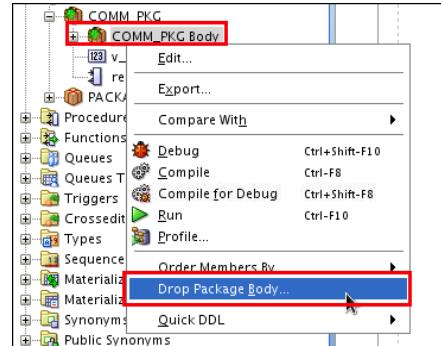
1. Klicken Sie in der Registerkarte **Connections** auf den Knoten **Packages**.
2. Klicken Sie auf den Body des Packages.
3. Der Bodycode des Packages wird in der Registerkarte **Code** angezeigt.

Packages mit SQL Developer oder der SQL-Anweisung DROP entfernen

Packagespezifikation und Packagebody löschen



Nur Packagebody löschen



```
-- Remove the package specification and body
DROP PACKAGE package_name;
```

```
-- Remove the package body only
DROP PACKAGE BODY package_name;
```

ORACLE

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

Packages entfernen

Wird ein Package nicht mehr benötigt, können Sie es mit einer SQL-Anweisung in SQL Developer entfernen. Packages bestehen aus zwei Teilen. Sie können entweder das gesamte Package oder nur den Packagebody entfernen. (Im zweiten Fall bleibt die Packagespezifikation erhalten.)

Packages erstellen – Richtlinien

- **Packages für den allgemeinen Gebrauch entwickeln**
- **Packagespezifikation vor dem Body definieren**
- **Nur Konstrukte in die Packagespezifikation aufnehmen, die für den öffentlichen Zugriff vorgesehen sind**
- **Elemente im Deklarationsteil des Packagebodys platzieren, wenn sie in der gesamten Session oder transaktionsübergreifend verwaltet werden müssen**
- **Aufgrund der fein granulierten Abhängigkeitsverwaltung müssen referenzierende Unterprogramme bei Änderung einer Packagespezifikation seltener rekompiliert werden.**
- **Möglichst wenig Konstrukte in die Packagespezifikation aufnehmen**

ORACLE

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

Formulieren Sie die Packages so allgemein wie möglich, damit sie auch in zukünftigen Anwendungen wiederverwendet werden können. Erstellen Sie außerdem keine Packages, die Features bieten, die auch vom Oracle-Server bereitgestellt werden.

Packagespezifikationen spiegeln das Design Ihrer Anwendung wider. Daher sollten Sie diese vor den Packagebodys definieren. Die Packagespezifikation darf nur Konstrukte enthalten, die für die Benutzer des Packages sichtbar sein müssen. Auf diese Weise können andere Entwickler das Package nicht missbrauchen, indem sie Code auf irrelevanten Details aufbauen.

Platzieren Sie Elemente im Deklarationsteil des Packagebodys, wenn Sie diese in der gesamten Session oder transaktionsübergreifend verwalten müssen. Deklarieren Sie z. B. die Variable NUMBER_EMPLOYED als private Variable, wenn jeder Prozeduraufruf verwaltet werden muss, in dem diese Variable verwendet wird. Wenn Sie die Variable in der Packagespezifikation als globale Variable deklariert haben, wird ihr Wert in einer Session initialisiert, sobald ein Konstrukt aus dem Package zum ersten Mal aufgerufen wird.

Vor Oracle Database 11g erforderten Änderungen am Packagebody keine Rekompilierung der abhängigen Konstrukte. Bei Änderungen der Packagespezifikation hingegen war eine Rekompilierung aller Stored Subprograms nötig, die das Package referenzieren. Oracle Database 11g reduziert diese Abhängigkeit. Abhängigkeiten werden nun auf Elementebene innerhalb der Einheit überwacht. Die fein granulierte Abhängigkeitsverwaltung ist Thema einer späteren Lektion.

Quiz

Die Packagespezifikation ist die Schnittstelle zu den Anwendungen. Sie deklariert die verfügbaren öffentlichen Typen, Variablen, Konstanten, Exceptions, Cursor und Unterprogramme. Die Packagespezifikation kann auch PRAGMAAs enthalten. Dabei handelt es sich um Anweisungen für den Compiler.

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

ORACLE

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

Richtige Antwort: a

Zusammenfassung

In dieser Lektion haben Sie Folgendes gelernt:

- Packages beschreiben und ihre Komponenten auflisten
- Packages erstellen, um zusammengehörige Variablen, Cursor, Konstanten, Exceptions, Prozeduren und Funktionen zu gruppieren
- Packagekonstrukte als öffentlich oder privat angeben
- Packagekonstrukte aufrufen
- Verwendungsmöglichkeiten von Packages ohne Body beschreiben

ORACLE

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

Zusammengehörige Prozeduren und Funktionen werden in einem Package gruppiert. Packages verbessern die Organisation, Verwaltung, Sicherheit und Performance.

Ein Package besteht aus einer Packagespezifikation und einem Packagebody. Sie können einen Packagebody ändern, ohne dass die zugehörige Packagespezifikation davon betroffen ist.

Mit Packages können Sie Quellcode vor Benutzern verbergen. Beim ersten Aufruf eines Packages wird das gesamte Package in den Speicher geladen. Dadurch wird der Datenträgerzugriff im Hinblick auf nachfolgende Aufrufe so gering wie möglich gehalten.

Übungen zu Lektion 4 – Überblick: Packages erstellen und verwenden

Diese Übungen behandeln folgende Themen:

- Packages erstellen
- Packageprogrammeinheiten aufrufen

ORACLE

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

Übungen zu Lektion 4 – Überblick

In dieser Übung erstellen Sie Packagespezifikationen und Packagebodys. Anschließend rufen Sie mithilfe von Beispieldaten die Konstrukte in den Packages auf.

Hinweis: Bei Verwendung von SQL Developer werden die Kompilierungszeitfehler in der Registerkarte **Messages - Log** angezeigt. Wenn Sie den gespeicherten Code mit SQL*Plus erstellen, zeigen Sie Kompilierungsfehler mit SHOW ERRORS an.

Mit Packages arbeiten

ORACLE

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

Ziele

Nach Ablauf dieser Lektion haben Sie folgende Ziele erreicht:

- **Packageprozeduren und -funktionen überladen**
- **Vorwärtsdeklarationen verwenden**
- **Initialisierungsblöcke in einem Packagebody erstellen**
- **Persistente Zustände von Packagedaten für die Sessiondauer verwalten**
- **Assoziative Arrays (Index By-Tabellen) und Records in Packages verwenden**

ORACLE

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

In dieser Lektion werden anspruchsvollere PL/SQL-Features wie Überladung, Vorwärtsreferenzierung, Einmalprozeduren sowie die Persistenz von Variablen, Konstanten, Exceptions und Cursorn vorgestellt. Außerdem wird erläutert, welche Auswirkungen das Packaging von Funktionen hat, die in SQL-Anweisungen verwendet werden.

Lektionsagenda

- **Packageunterprogramme überladen, Vorwärts-deklarationen verwenden und Initialisierungsblöcke in Packagebodys erstellen**
- **Persistente Zustände von Packagedaten für die Sessiondauer verwalten und assoziative Arrays (Index By-Tabellen) sowie Records in Packages verwenden**

ORACLE

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

Unterprogramme in PL/SQL überladen

- Ermöglicht die Erstellung mehrerer Unterprogramme mit identischem Namen
- Erfordert eine unterschiedliche Anzahl, Reihenfolge oder Datentypfamilie der formalen Parameter des Unterprogramms
- Ermöglicht die Erstellung flexibler Methoden zum Aufrufen von Unterprogrammen mit unterschiedlichen Daten
- Bietet eine Möglichkeit zum Erweitern der Funktionalität, ohne vorhandenen Code zu verlieren, indem vorhandenen Unterprogrammen neue Parameter hinzugefügt werden
- Bietet eine Möglichkeit zum Überladen von lokalen Unterprogrammen, Packageunterprogrammen und Typmethoden, nicht jedoch von Standalone-Unterprogrammen

ORACLE

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

Mit dem Überladungsfeature von PL/SQL können Sie zwei oder mehr Packageunterprogramme mit identischem Namen entwickeln. Die Überladung ist sinnvoll, wenn ein Unterprogramm ähnliche Parametergruppen mit unterschiedlichen Datentypen annehmen soll. Beispiel: Die Funktion TO_CHAR lässt sich auf mehrere Arten aufrufen, sodass Sie eine Zahl oder ein Datum in eine Zeichenfolge konvertieren können.

PL/SQL ermöglicht Ihnen die Überladung von Namen von Packageunterprogrammen und von Methoden für Objekttypen.

Grundsätzlich gilt, dass verschiedene Unterprogramme denselben Namen aufweisen können, solange sich ihre formalen Parameter in *Anzahl*, *Reihenfolge* oder *Datentypfamilie* unterscheiden.

Das Überladen empfiehlt sich in folgenden Situationen:

- Zwei oder mehr Unterprogramme haben ähnliche Verarbeitungsregeln, unterscheiden sich jedoch durch Typ oder Anzahl der verwendeten Parameter.
- Es sollen alternative Verfahren für die Suche nach verschiedenen Daten mit unterschiedlichen Suchkriterien bereitgestellt werden. Beispiel: Sie möchten Mitarbeiter nach ihrer Personalnummer suchen, aber auch die Suche anhand des Nachnamens ermöglichen. Die eigentliche Logik ist identisch, die Parameter oder Suchkriterien unterscheiden sich jedoch.
- Die Funktionalität soll erweitert werden, ohne vorhandenen Code ersetzen zu müssen.

Hinweis: Standalone-Unterprogramme können nicht überladen werden. Die Erstellung lokaler Unterprogramme in Methoden des Objekttyps wird in diesem Kurs nicht behandelt.

Beschränkungen

In folgenden Fällen ist die Überladung nicht möglich:

- In zwei Unterprogrammen, wenn ihre formalen Parameter sich nur im Datentyp unterscheiden und die unterschiedlichen Datentypen aus der gleichen Familie stammen (die Datentypen `NUMBER` und `DECIMAL` gehören beispielsweise zur gleichen Familie)
- In zwei Unterprogrammen, wenn ihre formalen Parameter sich nur im Subtyp unterscheiden und die unterschiedlichen Subtypen auf Typen in der gleichen Familie basieren (`VARCHAR` und `STRING` sind z. B. PL/SQL-Subtypen von `VARCHAR2`)
- In zwei Funktionen, die sich nur im Rückgabetyp unterscheiden, selbst wenn die Typen aus unterschiedlichen Familien stammen

Sie erhalten einen Laufzeitfehler, wenn Sie Unterprogramme mit den genannten Features überladen.

Hinweis: Die zuvor aufgeführten Einschränkungen gelten, wenn die Namen der Parameter gleich lauten.

Wenn Sie die Parameter unterschiedlich benennen, können Sie Unterprogramme mittels namentlicher Notation für die Parameter aufrufen.

Aufrufe auflösen

Der Compiler sucht eine Deklaration, die mit dem Aufruf übereinstimmt. Er sucht zuerst im aktuellen Gültigkeitsbereich, danach gegebenenfalls in den nachfolgenden, übergeordneten Gültigkeitsbereichen. Der Compiler stoppt den Suchvorgang, wenn er mindestens eine Unterprogrammdeklaration gefunden hat, deren Name mit dem Namen des aufgerufenen Unterprogramms übereinstimmt. Bei Unterprogrammen mit ähnlich lautenden Namen auf gleicher Gültigkeitsbereichsebene benötigt der Compiler eine präzise Übereinstimmung der Anzahl, der Reihenfolge und des Datentyps der tatsächlichen und formalen Parameter.

Prozeduren überladen – Beispiel: Packagespezifikation erstellen

```

CREATE OR REPLACE PACKAGE dept_pkg IS
  PROCEDURE add_department
    (p_deptno departments.department_id%TYPE,
     p_name  departments.department_name%TYPE := 'unknown',
     p_loc   departments.location_id%TYPE := 1700);

  PROCEDURE add_department
    (p_name  departments.department_name%TYPE := 'unknown',
     p_loc   departments.location_id%TYPE := 1700);
END dept_pkg;
/

```

ORACLE

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

Überladung – Beispiel

Auf der Folie wird die Packagespezifikation `dept_pkg` mit der überladenen Prozedur `add_department` gezeigt. In der ersten Deklaration werden drei Parameter verwendet. Diese stellen die Daten für einen neuen Abteilungs-Record bereit, der in die Abteilungstabelle eingefügt wurde. In der zweiten Deklaration kommen nur zwei Parameter zum Einsatz, weil diese Version die Abteilungsnummer intern über eine Oracle-Sequence generiert.

Bei Variablen, mit denen Spalten in Datenbanktabellen gefüllt werden, empfiehlt es sich, die Datentypen mithilfe des Attributs `%TYPE` anzugeben (siehe Beispiel auf der Folie). Eine weitere Möglichkeit sieht wie folgt aus:

```

CREATE OR REPLACE PACKAGE dept_pkg_method2 IS
  PROCEDURE add_department(p_deptno NUMBER,
                           p_name  VARCHAR2 := 'unknown', p_loc NUMBER := 1700);
  ...

```

Prozeduren überladen – Beispiel: Packagebody erstellen

```
-- Package body of package defined on previous slide.
CREATE OR REPLACE PACKAGE BODY dept_pkg  IS
PROCEDURE add_department -- First procedure's declaration
  (p_deptno departments.department_id%TYPE,
   p_name    departments.department_name%TYPE := 'unknown',
   p_loc     departments.location_id%TYPE := 1700) IS
BEGIN
  INSERT INTO departments(department_id,
                          department_name, location_id)
  VALUES  (p_deptno, p_name, p_loc);
END add_department;
PROCEDURE add_department -- Second procedure's declaration
  (p_name    departments.department_name%TYPE := 'unknown',
   p_loc     departments.location_id%TYPE := 1700) IS
BEGIN
  INSERT INTO departments (department_id,
                          department_name, location_id)
  VALUES (departments_seq.NEXTVAL, p_name, p_loc);
END add_department;
END dept_pkg; /
```

ORACLE

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

Überladung – Beispiel

Wenn Sie add_department mit einer explizit angegebenen Abteilungsnummer aufrufen, verwendet PL/SQL die erste Version der Prozedur. Betrachten Sie das folgende Beispiel:

```
EXECUTE dept_pkg.add_department(980, 'Education', 2500)
SELECT * FROM departments
WHERE department_id = 980;
```

DEPARTMENT_ID	DEPARTMENT_NAME	MANAGER_ID	LOCATION_ID
980	Education		2500

Wenn Sie add_department jedoch ohne Abteilungsnummer aufrufen, verwendet PL/SQL die zweite Version:

```
EXECUTE dept_pkg.add_department ('Training', 2400)
SELECT * FROM departments
WHERE department_name = 'Training';
```

DEPARTMENT_ID	DEPARTMENT_NAME	MANAGER_ID	LOCATION_ID
350	Training		2400

Überladung und Package STANDARD

- Das Package STANDARD definiert die PL/SQL-Umgebung und die Built-In-Funktionen.
- Die meisten Built-In-Funktionen sind überladen. Ein Beispiel ist die Funktion TO_CHAR:

```
FUNCTION TO_CHAR (p1 DATE) RETURN VARCHAR2;
FUNCTION TO_CHAR (p2 NUMBER) RETURN VARCHAR2;
FUNCTION TO_CHAR (p1 DATE, p2 VARCHAR2) RETURN VARCHAR2;
FUNCTION TO_CHAR (p1 NUMBER, p2 VARCHAR2) RETURN
    VARCHAR2;
. . .
```

- Ein PL/SQL-Unterprogramm mit demselben Namen wie ein Built-In-Unterprogramm überschreibt die Standarddeklaration im lokalen Kontext, sofern sie nicht durch den Packagenamen angegeben ist.

ORACLE

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

Das Package STANDARD definiert die PL/SQL-Umgebung und deklariert Typen, Exceptions und Unterprogramme global, die PL/SQL-Programmen automatisch zur Verfügung stehen. Die meisten Built-In-Funktionen im Package STANDARD sind überladen. Beispiel: Die Funktion TO_CHAR hat vier verschiedene Deklarationen (siehe Folie). Die Funktion TO_CHAR kann den Datentyp DATE oder NUMBER annehmen und ihn in den Datentyp CHARACTER konvertieren. Das Format, in das der Datentyp DATE bzw. NUMBER zu konvertieren ist, lässt sich auch im Funktionsaufruf angeben.

Wenn Sie ein Built-In-Unterprogramm in einem anderen PL/SQL-Programm neu deklarieren, wird das Standard- bzw. Built-In-Unterprogramm mit der lokalen Deklaration überschrieben. Um auf das Built-In-Unterprogramm zugreifen zu können, müssen Sie es mit seinem Packagenamen angeben. Beispiel: Wird die Funktion TO_CHAR neu deklariert, ist die entsprechende Built-In-Funktion mit STANDARD.TO_CHAR aufzurufen.

Wenn Sie jedoch ein Built-In-Unterprogramm als Standalone-Unterprogramm neu deklarieren, müssen Sie es mit Ihrem Schemanamen aufrufen, beispielsweise SCOTT.TO_CHAR.

Auf der Folie löst PL/SQL einen Aufruf von TO_CHAR durch Abgleich von Anzahl und Datentypen der formalen und tatsächlichen Parameter auf.

Ungültige Prozedurreferenzen

- In blockstrukturierten Sprachen wie PL/SQL müssen Bezeichner, die referenziert werden sollen, zuerst deklariert werden.
- Beispiel für ein Referenzierungsproblem:

```
CREATE OR REPLACE PACKAGE BODY forward_pkg IS
  PROCEDURE award_bonus( . . . ) IS
    BEGIN
      calc_rating( . . . );      --illegal reference
    END;

  PROCEDURE calc_rating( . . . ) IS
    BEGIN
      ...
    END;
END forward_pkg;
/
```

ORACLE

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

Vorwärtsdeklarationen

Im Allgemeinen sind in PL/SQL wie in anderen blockstrukturierten Sprachen keine Vorwärtsreferenzen möglich. Sie müssen die Bezeichner erst deklarieren, bevor Sie sie verwenden können. Beispiel: Ein Unterprogramm kann erst aufgerufen werden, nachdem es deklariert wurde.

Oft verlangen die Codierungsstandards, dass Unterprogramme in alphabetischer Reihenfolge angeordnet werden, um sie leichter finden zu können. Dies kann zu Problemen führen, wie auf der Folie gezeigt: Die Prozedur `calc_rating` lässt sich nicht referenzieren, da sie noch nicht deklariert wurde.

Sie können das Problem der ungültigen Referenzierung lösen, indem Sie die Reihenfolge der zwei Prozeduren umkehren. Diese einfache Lösung funktioniert jedoch nicht, wenn die Codierungsregeln die Deklaration der Unterprogramme in alphabetischer Reihenfolge verlangen.

In diesem Fall lässt sich das Problem mit den in PL/SQL angebotenen Vorwärtsdeklarationen lösen. Mit Vorwärtsdeklarationen können Sie die Überschrift von Unterprogrammen deklarieren, d. h. die Unterprogrammspezifikation mit einem Semikolon als Abschluss.

Hinweis: Der Kompilierungsfehler für `calc_rating` tritt nur auf, wenn `calc_rating` eine private Packageprozedur ist. Wird `calc_rating` in der Packagespezifikation deklariert, ist die Deklaration bereits wie bei einer Vorwärtsdeklaration erfolgt, und der Compiler kann die Referenz auflösen.

Ungültige Prozedurreferenzen mithilfe von Vorwärtsdeklarationen auflösen

Im Packagebody ist eine Vorwärtsdeklaration die Spezifikation eines privaten Unterprogramms mit einem Semikolon als Abschluss.

```

CREATE OR REPLACE PACKAGE BODY forward_pkg IS
  PROCEDURE calc_rating (...) -- forward declaration
    -- Subprograms defined in alphabetical order

  PROCEDURE award_bonus (...) IS
  BEGIN
    calc_rating (...); -- reference resolved!
    . . .
  END;

  PROCEDURE calc_rating (...) IS -- implementation
  BEGIN
    . . .
  END;
END forward_pkg;

```

ORACLE

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

Vorwärtsdeklarationen

Wie schon erwähnt, können Sie mit PL/SQL spezielle Deklarationen von Unterprogrammen erstellen, die als Vorwärtsdeklarationen bezeichnet werden. Vorwärtsdeklarationen sind unter Umständen für private Unterprogramme im Packagebody erforderlich. Sie umfassen eine Unterprogrammspezifikation mit einem Semikolon als Abschluss. Mit Vorwärtsdeklarationen können Sie:

- Unterprogramme in logischer oder alphabetischer Reihenfolge definieren
- gegenseitig rekursive Unterprogramme definieren. Gegenseitig rekursive Programme sind Programme, die sich (direkt oder indirekt) gegenseitig aufrufen.
- Unterprogramme in einem Packagebody gruppieren und logisch organisieren

Beim Erstellen von Vorwärtsdeklarationen:

- müssen die formalen Parameter sowohl in der Vorwärtsdeklaration als auch im Unterprogrammbody angezeigt werden
- kann der Unterprogrammbody an beliebiger Stelle nach der Vorwärtsdeklaration stehen. Beide müssen jedoch in derselben Programmeinheit enthalten sein.

Vorwärtsdeklarationen und Packages

Normalerweise werden die Unterprogrammspezifikationen in die Packagespezifikation aufgenommen und die Unterprogrammbodys in den Packagebody. Für die Deklarationen von öffentlichen Unterprogrammen in der Packagespezifikation sind keine Vorwärtsdeklarationen erforderlich.

Packages initialisieren

Der Block am Ende des Packagebodys wird nur einmal ausgeführt und dient zum Initialisieren der öffentlichen und privaten Packagevariablen.

```

CREATE OR REPLACE PACKAGE taxes IS
    v_tax    NUMBER;
    ... -- declare all public procedures/functions
END taxes;
/
CREATE OR REPLACE PACKAGE BODY taxes IS
    ... -- declare all private variables
    ... -- define public/private procedures/functions
BEGIN
    SELECT    rate_value INTO v_tax
    FROM      tax_rates
    WHERE     rate_name = 'TAX';
END taxes;
/

```

ORACLE

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

Packageinitialisierungsblöcke

Beim ersten Referenzieren einer Komponente in einem Package wird das gesamte Package für die Benutzersession in den Speicher geladen. Der Ausgangswert von Variablen ist standardmäßig NULL (sofern nicht explizit initialisiert). Zum Initialisieren von Packagevariablen können Sie:

- Zuweisungsvorgänge in deren Deklarationen verwenden (einfache Initialisierung)
- einen Codeblock an das Ende eines Packagebodys anfügen (komplexere Initialisierung)

Betrachten Sie den Codeblock am Ende eines Packagebodys als Packageinitialisierungsblock, der einmal ausgeführt wird, wenn das Package zum ersten Mal in der Benutzersession aufgerufen wird.

Das Beispiel auf der Folie zeigt, wie die öffentliche Variable `v_tax` beim ersten Referenzieren des Packages `taxes` mit dem Wert in der Tabelle `tax_rates` initialisiert wird.

Hinweis: Wenn Sie die Variable in der Deklaration mit einem Zuweisungsvorgang initialisieren, wird sie vom Code im Initialisierungsblock am Ende des Packagebodys überschrieben. Der Initialisierungsblock des Packagebodys schließt mit dem Schlüsselwort `END` ab.

Packagefunktionen in SQL

- **Packagefunktionen werden in SQL-Anweisungen verwendet.**
- **Zur Ausführung einer SQL-Anweisung, die eine Member Function aufruft, muss die Oracle-Datenbank den Reinheitsgrad der Funktion kennen.**
- **Der Reinheitsgrad gibt an, inwieweit die Funktion frei von Seiteneffekten ist. Dies bezieht sich wiederum auf den Lese-/Schreibzugriff auf Datenbanktabellen, Packagevariablen usw.**
- **Das Ausschalten von Seiteneffekten ist wichtig, da sie:**
 - die korrekte Parallelisierung einer Abfrage verhindern
 - von der Reihenfolge abhängige (und damit nicht verlässliche) Ergebnisse liefern
 - nicht zulässige Aktionen erfordern, beispielsweise die Verwaltung des Packagezustands über mehrere Benutzersessions hinweg

ORACLE

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

Packagefunktionen in SQL – Verwendung und Einschränkungen

Zur Ausführung einer SQL-Anweisung, die eine Stored Function aufruft, muss der Oracle-Server den Reinheitsgrad der Funktion kennen. Er muss also wissen, inwieweit die Funktion frei von Seiteneffekten ist. Der Begriff *Seiteneffekt* bezieht sich auf den Lese-/Schreibzugriff auf Datenbanktabellen, Packagevariablen usw. Das Ausschalten von Seiteneffekten ist wichtig, da sie die korrekte Parallelisierung einer Abfrage verhindern, von der Reihenfolge abhängige (und damit nicht verlässliche) Ergebnisse liefern oder nicht zulässige Aktionen erfordern können, beispielsweise die Verwaltung des Packagezustands über mehrere Benutzersessions hinweg.

Unter Einschränkungen versteht man im Allgemeinen Änderungen an Datenbanktabellen oder an öffentlichen Packagevariablen (also Variablen, die in einer Packagespezifikation deklariert sind). Einschränkungen können die Ausführung einer Abfrage verzögern, von der Reihenfolge abhängige (und damit nicht verlässliche) Ergebnisse liefern oder erfordern, dass die Packagezustandsvariablen über mehrere Benutzersessions hinweg verwaltet werden. Verschiedene Einschränkungen sind nicht zulässig, wenn eine Funktion aus einer SQL-Abfrage oder DML-Anweisung aufgerufen wird.

Seiteneffekte von PL/SQL-Unterprogrammen ausschalten

Damit Stored Functions aus SQL-Anweisungen aufrufbar sind, müssen sie folgende Reinheitsregeln befolgen, um Seiteneffekte auszuschalten:

- Beim Aufrufen aus einer **SELECT**-Anweisung oder einer parallelisierten DML-Anweisung darf die Funktion keine Datenbanktabellen ändern.
- Beim Aufrufen aus einer DML-Anweisung darf die Funktion keine Datenbanktabellen abfragen oder ändern, die durch diese Anweisung geändert werden.
- Beim Aufrufen aus einer **SELECT**- oder DML-Anweisung darf die Funktion keine Steueranweisungen für SQL-Transaktionen, Sessions oder das System ausführen.



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

Je weniger Seiteneffekte eine Funktion aufweist, desto besser kann sie innerhalb einer Abfrage optimiert werden, insbesondere bei Verwendung der Hints `PARALLEL_ENABLE` oder `DETERMINISTIC`.

Damit Stored Functions (und alle Unterprogramme, die sie aufrufen) aus SQL-Anweisungen aufrufbar sind, müssen sie die auf der Folie aufgeführten Reinheitsregeln befolgen. Durch diese Regeln sollen Seiteneffekte ausgeschaltet werden.

Wenn eine SQL-Anweisung im Funktionsbody gegen eine Regel verstößt, erhalten Sie zur Laufzeit (beim Parsen der Anweisung) einen Fehler.

Um Code zur Komplizierungszeit auf Verstöße gegen Reinheitsregeln zu prüfen, bestätigen Sie mit dem Pragma `RESTRICT_REFERENCES`, dass keine Funktion Lese- oder Schreibvorgänge an Datenbanktabellen oder Packagevariablen ausführt.

Hinweis:

- Auf der Folie bezieht sich DML-Anweisung auf eine `INSERT`-, `UPDATE`- oder `DELETE`-Anweisung.
- Informationen zur Verwendung des Pragmas `RESTRICT_REFERENCES` finden Sie im Dokument *Oracle Database PL/SQL Language Reference*.

Packagefunktionen in SQL – Beispiel

```

CREATE OR REPLACE PACKAGE taxes_pkg IS
    FUNCTION tax (p_value IN NUMBER) RETURN NUMBER;
END taxes_pkg;
/
CREATE OR REPLACE PACKAGE BODY taxes_pkg IS
    FUNCTION tax (p_value IN NUMBER) RETURN NUMBER IS
        v_rate NUMBER := 0.08;
    BEGIN
        RETURN (p_value * v_rate);
    END tax;
END taxes_pkg;
/

```

```

SELECT taxes_pkg.tax(salary), salary, last_name
FROM   employees;

```

ORACLE

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

Im ersten Codebeispiel auf der Folie sehen Sie, wie die Packagespezifikation und der Body erstellt werden, wobei die Funktion **tax** im Package `taxes_pkg` gekapselt wird. Das zweite Codebeispiel zeigt, wie Sie die Packagefunktion **tax** in der `SELECT`-Anweisung aufrufen. Die Ergebnisse sehen wie folgt aus:

TAXES_PKG.TAX(SALARY)	SALARY	LAST_NAME
1920	24000	King
1360	17000	Kochhar
1360	17000	De Haan
720	9000	Hunold
480	6000	Ernst
384	4800	Austin
384	4800	Pataballa
336	4200	Lorentz
960.64	12008	Greenberg

109 rows selected

Lektionsagenda

- **Packageunterprogramme überladen, Vorwärts-deklarationen verwenden und Initialisierungsblöcke in einem Packagebody erstellen**
- **Persistente Zustände von Packagedaten für die Sessiondauer verwalten und assoziative Arrays (Index By-Tabellen) sowie Records in Packages verwenden**

ORACLE

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

Persistente Packagezustände

Der Packagezustand wird durch die Collection der Packagevariablen und die Werte definiert. Der Packagezustand:

- **wird initialisiert, wenn das Package zum ersten Mal geladen wird**
- **ist persistent (Standard) für die Dauer der Session:**
 - In der User Global Area (UGA) gespeichert
 - Eindeutig für jede Session
 - Veränderbar, wenn Packageunterprogramme aufgerufen oder öffentliche Variablen geändert werden
- **ist bei Verwendung von PRAGMA SERIALLY_REUSEABLE in der Packagespezifikation nicht für die Session, jedoch für die Dauer eines Unterprogrammaufrufs persistent**

ORACLE

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

Die Collection der öffentlichen und privaten Packagevariablen stellt den Packagezustand in der Benutzersession dar. Der Packagezustand ergibt sich somit aus den Werten, die zu einem bestimmten Zeitpunkt in allen Packagevariablen gespeichert sind. In der Regel gilt der Packagezustand für die Dauer der Benutzersession.

Packagevariablen werden initialisiert, wenn ein Package zum ersten Mal für eine Benutzersession in den Speicher geladen wird. Die Packagevariablen sind standardmäßig für jede Session eindeutig und behalten ihre Werte bei, bis die Benutzersession beendet wird. Die Variablen werden also im User Global Area-(UGA)-Speicher abgelegt, den die Datenbank für jede Benutzersession zuweist. Der Packagezustand ändert sich, wenn ein Packageunterprogramm aufgerufen wird und seine Logik den Variabenzustand ändert. Der Zustand eines öffentlichen Packages kann mit für den Packagetyp geeigneten Vorgängen direkt geändert werden.

PRAGMA zeigt an, dass es sich bei der Anweisung um eine Compileranweisung handelt. PRAGMAS werden zur Kompilierungszeit, nicht zur Laufzeit verarbeitet. Sie wirken sich nicht auf die Bedeutung eines Programms aus, sondern übermitteln lediglich Informationen an den Compiler. Wenn Sie PRAGMA SERIALLY_RESUSABLE zur Packagespezifikation hinzufügen, speichert die Datenbank die Packagevariablen in der System Global Area (SGA), die von allen Benutzersessions gemeinsam verwendet wird. In diesem Fall wird der Packagezustand für die Dauer eines Unterprogrammaufrufs oder einer einzelnen Referenz auf ein Packagekonstrukt beibehalten. Die SERIALLY_RESUSABLE-Anweisung ist nützlich, wenn Sie Speicher einsparen möchten und der Packagezustand nicht in jeder Benutzersession persistent sein muss.

Dieses PRAGMA eignet sich für Packages, die große, temporäre Arbeitsbereiche deklarieren, die nur einmal verwendet und während nachfolgenden Datenbankaufrufen in derselben Session nicht mehr benötigt werden.

Sie können ein Package ohne Body als seriell wiederverwendbar markieren. Wenn ein Package über eine Spezifikation und einen Body verfügt, müssen Sie beide markieren. Der Body allein lässt sich nicht markieren.

Der globale Speicher für seriell wiederverwendbare Packages wird in der System Global Area (SGA) in einen Pool gestellt und nicht in der User Global Area (UGA) einzelnen Benutzern zugewiesen. Auf diese Weise kann der Packagearbeitsbereich wiederverwendet werden. Wenn der Aufruf des Servers beendet ist, wird der Speicher an den Pool zurückgegeben. Bei jeder Wiederverwendung des Packages werden seine öffentlichen Variablen mit ihren Standardwerten oder NULL initialisiert.

Hinweis: Der Zugriff auf seriell wiederverwendbare Packages kann nicht über Datenbanktrigger oder andere PL/SQL-Unterprogramme erfolgen, die aus SQL-Anweisungen aufgerufen werden. Der Oracle-Server generiert sonst einen Fehler.

Persistente Zustände von Packagevariablen – Beispiel

Zustand für Scott Zustand für Jones

Zeit	Ereignisse	v_std_comm [variable]	MAX (commission_pct) [column]	v_std_comm [variable]	MAX (commission_pct) [Column]
9:00	Scott> EXECUTE comm_pkg.reset_comm(0.25)	0.10 0.25	0.4	-	0.4
9:30	Jones> INSERT INTO employees(last_name, commission_pct) VALUES('Madonna', 0.8);	0.25	0.4		0.8
9:35	Jones> EXECUTE comm_pkg.reset_comm (0.5)	0.25	0.4	0.10 0.5	0.8
10:00	Scott> EXECUTE comm_pkg.reset_comm(0.6) <i>Err -20210 'Bad Commission'</i>	0.25	0.4	0.5	0.8
11:00 11:01 12:00	Jones> ROLLBACK; EXIT ... EXEC comm_pkg.reset_comm(0.2)	0.25 0.25 0.25	0.4 0.4 0.4	0.5 - 0.2	0.4 0.4 0.4

ORACLE

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

Die Sequence auf der Folie basiert auf den beiden Benutzern Scott und Jones, die `comm_pkg` (wird in der Lektion "Packages erstellen" behandelt) ausführen. Darin ruft die Prozedur `reset_comm` die Funktion `validate` auf, um die neue Provision zu prüfen. Im Beispiel wird gezeigt, wie der persistente Zustand der Packagevariable `v_std_comm` in jeder Benutzersession beibehalten wird.

Um 09:00: Scott ruft `reset_comm` mit dem neuen Provisionswert 0.25 auf, der Packagezustand für `v_std_comm` wird mit 0.10 initialisiert und dann auf 0.25 eingestellt. Dies wird validiert, weil der Wert unter dem Datenbank-Maximalwert von 0.4 liegt.

Um 09:30: Jones fügt eine neue Zeile mit dem neuen Maximalwert 0.8 für `v_commission_pct` in die Tabelle `EMPLOYEES` ein. Dies wird nicht festgeschrieben und ist daher nur für Jones sichtbar. Der Zustand von Scott ist nicht betroffen.

Um 09:35: Jones ruft `reset_comm` mit dem neuen Provisionswert 0.5 auf. Der Zustand für `v_std_comm` von Jones wird zuerst mit 0.10 initialisiert und dann auf den neuen Wert 0.5 eingestellt. Dieser Wert ist für seine Session mit dem Datenbank-Maximalwert 0.8 gültig.

Um 10:00: Scott ruft `reset_comm` mit dem neuen Provisionswert 0.6 auf. Dieser Wert ist größer als der Maximalwert für die Provision in der Datenbank, der für diese Session sichtbar ist, nämlich 0.4. (Jones hat den Wert 0.8 nicht festgeschrieben.)

Zwischen 11:00 und 12:00: Jones rollt die Transaktion zurück (`INSERT`-Anweisung) und beendet die Session. Jones meldet sich um 11:45 an und führt die Prozedur erfolgreich aus. Sein Zustand wird auf 0.2 eingestellt.

Persistente Zustände von Packagecursorn – Beispiel

```
CREATE OR REPLACE PACKAGE curs_pkg IS -- Package spec
  PROCEDURE open;
  FUNCTION next(p_n NUMBER := 1) RETURN BOOLEAN;
  PROCEDURE close;
END curs_pkg;

CREATE OR REPLACE PACKAGE BODY curs_pkg IS
  -- Package body
  CURSOR cur_c IS
    SELECT employee_id FROM employees;
  PROCEDURE open IS
  BEGIN
    IF NOT cur_c%ISOPEN THEN
      OPEN cur_c;
    END IF;
  END open;
  . . . -- code continued on next slide
```

ORACLE

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

Im Beispiel auf der Folie werden die Spezifikation und der Body des Packages CURS_PKG gezeigt.
Die Body-Deklaration wird auf der nächsten Folie fortgesetzt.

Um dieses Package zu verwenden, verarbeiten Sie die Zeilen wie folgt:

1. Um den Cursor zu öffnen, rufen Sie die Prozedur open auf.

Persistente Zustände von Packagecursorn – Beispiel

```

. . .
FUNCTION next(p_n NUMBER := 1) RETURN BOOLEAN IS
    v_emp_id employees.employee_id%TYPE;
BEGIN
    FOR count IN 1 .. p_n LOOP
        FETCH cur_c INTO v_emp_id;
        EXIT WHEN cur_c%NOTFOUND;
        DBMS_OUTPUT.PUT_LINE('Id: ' || (v_emp_id));
    END LOOP;
    RETURN cur_c%FOUND;
END next;
PROCEDURE close IS
BEGIN
    IF cur_c%ISOPEN THEN
        CLOSE cur_c;
    END IF;
END close;
END curs_pkg;

```

ORACLE

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

2. Um eine Zeile oder eine angegebene Anzahl von Zeilen abzurufen, rufen Sie die Prozedur `next` auf. Wenn Sie mehr Zeilen anfordern, als tatsächlich vorhanden sind, behandelt die Prozedur den Abschluss erfolgreich.
Sie gibt `TRUE` zurück, wenn weitere Zeilen verarbeitet werden müssen. Andernfalls wird `FALSE` zurückgegeben.
3. Um den Cursor vor oder am Ende der Zeilenverarbeitung zu schließen, rufen Sie die Prozedur `close` auf.

Hinweis: Die Cursordeklaration erfolgt privat für das Package. Daher kann der Cursorzustand durch Aufrufen der auf der Folie aufgeführten Packageprozedur und -funktionen beeinflusst werden.

Package CURS_PKG ausführen

```

1 SET SERVEROUTPUT ON
2
3 EXECUTE curs_pkg.open
4 DECLARE
5   v_more BOOLEAN := curs_pkg.next(3);
6 BEGIN
7   IF NOT v_more THEN
8     curs_pkg.close;
9   END IF;
10 END;
11

```

Script Output:

```

anonymous block completed
anonymous block completed
Id: 100
Id: 101
Id: 102

anonymous block completed
anonymous block completed
Id: 103
Id: 104
Id: 105

```

ORACLE

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

Wie Sie bereits wissen, ist der Zustand einer Packagevariablen oder eines Cursors innerhalb einer Session transaktionsübergreifend persistent. Der Zustand persistiert jedoch nicht über verschiedene Sessions desselben Benutzers. Die Datenbanktabellen enthalten Daten, die über Sessions und Benutzer hinweg persistieren. Der Aufruf von `curs_pkg.open` öffnet den Cursor. Er bleibt geöffnet, bis die Session beendet oder der Cursor explizit geschlossen wird. Der anonyme Block führt die Funktion `next` im Bereich Declaration aus. Dabei wird die BOOLEAN-Variable `b_more` mit TRUE initialisiert, weil die Tabelle EMPLOYEES mehr als drei Zeilen umfasst. Der Block prüft das Ende der Ergebnismenge und schließt im entsprechenden Fall den Cursor. Wenn der Block ausgeführt wird, zeigt er die ersten drei Zeilen an:

```

Id :100
Id :101
Id :102

```

Wenn Sie erneut auf das Symbol **Run Script** (F5) klicken, werden die nächsten drei Zeilen angezeigt:

```

Id :103
Id :104
Id :105

```

Zum Schließen des Cursors können Sie den folgenden Befehl absetzen, um den Cursor im Package zu schließen oder die Session zu beenden:

```
EXECUTE curs_pkg.close
```

Assoziative Arrays in Packages

```
CREATE OR REPLACE PACKAGE emp_pkg IS
    TYPE emp_table_type IS TABLE OF employees%ROWTYPE
        INDEX BY BINARY_INTEGER;
    PROCEDURE get_employees(p_emps OUT emp_table_type);
END emp_pkg;
```

```
CREATE OR REPLACE PACKAGE BODY emp_pkg IS
    PROCEDURE get_employees(p_emps OUT emp_table_type) IS
        v_i BINARY_INTEGER := 0;
    BEGIN
        FOR emp_record IN (SELECT * FROM employees)
        LOOP
            p_emps(v_i) := emp_record;
            v_i:= v_i + 1;
        END LOOP;
    END get_employees;
END emp_pkg;
```

ORACLE

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

Assoziative Arrays wurden früher Index By-Tabellen genannt.

Das Package `emp_pkg` enthält die Prozedur `get_employees`. Sie liest Zeilen aus der Tabelle `EMPLOYEES` und gibt die Zeilen mit dem Parameter `OUT` zurück. Hierbei handelt es sich um ein assoziatives Array (PL/SQL-Record-Tabelle). Dabei gilt:

- `employee_table_type` wird als öffentlicher Typ deklariert.
- `employee_table_type` wird als formaler Ausgabeparameter in der Prozedur und die Variable `employees` im aufrufenden Block (siehe unten) verwendet.

Sie können die Prozedur `get_employees` in SQL Developer in einem anonymen PL/SQL-Block aufrufen, indem Sie die Variable `v_employees` verwenden, wie im folgenden Beispiel und in der folgenden Ausgabe gezeigt:

```
SET SERVEROUTPUT ON
DECLARE
    v_employees    emp_pkg.emp_table_type;
BEGIN
    emp_pkg.get_employees(v_employees);
    DBMS_OUTPUT.PUT_LINE('Emp 5: ' || v_employees(4).last_name);
END;
anonymous block completed
Emp 5: De Haan
```

Quiz

Die Überladung von Unterprogrammen in PL/SQL:

- a. ermöglicht das Erstellen mehrerer Unterprogramme mit identischem Namen
- b. erfordert eine unterschiedliche Anzahl, Reihenfolge oder Datentypfamilie der formalen Parameter des Unterprogramms
- c. ermöglicht das Erstellen flexibler Methoden zum Aufrufen von Unterprogrammen mit unterschiedlichen Daten
- d. bietet eine Möglichkeit zum Erweitern der Funktionalität, ohne vorhandenen Code zu verlieren, indem vorhandenen Unterprogrammen neue Parameter hinzugefügt werden

ORACLE

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

Richtige Antwort: a, b, c, d

Mit dem Überladungsfeature von PL/SQL können Sie zwei oder mehr Packageunterprogramme mit identischem Namen entwickeln. Die Überladung ist sinnvoll, wenn ein Unterprogramm ähnliche Parametergruppen mit unterschiedlichen Datentypen annehmen soll. Beispiel: Die Funktion TO_CHAR lässt sich auf mehrere Arten aufrufen. Dadurch können Sie eine Zahl oder ein Datum in eine Zeichenfolge konvertieren.

PL/SQL ermöglicht es Ihnen, Namen von Packageunterprogrammen und Methoden für Objekttypen zu überladen.

Grundsätzlich gilt, dass verschiedene Unterprogramme denselben Namen aufweisen können, solange sich ihre formalen Parameter durch die *Anzahl*, *Reihenfolge* oder *Datentypfamilie* unterscheiden.

Das Überladen empfiehlt sich in folgenden Situationen:

- Zwei oder mehr Unterprogramme weisen ähnliche Verarbeitungsregeln auf, unterscheiden sich jedoch durch den Typ oder die Anzahl der verwendeten Parameter.
- Es sollen alternative Verfahren für die Suche nach verschiedenen Daten mit unterschiedlichen Suchkriterien bereitgestellt werden. Beispiel: Sie möchten Mitarbeiter nach ihrer Personalnummer suchen, aber auch die Suche anhand des Nachnamens ermöglichen. Die eigentliche Logik ist identisch, die Parameter oder Suchkriterien unterscheiden sich jedoch.
- Die Funktionalität soll erweitert werden, ohne vorhandenen Code ersetzen zu müssen.

Zusammenfassung

In dieser Lektion haben Sie Folgendes gelernt:

- **Packageprozeduren und -funktionen überladen**
- **Vorwärtsdeklarationen verwenden**
- **Initialisierungsblöcke in einem Packagebody erstellen**
- **Persistente Zustände von Packagedaten für die Sessiondauer verwalten**
- **Assoziative Arrays (Index By-Tabellen) und Records in Packages verwenden**

ORACLE

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

Beim Überladen handelt es sich um ein Feature, mit dem Sie unterschiedliche Unterprogramme mit identischem Namen definieren können. Es ist sinnvoll, zwei Unterprogrammen den gleichen Namen zu geben, wenn ihre Verarbeitung identisch ist, die ihnen übergebenen Parameter jedoch variieren.

In PL/SQL ist eine spezielle Deklaration von Unterprogrammen zulässig, die als Vorwärts-deklaration bezeichnet wird. Mit Vorwärtsdeklarationen können Sie Unterprogramme in logischer oder alphabetischer Reihenfolge definieren, gegenseitig rekursive Unterprogramme definieren und Unterprogramme in einem Package gruppieren.

Ein Packageinitialisierungsblock wird nur ausgeführt, wenn das Package zum ersten Mal in der anderen Benutzersession aufgerufen wird. Dieses Feature gibt Ihnen die Möglichkeit, Variablen nur ein einziges Mal pro Session zu initialisieren.

Sie können den Zustand einer Packagevariablen oder eines Cursors überwachen, der für die Dauer der Benutzersession persistiert, und zwar von der ersten Referenzierung der Variablen oder des Cursors bis zur Beendigung der Session.

Mit PL/SQL Wrapper können Sie zum Schutz Ihres geistigen Eigentums den in der Datenbank gespeicherten Quellcode verdecken.

Übungen zu Lektion 5 – Überblick: Mit Packages arbeiten

Diese Übungen behandeln folgende Themen:

- **Überladene Unterprogramme verwenden**
- **Packageinitialisierungsblock erstellen**
- **Vorwärtsdeklaration verwenden**

ORACLE

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

Übungen zu Lektion 5 – Überblick

In diesen Übungen fügen Sie überladene Unterprogramme in ein vorhandenes Package ein und verwenden Vorwärtsdeklarationen. Außerdem erstellen Sie in einem Packagebody einen Packageinitialisierungsblock zum Füllen einer PL/SQL-Tabelle.



Von Oracle bereitgestellte Packages zur Anwendungsentwicklung

ORACLE

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

Ziele

Nach Ablauf dieser Lektion haben Sie folgende Ziele erreicht:

- **Funktionsweise des Packages DBMS_OUTPUT beschreiben**
- **Ausgabe mit UTL_FILE in Betriebssystemdateien leiten**
- **Hauptfeatures von UTL_MAIL beschreiben**



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

In dieser Lektion werden einige der von Oracle bereitgestellten Packages vorgestellt und ihre Funktionsmöglichkeiten erläutert.

Lektionsagenda

- **Vorteile der von Oracle bereitgestellten Packages bestimmen und einige Packages auflisten**
- Folgende von Oracle bereitgestellte Packages verwenden:
 - DBMS_OUTPUT
 - UTL_FILE
 - UTL_MAIL

ORACLE

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

Von Oracle bereitgestellte Packages

- **Die von Oracle bereitgestellten Packages:**
 - werden mit dem Oracle-Server geliefert
 - erweitern die Funktionalität der Datenbank
 - ermöglichen den Zugriff auf bestimmte SQL-Features, die normalerweise für PL/SQL eingeschränkt sind
- **Beispiel: Das Package DBMS_OUTPUT war ursprünglich zum Debuggen von PL/SQL-Programmen vorgesehen.**



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

Mit den vom Oracle-Server bereitgestellten Packages können Sie wahlweise:

- über PL/SQL auf bestimmte SQL-Features zugreifen
- die Funktionalität der Datenbank erweitern

Sie können die Funktionalität dieser Packages für das Erstellen von Anwendungen nutzen oder die Packages einfach als Anregung für eigene Stored Procedures verwenden.

Die meisten Standardpackages werden durch Ausführen von `catproc.sql` erstellt. Sie werden im Verlauf dieses Kurses vor allem das Package `DBMS_OUTPUT` kennenlernen. Falls Sie den Kurs *Oracle Database: PL/SQL Fundamentals* absolviert haben, dürften Sie mit diesem Package bereits vertraut sein.

Von Oracle bereitgestellte Packages – Beispiele

Auszug aus der Liste der von Oracle bereitgestellten Packages:

- **DBMS_OUTPUT**
- **UTL_FILE**
- **UTL_MAIL**
- **DBMS_ALERT**
- **DBMS_LOCK**
- **DBMS_SESSION**
- **DBMS_APPLICATION_INFO**
- **HTP**
- **DBMS_SCHEDULER**

ORACLE

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

Von Oracle bereitgestellte Packages – Liste

Mit jedem neuen Release wächst die Anzahl der mit einer Oracle-Datenbank bereitgestellten PL/SQL-Packages. In dieser Lektion werden die ersten drei Packages auf der Folie behandelt. Weitere Informationen finden Sie im Dokument *Oracle Database PL/SQL Packages and Types Reference*. Nachstehend eine Kurzbeschreibung der auf der Folie aufgeführten Packages:

- Mit DBMS_OUTPUT können Sie Textdaten debuggen und in den Puffer schreiben.
- Mit UTL_FILE können Sie Textdateien des Betriebssystems lesen und schreiben.
- UTL_MAIL unterstützt das Verfassen und Senden von E-Mail-Nachrichten.
- DBMS_ALERT unterstützt asynchrone Benachrichtigungen zu Datenbankereignissen. Nachrichten oder Alerts werden bei einem COMMIT-Befehl gesendet.
- Mit DBMS_LOCK können Sie Sperren über die Oracle Lock Management Services anfordern, konvertieren und freigeben.
- DBMS_SESSION ermöglicht die programmgesteuerte Verwendung der SQL-Anweisung ALTER SESSION und anderer Befehle auf Sessionebene.
- DBMS_APPLICATION_INFO kann mit Oracle Trace und der SQL-Trace-Funktion verwendet werden. Es dient der Aufzeichnung von Namen ausgeführter Module oder Transaktionen in der Datenbank, anhand derer die Performance verschiedener Module verfolgt und Debugging ausgeführt werden kann.

- Das Package HTP ermöglicht es Ihnen, Daten mit HTML-Tags in Datenbankpuffer zu schreiben.
- Mit DBMS_SCHEDULER können Sie die Ausführung von PL/SQL-Blöcken, Stored Procedures, externen Prozeduren und ausführbaren Programmen oder Routinen planen und automatisieren (wird in Anhang I behandelt).

Lektionsagenda

- Vorteile der von Oracle bereitgestellten Packages bestimmen und einige Packages auflisten
- **Folgende von Oracle bereitgestellte Packages verwenden:**
 - DBMS_OUTPUT
 - UTL_FILE
 - UTL_MAIL

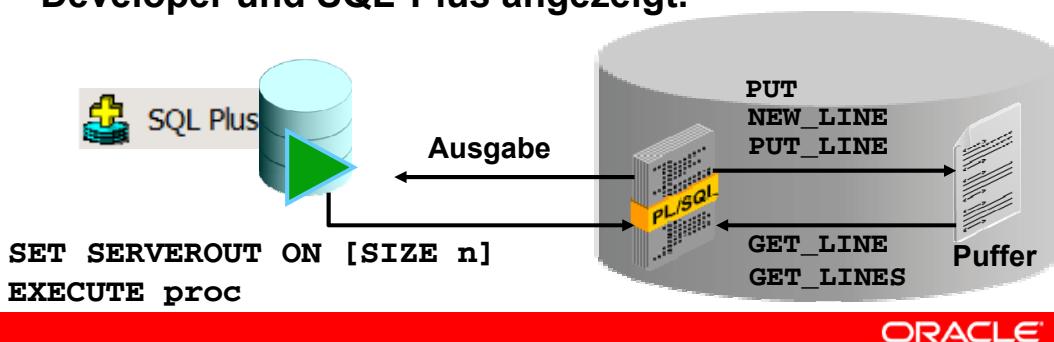
ORACLE

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

Package DBMS_OUTPUT – Funktionsweise

Mit dem Package DBMS_OUTPUT können Sie Nachrichten aus Stored Subprograms und Triggern senden.

- **PUT** und **PUT_LINE** schreiben Text in den Puffer.
- **GET_LINE** und **GET_LINES** lesen den Puffer.
- Nachrichten werden erst gesendet, nachdem die sendenden Unterprogramme oder Trigger abgeschlossen sind.
- Mit **SET SERVEROUTPUT ON** werden Nachrichten in SQL Developer und SQL*Plus angezeigt.



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

Das Package DBMS_OUTPUT sendet Textnachrichten von einem PL/SQL-Block in einen Puffer der Datenbank. Das Package stellt unter anderem folgende Prozeduren bereit:

- **PUT** hängt Text aus der Prozedur an die aktuelle Zeile des Zeilenausgabepuffers an.
- **NEW_LINE** setzt im Ausgabepuffer eine Zeilenendmarkierung.
- **PUT_LINE** schneidet mit einer Kombination aus den Aktionen **PUT** und **NEW_LINE** führende Leerzeichen ab.
- **GET_LINE** ruft die aktuelle Zeile aus dem Puffer in eine Prozedurvariable ab.
- **GET_LINES** ruft ein Zeilenarray in eine Arrayvariable einer Prozedur ab.
- **ENABLE/DISABLE** aktiviert und deaktiviert Aufrufe von DBMS_OUTPUT-Prozeduren.

Sie können die Puffergröße wie folgt einstellen:

- Mit der Option **SIZE n**, die an den Befehl **SET SERVEROUTPUT ON** angehängt wird, wobei **n** für die Anzahl der Zeichen steht. Die minimale Größe liegt bei 2.000, die maximale Größe ist unbegrenzt. Der Standardwert ist 20.000.
- Mit einem Ganzzahlparameter zwischen 2.000 und 1.000.000 in der Prozedur **ENABLE**

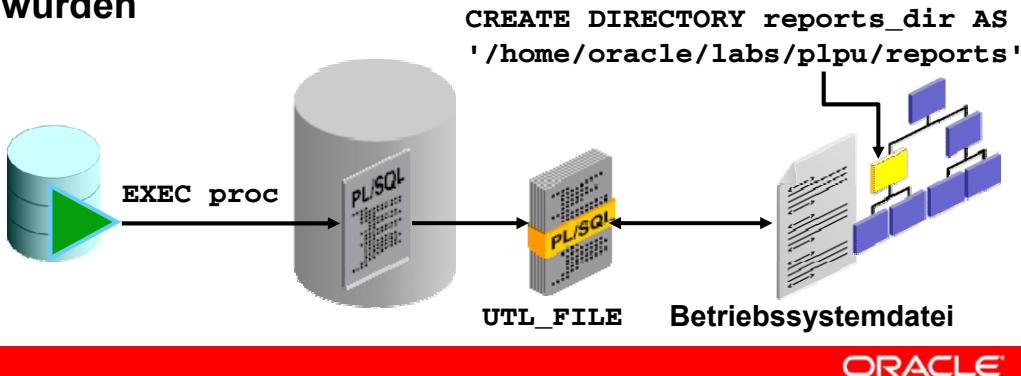
Sie können die Ergebnisse zum Debuggen in das Fenster ausgeben. Der Codeausführungspfad für eine Funktion oder Prozedur lässt sich verfolgen. Sie können Nachrichten zwischen Unterprogrammen und Triggern senden.

Hinweis: Es gibt keine Möglichkeit, die Ausgabe während der Ausführung einer Prozedur wegzuschreiben.

Mit dem Package UTL_FILE mit Betriebssystemdateien interagieren

Das Package UTL_FILE erweitert PL/SQL-Programme, sodass sie Textdateien des Betriebssystems lesen und schreiben können:

- **Bietet eine eingeschränkte Version von I/O in Streamdateien des Betriebssystems für Textdateien**
- **Kann auf Verzeichnisse des Betriebssystems zugreifen, die mit einer CREATE DIRECTORY-Anweisung definiert wurden**



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

Interaktion mit Betriebssystemdateien

Mit dem von Oracle bereitgestellten Package UTL_FILE greifen Sie im Betriebssystem des Datenbankservers auf Textdateien zu. Die Datenbank bietet folgendermaßen Lese- und Schreibzugriff auf bestimmte Verzeichnisse des Betriebssystems:

- Mit einer CREATE DIRECTORY-Anweisung, die einem Betriebssystemverzeichnis einen Alias zuordnet. Dem Alias des Datenbankverzeichnisses können die Berechtigungen READ und WRITE erteilt werden, um die Art des Zugriffs auf die Dateien des Betriebssystems zu steuern. Beispiel:

```
CREATE DIRECTORY my_dir AS '/temp/my_files';
GRANT READ, WRITE ON DIRECTORY my_dir TO public;
```

- Mit den Pfadangaben im Datenbank-Initialisierungsparameter `utl_file_dir`

Zur Verifizierung des Verzeichniszugriffs wird das Feature CREATE DIRECTORY statt UTL_FILE_DIR empfohlen. Directory-Objekte bieten dem Anwendungsadministrator von UTL_FILE mehr Flexibilität und granulare Kontrolle, können dynamisch verwaltet werden (ohne dass die Datenbank heruntergefahren werden muss) und sind mit anderen Oracle-Tools konsistent. Die Berechtigung CREATE DIRECTORY wird nur SYS und SYSTEM standardmäßig erteilt.

Die von diesen Verfahren angegebenen Betriebssystemverzeichnisse müssen sich auf denselben Server befinden wie die Datenbank-Serverprozesse. In Bezug auf die Pfadnamen (Verzeichnisnamen) ist bei manchen Betriebssystemen die Groß-/Kleinschreibung zu beachten.

Einige Prozeduren und Funktionen von UTL_FILE

Unterprogramm	Beschreibung
Funktion <code>ISOPEN</code>	Ermittelt, ob ein Datei-Handle eine geöffnete Datei referenziert
Funktion <code>FOPEN</code>	Öffnet Dateien für die Ein- oder Ausgabe
Funktion <code>FCLOSE</code>	Schließt alle geöffneten Datei-Handles
Prozedur <code>FCOPY</code>	Kopiert einen zusammenhängenden Teil einer Datei in eine neu erstellte Datei
Prozedur <code>FGETATTR</code>	Liest die Attribute einer Datenträgerdatei und gibt sie zurück
Prozedur <code>GET_LINE</code>	Liest Text aus einer geöffneten Datei
Prozedur <code>FREMOVE</code>	Löscht Datenträgerdateien, wenn Sie über entsprechende Berechtigungen verfügen
Prozedur <code>FRENAME</code>	Benennt vorhandene Dateien um
Prozedur <code>PUT</code>	Schreibt eine Zeichenfolge in eine Datei
Prozedur <code>PUT_LINE</code>	Schreibt eine Zeile in eine Datei und hängt so ein betriebssystemspezifisches Zeilenabschlusszeichen an

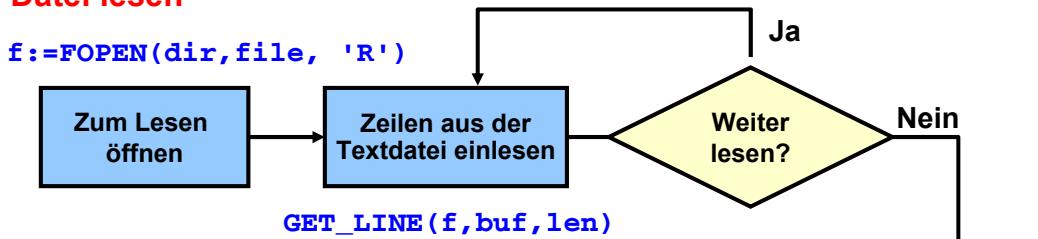
ORACLE

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

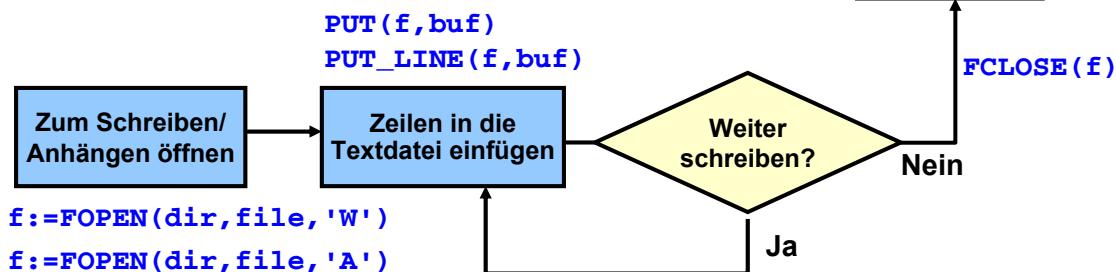
In der Tabelle auf der Folie sind einige der Unterprogramme des Packages `UTL_FILE` aufgeführt. Eine vollständige Liste der Unterprogramme des Packages finden Sie im Dokument *Oracle Database PL/SQL Packages and Types Reference*.

Dateibearbeitung mit dem Package UTL_FILE – Überblick

Datei lesen



Datei schreiben oder Text anhängen



ORACLE

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

Sie öffnen Dateien mit der Funktion `FOPEN` des Packages `UTL_FILE` und führen dann Bearbeitungsschritte wie Lesen, Schreiben oder Anhängen aus. Nach Abschluss der Bearbeitung schließen Sie sie die Datei mit der Prozedur `FCLOSE`. Nachstehend sind die Unterprogramme aufgeführt:

- Die Funktion `FOPEN` öffnet eine Datei in einem angegebenen Verzeichnis für I/O und gibt ein Datei-Handle zurück, das in nachfolgenden I/O-Vorgängen verwendet wird.
- Die Funktion `IS_OPEN` gibt einen booleschen Wert zurück, sobald ein Datei-Handle eine geöffnete Datei referenziert. Mit `IS_OPEN` können Sie prüfen, ob die Datei, die Sie öffnen möchten, bereits geöffnet ist.
- Die Prozedur `GET_LINE` liest eine Textzeile aus der Datei in einen Ausgabepufferparameter ein. (Die maximale Größe eines Eingabe-Records beträgt 1.023 Byte. Sie können in der überladenen Version von `FOPEN` jedoch einen höheren Wert angeben.)
- Die Prozeduren `PUT` und `PUT_LINE` schreiben Text in die geöffnete Datei.
- Die Prozedur `PUTF` stellt mit zwei Formatspezifikationen eine formatierte Ausgabe bereit: `%s` zum Ersetzen eines Wertes in der Ausgabezeichenfolge, `\n` für ein Zeilenvorschubzeichen.
- Die Prozedur `NEW_LINE` schließt eine Zeile in einer Ausgabedatei ab.
- Die Prozedur `FFLUSH` schreibt alle im Speicher gepufferten Daten in eine Datei.
- Die Prozedur `FCLOSE` schließt eine geöffnete Datei.
- Die Prozedur `FCLOSE_ALL` schließt alle für die Session geöffneten Datei-Handles.

Verfügbare deklarierte Exceptions im Package UTL_FILE

Name der Exception	Beschreibung
INVALID_PATH	Ungültiger Dateispeicherort
INVALID_MODE	Der Parameter <code>open_mode</code> in <code>FOPEN</code> ist ungültig.
INVALID_FILEHANDLE	Ungültiges Datei-Handle
INVALID_OPERATION	Datei konnte nicht geöffnet oder wie gewünscht bearbeitet werden.
READ_ERROR	Beim Lesevorgang ist ein Betriebssystemfehler aufgetreten.
WRITE_ERROR	Beim Schreibvorgang ist ein Betriebssystemfehler aufgetreten.
INTERNAL_ERROR	Nicht angegebener PL/SQL-Fehler

ORACLE

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

Das Package UTL_FILE deklariert fünfzehn Exceptions, die Fehlerbedingungen bei der Bearbeitung von Betriebssystemdateien anzeigen. Bei der Arbeit mit Unterprogrammen von UTL_FILE müssen Sie unter Umständen eine dieser Exceptions behandeln.

Ein Auszug der Exceptions ist auf der Folie zu sehen. Weitere Informationen zu den verbleibenden Exceptions finden Sie im Dokument *Oracle Database PL/SQL Packages and Types Reference*.

Hinweis: Exceptions muss immer der Packagename vorangestellt werden. Prozeduren von UTL_FILE können auch vordefinierte PL/SQL-Exceptions wie NO_DATA_FOUND oder VALUE_ERROR auslösen.

Die Exception NO_DATA_FOUND wird ausgelöst, wenn Sie mit UTL_FILE.GET_LINE oder UTL_FILE.GET_LINES über das Ende einer Datei hinaus lesen.

Funktionen **FOPEN** und **IS_OPEN** – Beispiel

- Die Funktion **FOPEN** öffnet eine Datei für die Ein- oder Ausgabe.

```
FUNCTION FOPEN (p_location  IN VARCHAR2,
               p_filename   IN VARCHAR2,
               p_open_mode  IN VARCHAR2)
RETURN UTL_FILE.FILE_TYPE;
```

- Die Funktion **IS_OPEN** bestimmt, ob ein Datei-Handle eine geöffnete Datei referenziert.

```
FUNCTION IS_OPEN (p_file IN FILE_TYPE)
RETURN BOOLEAN;
```

ORACLE

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

Die Parameter umfassen:

- Parameter **p_location**: Gibt einen mit der Anweisung `CREATE DIRECTORY` definierten Verzeichnisalias oder einen mit dem Datenbankparameter `utl_file_dir` angegebenen betriebssystemspezifischen Pfad an
- Parameter **p_filename**: Gibt den Namen der Datei mit Erweiterung ohne Pfadinformationen an
- Zeichenfolge **open_mode**: Gibt an, wie die Datei geöffnet werden soll. Verfügbare Werte:
 'r' Text lesen (GET_LINE verwenden)
 'w' Text schreiben (PUT, PUT_LINE, NEW_LINE, PUTF, FFLUSH)
 'A' Text anhängen (PUT, PUT_LINE, NEW_LINE, PUTF, FFLUSH)

Der Rückgabewert von **FOPEN** ist ein Datei-Handle vom Typ `UTL_FILE.FILE_TYPE`. Das Handle muss bei nachfolgenden Aufrufen von Routinen verwendet werden, die auf die geöffnete Datei zugreifen.

Der Funktionsparameter **IS_OPEN** ist das Datei-Handle. Die Funktion **IS_OPEN** testet ein Datei-Handle, um festzustellen, ob es eine geöffnete Datei identifiziert. Wenn die Datei geöffnet wurde, gibt die Funktion den booleschen Wert `TRUE` zurück. Andernfalls gibt sie den Wert `FALSE` zurück, was bedeutet, dass die Datei nicht geöffnet wurde. Das Beispiel auf der Folie zeigt die Kombination der beiden Unterprogramme. Eine vollständige Beschreibung der Syntax finden Sie im Dokument *Oracle Database PL/SQL Packages and Types Reference*.

Stellen Sie sicher, dass sich die externe Datei und die Datenbank auf demselben Rechner befinden.

```
CREATE OR REPLACE PROCEDURE read_file(p_dir VARCHAR2, p_filename
                                     VARCHAR2) IS
    f_file UTL_FILE.FILE_TYPE;
    buffer VARCHAR2(200);
    lines  PLS_INTEGER := 0;
BEGIN
    DBMS_OUTPUT.PUT_LINE(' Start ');
    IF NOT UTL_FILE.IS_OPEN(f_file) THEN
        DBMS_OUTPUT.PUT_LINE(' Open ');
        f_file := UTL_FILE.FOPEN (p_dir, p_filename, 'R');
        DBMS_OUTPUT.PUT_LINE(' Opened ');
        BEGIN
            LOOP
                UTL_FILE.GET_LINE(f_file, buffer);
                lines := lines + 1;
                DBMS_OUTPUT.PUT_LINE(TO_CHAR(lines, '099') || ' ' || buffer);
            END LOOP;
        EXCEPTION
            WHEN NO_DATA_FOUND THEN
                DBMS_OUTPUT.PUT_LINE(' ** End of File **');
        END;
        DBMS_OUTPUT.PUT_LINE(lines||' lines read from file');
        UTL_FILE.FCLOSE(f_file);
    END IF;
END read_file;
/
SHOW ERRORS
SET SERVEROUTPUT ON
EXECUTE read_file('REPORTS_DIR', 'instructor.txt')
```

Der vorstehende Code erzeugt die folgende Teilausgabe:

```
PROCEDURE READ_FILE compiled
No Errors.
anonymous block completed
Start
Open
Opened
001 SALARY REPORT: GENERATED ON
002                               08-MAR-01
003
004 DEPARTMENT: 10
005   EMPLOYEE: Whalen earns: 4400
006 DEPARTMENT: 20
007   EMPLOYEE: Hartstein earns: 13000
008   EMPLOYEE: Fay earns: 6000
009 DEPARTMENT: 30
```

```
...
120 DEPARTMENT: 110
121   EMPLOYEE: Higgins earns: 12000
122   EMPLOYEE: Gietz earns: 8300
123   EMPLOYEE: Grant earns: 7000
124 *** END OF REPORT ***
** End of File **
124 lines read from file
```

UTL_FILE – Beispiel

```

CREATE OR REPLACE PROCEDURE sal_status(
    p_dir IN VARCHAR2, p_filename IN VARCHAR2) IS
    f_file UTL_FILE.FILE_TYPE;
    CURSOR cur_emp IS
        SELECT last_name, salary, department_id
        FROM employees ORDER BY department_id;
        v_newdeptno employees.department_id%TYPE;
        v_olddeptno employees.department_id%TYPE := 0;
BEGIN
    f_file:= UTL_FILE.FOPEN (p_dir, p_filename, 'W');
    UTL_FILE.PUT_LINE(f_file,
        'REPORT: GENERATED ON ' || SYSDATE);
    UTL_FILE.NEW_LINE (f_file);
    ...

```

ORACLE

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

Im Beispiel auf der Folie erstellt die Prozedur `sal_status` einen Bericht über die Mitarbeiter der einzelnen Abteilungen und ihre Gehälter. Die Daten werden mithilfe des Packages `UTL_FILE` in eine Textdatei geschrieben. Im Codebeispiel wird die Variable `file` als `UTL_FILE.FILE_TYPE` deklariert. Dieser Packagetyp ist ein Record mit einem Feld namens `ID` mit dem Datentyp `BINARY_INTEGER`. Beispiel:

```
TYPE file_type IS RECORD (id BINARY_INTEGER);
```

Das Feld des Records `FILE_TYPE` ist innerhalb des Packages `UTL_FILE` privat und darf nicht referenziert oder geändert werden. Die Prozedur `sal_status` nimmt zwei Parameter an:

- `p_dir` für den Namen des Verzeichnisses, in das die Textdatei geschrieben werden soll
- `p_filename` zur Angabe des Dateinamens

Sie können die Prozedur beispielsweise wie folgt aufrufen (stellen Sie zuvor sicher, dass sich die externe Datei und die Datenbank auf demselben Rechner befinden):

```
EXECUTE sal_status('REPORTS_DIR', 'salreport2.txt')
```

Hinweis: Der Verzeichnispfad (`REPORTS_DIR`) muss in Großbuchstaben eingegeben werden, wenn es sich um einen mit der Anweisung `CREATE DIRECTORY` erstellten Verzeichnisalias handelt. Beim Lesen von Dateien in einer Schleife muss die Schleife beendet werden, wenn sie die Exception `NO_DATA_FOUND` erkennt. Die Ausgabe von `UTL_FILE` wird synchron gesendet. Eine `DBMS_OUTPUT`-Prozedur generiert die Ausgabe erst nach Abschluss der Prozedur.

UTL_FILE – Beispiel

```

    .
    .
    .
    FOR emp_rec IN cur_emp LOOP
        IF emp_rec.department_id <> v_olddeptno THEN
            UTL_FILE.PUT_LINE (f_file,
                'DEPARTMENT: ' || emp_rec.department_id);
            UTL_FILE.NEW_LINE (f_file);
        END IF;
        UTL_FILE.PUT_LINE (f_file,
            'EMPLOYEE: ' || emp_rec.last_name ||
            ' earns: ' || emp_rec.salary);
        v_olddeptno := emp_rec.department_id;
        UTL_FILE.NEW_LINE (f_file);
    END LOOP;
    UTL_FILE.PUT_LINE(f_file,'*** END OF REPORT ***');
    UTL_FILE.FCLOSE (f_file);
EXCEPTION
    WHEN UTL_FILE.INVALID_FILEHANDLE THEN
        RAISE_APPLICATION_ERROR(-20001,'Invalid File.');
    WHEN UTL_FILE.WRITE_ERROR THEN
        RAISE_APPLICATION_ERROR (-20002, 'Unable to write to file');
END sal_status;/
```

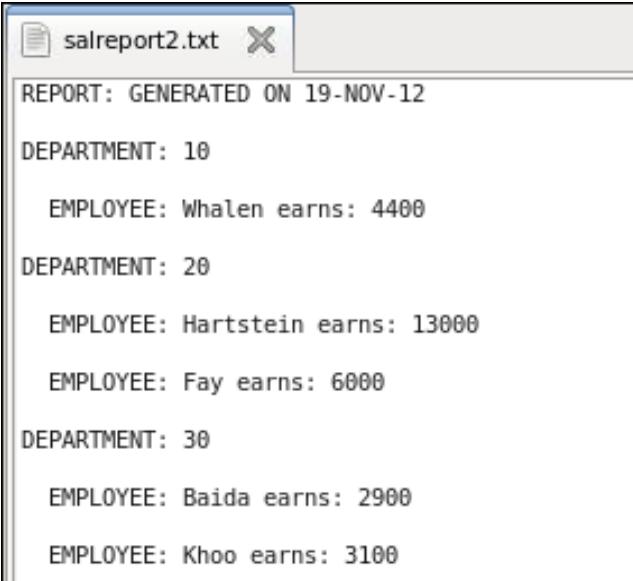
ORACLE

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

Führen Sie die Prozedur `sal_status` aus:

```
EXECUTE sal_status('REPORTS_DIR', 'salreport2.txt')
```

Im Folgenden ein Beispiel der Ausgabedatei **salreport2.txt**:



The screenshot shows a Windows Notepad window with the following content:

```

salreport2.txt

REPORT: GENERATED ON 19-NOV-12

DEPARTMENT: 10

EMPLOYEE: Whalen earns: 4400

DEPARTMENT: 20

EMPLOYEE: Hartstein earns: 13000

EMPLOYEE: Fay earns: 6000

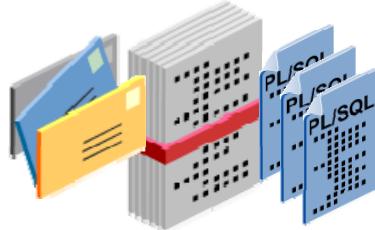
DEPARTMENT: 30

EMPLOYEE: Baida earns: 2900

EMPLOYEE: Khoo earns: 3100
  
```

Was ist das Package UTL_MAIL?

- Ein Utility zur E-Mail-Verwaltung
- Erfordert die Einstellung des Datenbank-Initialisierungsparameters **SMTP_OUT_SERVER**
- Stellt die folgenden Prozeduren bereit:
 - **SEND** für Nachrichten ohne Anhänge
 - **SEND_ATTACH_RAW** für Nachrichten mit binären Anhängen
 - **SEND_ATTACH_VARCHAR2** für Nachrichten mit Textanhängen



ORACLE

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

UTL_MAIL

Das Package UTL_MAIL ist ein Utility zur E-Mail-Verwaltung mit gängigen E-Mail-Features wie Anhänge, Cc, Bcc und Lesebestätigung.

Das Package UTL_MAIL wird aufgrund der erforderlichen Konfiguration von **SMTP_OUT_SERVER** und des damit einhergehenden Sicherheitsrisikos nicht standardmäßig installiert. Bei der Installation von UTL_MAIL sollten Sie Vorkehrungen treffen, um übermäßige Datenübertragungen über den mit **SMTP_OUT_SERVER** definierten Port zu verhindern. Um UTL_MAIL zu installieren, melden Sie sich in SQL*Plus als DBA-Benutzer an und führen folgende Skripte aus:

```
@$ORACLE_HOME/rdbms/admin/utlmail.sql  
@$ORACLE_HOME/rdbms/admin/prvtmail.plb
```

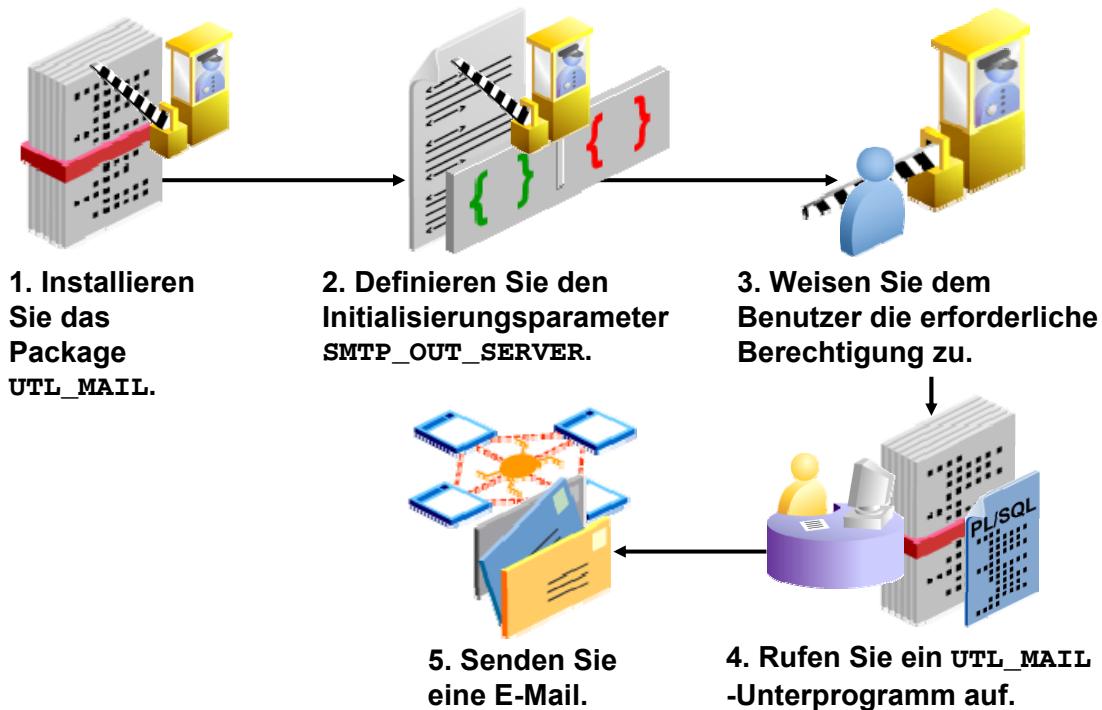
Definieren Sie den Parameter **SMTP_OUT_SERVER** in der Datenbank-Initialisierungsdatei **init.ora**:

```
SMTP_OUT_SERVER=mystmpserver.mydomain.com
```

Der Parameter `SMTP_OUT_SERVER` gibt den SMTP-Host und den Port an, an den `UTL_MAIL` ausgehende E-Mails leitet. Sie können mehrere Server angeben (durch Kommas getrennt). Ist der erste Server in der Liste nicht verfügbar, versucht `UTL_MAIL` den zweiten Server usw. Bei fehlender Definition von `SMTP_OUT_SERVER` wird eine aus `DB_DOMAIN` abgeleitete Standard-einstellung aufgerufen. Dies ist ein Datenbank-Initialisierungsparameter, der die logische Position der Datenbank innerhalb der Netzwerkstruktur angibt. Beispiel:

```
db_domain=mydomain.com
```

UTL_MAIL einrichten und verwenden – Überblick



ORACLE

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

In Oracle Database 11g ist das Package UTL_MAIL nun ein Package mit Rechten des ausführenden Benutzers, und der ausführende Benutzer benötigt die Berechtigung CONNECT. Diese wird in der Zugriffskontrollliste erteilt, die dem Remote-Netzwerkhost zugewiesen ist, mit dem der ausführende Benutzer eine Verbindung herstellen möchte. Der Sicherheitsadministrator führt diese Aufgabe durch.

Hinweis

- Informationen dazu, wie ein Benutzer mit SYSDBA-Berechtigungen einem anderen Benutzer die erforderlichen fein granulierten Berechtigungen zur Verwendung dieses Packages erteilt, finden Sie unter dem Thema "Managing Fine-Grained Access to External Network Services" im *Oracle Database Security Guide* und in dem von einem Dozenten gehaltenen Kurs *Oracle Database: Oracle Database Advanced PL/SQL*.
- Aufgrund von Firewalleinschränkungen können die UTL_MAIL-Beispiele in dieser Lektion nicht demonstriert werden. Daher wird das Package UTL_MAIL in den Übungen nicht verwendet.

UTL_MAIL-Unterprogramme – Zusammenfassung

Unterprogramm	Beschreibung
Prozedur SEND	E-Mail-Nachricht verpacken, SMTP-Informationen suchen und Nachricht dem SMTP-Server zur Weiterleitung an die Empfänger zustellen
Prozedur SEND_ATTACH_RAW	Stellt die für RAW-Anhänge überladene Prozedur SEND dar
Prozedur SEND_ATTACH_VARCHAR2	Stellt die für VARCHAR2-Anhänge überladene Prozedur SEND dar

ORACLE

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

UTL_MAIL installieren und verwenden

- Als **SYSDBA** mithilfe von **SQL Developer** oder **SQL*Plus**:
 - Package **UTL_MAIL** installieren

```
@?/rdbms/admin/utlmail.sql
@?/rdbms/admin/prvtmail.plb
```

- **SMTP_OUT_SERVER** einstellen

```
ALTER SYSTEM SET SMTP_OUT_SERVER='smtp.server.com'
SCOPE=SPFILE
```

- Als Entwickler eine **UTL_MAIL**-Prozedur aufrufen:

```
BEGIN
  UTL_MAIL.SEND('otn@oracle.com', 'user@oracle.com',
    message => 'For latest downloads visit OTN',
    subject => 'OTN Newsletter');
END;
```

ORACLE

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

Auf der Folie wird gezeigt, wie Sie den Parameter `SMTP_OUT_SERVER` auf den Namen des SMTP-Hosts in Ihrem Netzwerk konfigurieren und das (standardmäßig nicht installierte) Package `UTL_MAIL` installieren. Nach dem Ändern des Parameters `SMTP_OUT_SERVER` muss die Datenbankinstanz neu gestartet werden. Diese Aufgaben werden von einem Benutzer mit `SYSDBA`-Berechtigungen durchgeführt.

Das letzte Beispiel auf der Folie zeigt das einfachste Verfahren zum Senden einer Textnachricht über die Prozedur `UTL_MAIL.SEND` mit mindestens einem Betreff und einer Nachricht. Die zwei ersten obligatorischen Parameter sind:

- die E-Mail-Adresse des Absenders (`sender`), hier `otn@oracle.com`
- die E-Mail-Adresse der Empfänger (`recipients`), hier `user@oracle.com`. Der Wert kann eine durch Kommas getrennte Liste mit Adressen sein.

Die Prozedur `UTL_MAIL.SEND` stellt verschiedene andere Parameter wie `cc`, `bcc` und `priority` mit Standardwerten bereit, sofern nichts anderes angegeben wird. Im Beispiel gibt der Parameter `message` den Text der E-Mail an, und der Parameter `subject` enthält den Text für die Betreffzeile. Um HTML-Nachrichten mit HTML-Tags zu senden, fügen Sie den Parameter `mime_type` hinzu (zum Beispiel `mime_type=>'text/html'`).

Hinweis: Detaillierte Informationen zu allen Parametern der Prozedur `UTL_MAIL` finden Sie im Dokument *Oracle Database PL/SQL Packages and Types Reference*.

Prozedur SEND

Verpackt eine E-Mail-Nachricht in das entsprechende Format, sucht SMTP-Informationen und stellt die Nachricht dem SMTP-Server zur Weiterleitung an die Empfänger zu

```
UTL_MAIL.SEND (
    sender      IN      VARCHAR2 CHARACTER SET ANY_CS,
    recipients  IN      VARCHAR2 CHARACTER SET ANY_CS,
    cc          IN      VARCHAR2 CHARACTER SET ANY_CS
                      DEFAULT NULL,
    bcc         IN      VARCHAR2 CHARACTER SET ANY_CS
                      DEFAULT NULL,
    subject     IN      VARCHAR2 CHARACTER SET ANY_CS
                      DEFAULT NULL,
    message     IN      VARCHAR2 CHARACTER SET ANY_CS,
    mime_type   IN      VARCHAR2
                      DEFAULT 'text/plain; charset=us-ascii',
    priority    IN      PLS_INTEGER DEFAULT NULL);
```

ORACLE

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

Prozedur SEND

Diese Prozedur packt eine E-Mail-Nachricht in das entsprechende Format, sucht SMTP-Informationen und stellt die Nachricht dem SMTP-Server zur Weiterleitung an die Empfänger zu. Sie verbirgt die SMTP-API und stellt eine einzeilige, benutzerfreundliche E-Mail-Funktion zur Verfügung.

Parameter der Prozedur SEND

- **sender:** Die E-Mail-Adresse des Absenders
- **recipients:** Die E-Mail-Adressen der Empfänger (durch Kommas getrennt)
- **cc:** Die E-Mail-Adressen der Cc-Empfänger (durch Kommas getrennt). Der Standardwert ist NULL.
- **bcc:** Die E-Mail-Adressen der Bcc-Empfänger (durch Kommas getrennt). Der Standardwert ist NULL.
- **subject:** Die Zeichenfolge, die als E-Mail-Betreff aufgenommen werden soll. Der Standardwert ist NULL.
- **message:** Der Nachrichtentext
- **mime_type:** Der MIME-Typ der Nachricht. Der Standardwert ist 'text/plain; charset=us-ascii'.
- **priority:** Die Priorität der Nachricht. Der Standardwert ist NULL.

Prozedur SEND_ATTACH_RAW

Diese Prozedur ist die für RAW-Anhänge überladene Prozedur SEND.

```
UTL_MAIL.SEND_ATTACH_RAW (
    sender          IN      VARCHAR2 CHARACTER SET ANY_CS,
    recipients     IN      VARCHAR2 CHARACTER SET ANY_CS,
    cc              IN      VARCHAR2 CHARACTER SET ANY_CS DEFAULT NULL,
    bcc             IN      VARCHAR2 CHARACTER SET ANY_CS DEFAULT NULL,
    subject         IN      VARCHAR2 CHARACTER SET ANY_CS DEFAULT NULL,
    message         IN      VARCHAR2 CHARACTER SET ANY_CS DEFAULT NULL,
    mime_type       IN      VARCHAR2 DEFAULT CHARACTER SET ANY_CS
                            DEFAULT 'text/plain; charset=us-ascii',
    priority        IN      PLS_INTEGER DEFAULT 3,
    attachment     IN      RAW,
    att_inline      IN      BOOLEAN DEFAULT TRUE,
    att_mime_type   IN      VARCHAR2 CHARACTER SET ANY_CS
                            DEFAULT 'text/plain; charset=us-ascii',
    att_filename    IN      VARCHAR2 CHARACTER SET ANY_CS DEFAULT NULL);
```

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

Parameter der Prozedur SEND_ATTACH_RAW

- **sender:** Die E-Mail-Adresse des Absenders
- **recipients:** Die E-Mail-Adressen der Empfänger (durch Kommas getrennt)
- **cc:** Die E-Mail-Adressen der Cc-Empfänger (durch Kommas getrennt). Der Standardwert ist NULL.
- **bcc:** Die E-Mail-Adressen der Bcc-Empfänger (durch Kommas getrennt). Der Standardwert ist NULL.
- **subject:** Die Zeichenfolge, die als E-Mail-Betreff aufgenommen werden soll. Der Standardwert ist NULL.
- **message:** Der Nachrichtentext
- **mime_type:** Der MIME-Typ der Nachricht. Der Standardwert ist 'text/plain; charset=us-ascii'
- **priority:** Die Priorität der Nachricht. Der Standardwert ist NULL.
- **attachment:** Ein RAW-Anhang
- **att_inline:** Gibt an, ob der Anhang gemeinsam mit dem Nachrichtentext angezeigt werden kann. Der Standardwert ist TRUE.

E-Mails mit einem binären Anhang senden – Beispiel

```

CREATE OR REPLACE PROCEDURE send_mail_logo IS
BEGIN
    UTL_MAIL.SEND_ATTACH_RAW(
        sender => 'me@oracle.com',
        recipients => 'you@somewhere.net',
        message =>
            '<HTML><BODY>See attachment</BODY></HTML>',
        subject => 'Oracle Logo',
        mime_type => 'text/html'
        attachment => get_image('oracle.gif'),
        att_inline => true,
        att_mime_type => 'image/gif',
        att_filename => 'oralogo.gif');
END;
/

```

ORACLE

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

Auf der Folie wird eine Prozedur gezeigt, die die Prozedur `UTL_MAIL.SEND_ATTACH_RAW` aufruft, um eine Text- oder HTML-Nachricht mit einem binären Anhang zu senden. Neben den Parametern `sender`, `recipients`, `message`, `subject` und `mime_type`, die Werte für den Hauptteil der E-Mail-Nachricht bereitstellen, enthält die Prozedur `SEND_ATTACH_RAW` die folgenden hervorgehobenen Parameter:

- `attachment` (obligatorisch) nimmt einen `RAW`-Datentyp mit maximal 32.767 Binärzeichen an.
- `att_inline` (optional) ist ein boolescher Wert mit dem Standardwert `TRUE`. Der Parameter zeigt an, dass der Anhang gemeinsam mit dem Nachrichtentext angezeigt werden kann.
- `att_mime_type` (optional) gibt das Format des Anhangs an. Fehlt dieser Parameter, wird er auf `application/octet` eingestellt.
- `att_filename` (optional) weist dem Anhang einen Dateinamen zu. Der Standardwert ist `NULL`. In diesem Fall wird dem Anhang der Standardname zugewiesen.

Die Funktion `get_image` im Beispiel liest die Bilddaten mit einem `BFILE`. Um ein `BFILE` verwenden zu können, müssen Sie in der Datenbank mit der Anweisung `CREATE DIRECTORY` einen Namen für ein logisches Verzeichnis erstellen. Der Code für `get_image` wird auf der nächsten Seite gezeigt.

Die Funktion `get_image` liest mithilfe des Packages `DBMS_LOB` Binärdateien aus dem Betriebssystem:

```
CREATE OR REPLACE FUNCTION get_image(
    filename VARCHAR2, dir VARCHAR2 := 'TEMP')
RETURN RAW IS
    image RAW(32767);
    file BFILE := BFILENAME(dir, filename);
BEGIN
    DBMS_LOB.FILEOPEN(file, DBMS_LOB.FILE_READONLY);
    image := DBMS_LOB.SUBSTR(file);
    DBMS_LOB.CLOSE(file);
    RETURN image;
END;
/
```

Um das Verzeichnis namens `TEMP` zu erstellen, führen Sie die folgende Anweisung in SQL Developer oder SQL*Plus aus:

```
CREATE DIRECTORY temp AS 'd:\temp';
```

Hinweis

- Zur Ausführung dieser Anweisung benötigen Sie die Systemberechtigung `CREATE ANY DIRECTORY`.
- Aufgrund von Firewalleinschränkungen im Oracle Education Center können die Beispiele auf dieser und der vorherigen Seite nicht demonstriert werden.

Prozedur SEND_ATTACH_VARCHAR2

Diese Prozedur ist die für VARCHAR2-Anhänge überladene Prozedur SEND.

```
UTL_MAIL.SEND_ATTACH_VARCHAR2 (
    sender          IN  VARCHAR2 CHARACTER SET ANY_CS,
    recipients      IN  VARCHAR2 CHARACTER SET ANY_CS,
    cc              IN  VARCHAR2 CHARACTER SET ANY_CS DEFAULT NULL,
    bcc             IN  VARCHAR2 CHARACTER SET ANY_CS DEFAULT NULL,
    subject         IN  VARCHAR2 CHARACTER SET ANY_CS DEFAULT NULL,
    message         IN  VARCHAR2 CHARACTER SET ANY_CS DEFAULT NULL,
    mime_type       IN  VARCHAR2 CHARACTER SET ANY_CS
                           DEFAULT 'text/plain; charset=us-ascii',
    priority        IN  PLS_INTEGER DEFAULT 3,
    attachment      IN  VARCHAR2 CHARACTER SET ANY_CS,
    att_inline      IN  BOOLEAN DEFAULT TRUE,
    att_mime_type   IN  VARCHAR2 CHARACTER SET ANY_CS
                           DEFAULT 'text/plain; charset=us-ascii',
    att_filename    IN  VARCHAR2CHARACTER SET ANY_CS DEFAULT NULL);
```

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

Parameter der Prozedur SEND_ATTACH_VARCHAR2

- **sender:** Die E-Mail-Adresse des Absenders
- **recipients:** Die E-Mail-Adressen der Empfänger (durch Kommas getrennt)
- **cc:** Die E-Mail-Adressen der Cc-Empfänger (durch Kommas getrennt). Der Standardwert ist NULL.
- **bcc:** Die E-Mail-Adressen der Bcc-Empfänger (durch Kommas getrennt). Der Standardwert ist NULL.
- **subject:** Die Zeichenfolge, die als E-Mail-Betreff aufgenommen werden soll. Der Standardwert ist NULL.
- **Message:** Der Nachrichtentext
- **mime_type:** Der MIME-Typ der Nachricht. Der Standardwert ist 'text/plain; charset=us-ascii'
- **priority:** Die Priorität der Nachricht. Der Standardwert ist NULL.
- **attachment:** Der Textanhang
- **att_inline:** Gibt an, ob der Anhang inline ist. Der Standardwert ist TRUE.
- **att_mime_type:** Der MIME-Typ des Anhangs. Der Standardwert ist 'text/plain; charset=us-ascii'.
- **att_filename:** Die Zeichenfolge, die den Namen der Datei mit dem Anhang angibt. Der Standardwert ist NULL.

E-Mails mit einem Textanhang senden – Beispiel

```

CREATE OR REPLACE PROCEDURE send_mail_file IS
BEGIN
    UTL_MAIL.SEND_ATTACH_VARCHAR2(
        sender => 'me@oracle.com',
        recipients => 'you@somewhere.net',
        message =>
            '<HTML><BODY>See attachment</BODY></HTML>',
        subject => 'Oracle Notes',
        mime_type => 'text/html'
        attachment => get_file('notes.txt'),
        att_inline => false,
        att_mime_type => 'text/plain',
        att_filename => 'notes.txt');
END;
/

```

ORACLE

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

E-Mails mit einem Textanhang senden

Auf der Folie wird eine Prozedur gezeigt, die die Prozedur UTL_MAIL.SEND_ATTACH_VARCHAR2 aufruft, um eine Text- oder HTML-Nachricht mit einem Textanhang zu senden. Neben den Parametern `sender`, `recipients`, `message`, `subject` und `mime_type`, die Werte für den Hauptteil der E-Mail-Nachricht bereitstellen, enthält die Prozedur `SEND_ATTACH_VARCHAR2` die folgenden hervorgehobenen Parameter:

- `attachment` (obligatorisch) nimmt einen `VARCHAR2`-Datentyp mit maximal 32.767 Binärzeichen an.
- `att_inline` (optional) ist ein boolescher Wert mit dem Standardwert `TRUE`. Der Parameter zeigt an, dass der Anhang gemeinsam mit dem Nachrichtentext angezeigt werden kann.
- `att_mime_type` (optional) gibt das Format des Anhangs an. Fehlt dieser Parameter, wird er auf `application/octet` eingestellt.
- `att_filename` (optional) weist dem Anhang einen Dateinamen zu. Der Standardwert ist `NULL`. In diesem Fall wird dem Anhang der Standardname zugewiesen.

Die Funktion `get_file` im Beispiel liest mit `BFILE` den Wert des Parameters `attachment` aus einer Textdatei in den Betriebssystemverzeichnissen, der einfach von einer `VARCHAR2`-Variablen gefüllt werden könnte. Der Code für `get_file` wird auf der nächsten Seite gezeigt.

Die Funktion `get_file` liest mit dem Package `DBMS_LOB` eine Binärdatei des Betriebssystems und konvertiert die Binärdaten vom Typ `RAW` mit dem Package `UTL_RAW` in lesbare Textdaten vom Datentyp `VARCHAR2`:

```
CREATE OR REPLACE FUNCTION get_file(
    filename VARCHAR2, dir VARCHAR2 := 'TEMP')
RETURN VARCHAR2 IS
    contents VARCHAR2(32767);
    file BFILE := BFILENAME(dir, filename);
BEGIN
    DBMS_LOB.FILEOPEN(file, DBMS_LOB.FILE_READONLY);
    contents := UTL_RAW.CAST_TO_VARCHAR2(
        DBMS_LOB.SUBSTR(file));
    DBMS_LOB CLOSE(file);
    RETURN contents;
END;
/
```

Hinweis: Als Alternative können Sie den Inhalt der Textdatei mit der Funktionalität des Packages `UTL_FILE` in eine `VARCHAR2`-Variable einlesen.

Für das vorherige Beispiel muss das Verzeichnis `TEMP` ähnlich wie in der folgenden Anweisung in SQL*Plus erstellt werden:

```
CREATE DIRECTORY temp AS '/temp';
```

Hinweis

- Zur Ausführung dieser Anweisung benötigen Sie die Systemberechtigung `CREATE ANY DIRECTORY`.
- Aufgrund von Firewalleinschränkungen im Oracle Education Center können die Beispiele auf dieser und der vorherigen Seite nicht demonstriert werden.

Quiz

Mit dem von Oracle bereitgestellten Package UTL_FILE greifen Sie im Betriebssystem des Datenbankservers auf Textdateien zu.

Die Datenbank stellt Funktionalität über Directory-Objekte bereit, um Zugriff auf bestimmte Verzeichnisse des Betriebssystems zuzulassen.

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

ORACLE

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

Richtige Antwort: a

Mit dem von Oracle bereitgestellten Package UTL_FILE greifen Sie im Betriebssystem des Datenbankservers auf Textdateien zu. Die Datenbank bietet folgendermaßen Lese- und Schreibzugriff auf bestimmte Verzeichnisse des Betriebssystems:

- Mit einer CREATE DIRECTORY-Anweisung, die einem Betriebssystemverzeichnis einen Alias zuordnet. Dem Alias des Datenbankverzeichnisses können die Berechtigungen READ und WRITE erteilt werden, um die Art des Zugriffs auf die Dateien des Betriebssystems zu steuern.
- Mit den Pfadangaben im Datenbank-Initialisierungsparameter `utl_file_dir`

Zusammenfassung

In dieser Lektion haben Sie Folgendes gelernt:

- **Funktionsweise des Packages DBMS_OUTPUT**
- **Ausgabe mit UTL_FILE in Betriebssystemdateien leiten**
- **Hauptfeatures von UTL_MAIL**



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.

Ferner haben Sie in dieser Lektion gelernt, wie Sie mit UTL_FILE die leistungsstarken Features der Datenbank verwenden können, um Textdateien im Betriebssystem zu erstellen. Außerdem wurde erklärt, wie Sie mit dem Package UTL_MAIL E-Mails mit oder ohne binäre Anhänge oder Textanhänge senden.

Hinweis: Weitere Informationen zu allen PL/SQL-Packages und -Typen finden Sie im Dokument *PL/SQL Packages and Types Reference*.

Übungen zu Lektion 6 – Überblick: Von Oracle bereitgestellte Packages zur Anwendungsentwicklung

In diesen Übungen generieren Sie mit UTL_FILE einen Bericht in Form einer Textdatei.

ORACLE

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

Übungen zu Lektion 6 – Überblick

In diesen Übungen generieren Sie mit UTL_FILE einen Bericht über die Mitarbeiter der einzelnen Abteilungen in Form einer Textdatei.



Dynamisches SQL

ORACLE

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

Ziele

Nach Ablauf dieser Lektion haben Sie folgende Ziele erreicht:

- **Ausführungsablauf von SQL-Anweisungen beschreiben**
- **SQL-Anweisungen dynamisch mit nativem dynamischem SQL (NDS) erstellen und ausführen**
- **Situationen erkennen, in denen anstelle von NDS das Package DBMS_SQL verwendet werden muss, um SQL-Anweisungen dynamisch zu erstellen und auszuführen**



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

In dieser Lektion lernen Sie, SQL-Anweisungen dynamisch, d. h. zur Laufzeit, mit nativen dynamischen SQL-Anweisungen in PL/SQL zu erstellen und auszuführen.

Lektionsagenda

- **Natives dynamisches SQL (NDS) verwenden**
- **Package DBMS_SQL verwenden**

ORACLE

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

Ausführungsablauf von SQL-Anweisungen

- Alle SQL-Anweisungen durchlaufen einige oder alle der folgenden Phasen:
 - Parsen
 - Binden
 - Ausführen
 - Abrufen
- Bestimmte Phasen sind jedoch möglicherweise nicht für alle Anweisungen relevant:
 - Die Abruphase (Fetch) betrifft Abfragen.
 - Für eingebettete SQL-Anweisungen (`SELECT`, `DML`, `MERGE`, `COMMIT`, `SAVEPOINT` und `ROLLBACK`) erfolgt das Parsen und Binden während der Kompilierung.
 - Für dynamische SQL-Anweisungen werden alle Phasen zur Laufzeit ausgeführt.

ORACLE

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

SQL-Anweisungen verarbeiten – Vorgehensweise

Alle SQL-Anweisungen müssen verschiedene Phasen durchlaufen. Bestimmte Phasen sind jedoch möglicherweise nicht für alle Anweisungen relevant. Nachfolgend sind die wichtigsten Phasen aufgeführt:

- **Parsen:** Jede SQL-Anweisung muss geparsst werden. Das Parsen der Anweisung beinhaltet das Prüfen der Anweisungssyntax und das Validieren der Anweisung. Hierbei wird sichergestellt, dass alle Referenzen zu Objekten korrekt sind und die erforderlichen Berechtigungen für die Objekte vorhanden sind.
- **Binden:** Nach dem Parsen benötigt der Oracle-Server eventuell Werte von den oder für die Bind-Variablen in der Anweisung. Der Prozess zur Ermittlung dieser Werte wird als Binden von Variablen bezeichnet. Sie können diese Phase überspringen, wenn die Anweisung keine Bind-Variablen enthält.
- **Ausführen:** An diesem Punkt verfügt der Oracle-Server über alle notwendigen Informationen und Ressourcen, und die Anweisung wird ausgeführt. Sofern es sich nicht um Abfrageanweisungen handelt, ist dies die letzte Phase.
- **Abrufen:** In der Abruphase, die für Abfragen gilt, werden die Zeilen gewählt und sortiert (sofern von der Abfrage angefordert). In jedem der nachfolgenden Abrufe wird eine weitere Zeile des Ergebnisses geliefert, bis die letzte Zeile abgerufen wurde.

Mit dynamischem SQL arbeiten

Mithilfe von dynamischem SQL erstellen Sie SQL-Anweisungen, deren Struktur sich während der Laufzeit ändern kann. Dynamisches SQL:

- **wird innerhalb der Anwendung als Zeichenfolge, Zeichenfolgenvariable oder Zeichenfolgenausdruck erstellt und gespeichert**
- **ist eine SQL-Anweisung mit veränderbaren Spalten-daten oder unterschiedlichen Bedingungen mit oder ohne Platzhalter (Bind-Variablen)**
- **ermöglicht die Erstellung und Ausführung von DDL-, DCL- oder Sessionkontrollanweisungen über PL/SQL**
- **wird mit nativen dynamischen SQL-Anweisungen oder mit dem Package DBMS_SQL ausgeführt**



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

Dynamisches SQL

In PL/SQL sind nur die eingebetteten SQL-Anweisungen SELECT, INSERT, UPDATE, DELETE, MERGE, COMMIT und ROLLBACK verfügbar. All diese Anweisungen werden bei der Kompilierung geparsst, d. h. sie haben eine feste Struktur. Sie benötigen die Funktionalität von dynamischem SQL, wenn:

- die Struktur einer SQL-Anweisung zur Laufzeit geändert werden soll
- der Zugriff auf Data Definition Language-(DDL-)Anweisungen und andere SQL-Funktionen in PL/SQL benötigt wird

Um solche Aufgaben in PL/SQL durchzuführen, müssen Sie SQL-Anweisungen dynamisch in Zeichenfolgen erstellen und wahlweise wie folgt ausführen:

- In nativen dynamischen SQL-Anweisungen mit EXECUTE IMMEDIATE
- Mit dem Package DBMS_SQL

Der Prozess der Verwendung von SQL-Anweisungen, die nicht in Ihr Quellprogramm eingebettet sind und in Zeichenfolgen erstellt und zur Laufzeit ausgeführt werden, wird als "dynamisches SQL" bezeichnet. Die SQL-Anweisungen werden dynamisch zur Laufzeit erstellt und können auf PL/SQL-Variablen zugreifen und diese verwenden. Beispielsweise können Sie eine Prozedur erstellen, die mithilfe von dynamischem SQL eine Tabelle bearbeitet, deren Name erst zur Laufzeit bekannt ist. Sie können überdies DDL-Anweisungen (z. B. CREATE TABLE), Datenkontrollanweisungen (z. B. GRANT) oder Sessionkontrollanweisungen (z. B. ALTER SESSION) erstellen und ausführen.

Dynamisches SQL

- **Dynamisches SQL verwenden, wenn der vollständige Text der dynamischen SQL-Anweisung erst zur Laufzeit bekannt ist. Die Anweisungssyntax wird daher nicht bei der Kompilierung, sondern erst während der Laufzeit geprüft.**
- **Dynamisches SQL verwenden, wenn eines der folgenden Elemente zur Vorkompilierungszeit unbekannt ist:**
 - Text der SQL-Anweisung wie Befehle und Klauseln
 - Anzahl und Datentypen von Hostvariablen
 - Referenzen auf Datenbankobjekte wie Tabellen, Spalten, Indizes, Sequences, Benutzernamen und Views
- **Dynamisches SQL verwenden, um PL/SQL-Programme allgemeiner und flexibler zu machen**

ORACLE

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

In PL/SQL benötigen Sie dynamisches SQL zur Ausführung der folgenden SQL-Anweisungen, bei denen der vollständige Text zur Kompilierungszeit unbekannt ist:

- Eine SELECT-Anweisung mit einem Bezeichner, der bei der Kompilierung noch nicht vorliegt (z. B. ein Tabellename)
- Eine WHERE-Klausel, in der der Spaltenname bei der Kompilierung noch nicht vorliegt

Hinweis

Weitere Informationen über dynamisches SQL finden Sie in den folgenden Ressourcen:

- *Pro*C/C++ Programmer's Guide*
 - *Lesson 13, Oracle Dynamic SQL*, behandelt die vier verfügbaren Methoden zum Definieren von dynamischen SQL-Anweisungen. Die Funktionsmöglichkeiten und Beschränkungen der einzelnen Methoden werden kurz beschrieben. Anschließend folgen Richtlinien zur Wahl der richtigen Methode. In späteren Abschnitten desselben Handbuchs wird die Verwendung der Methoden erläutert. Dies erfolgt unter anderem anhand von Beispielprogrammen zum Selbststudium.
 - *Lesson 15, Oracle Dynamic SQL: Method 4*, enthält detaillierte Informationen über die 4. Methode zum Definieren dynamischer SQL-Anweisungen.
- Buch *Oracle PL/SQL Programming* von Steven Feuerstein und Bill Pribyl. *Lesson 16, Dynamic SQL and Dynamic PL/SQL*, bietet zusätzliche Informationen über dynamisches SQL.

Natives dynamisches SQL (NDS)

- **Bietet native Unterstützung für dynamisches SQL direkt in der PL/SQL-Sprache**
- **Ermöglicht die Ausführung von SQL-Anweisungen, deren Struktur bis zur Ausführungszeit unbekannt ist**
- **Handelt es sich bei der dynamischen SQL-Anweisung um eine SELECT-Anweisung, die mehrere Zeilen zurückgibt, haben Sie mit NDS mehrere Möglichkeiten:**
 - Anweisung **EXECUTE IMMEDIATE** mit der Klausel **BULK COLLECT INTO** verwenden
 - Anweisungen **OPEN-FOR**, **FETCH** und **CLOSE** verwenden
- **In Oracle Database 11g unterstützt NDS Anweisungen mit einer Größe von über 32 KB durch Annahme eines CLOB-Arguments.**

ORACLE

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

Natives dynamisches SQL ermöglicht die dynamische Ausführung von SQL-Anweisungen, deren Struktur zur Ausführungszeit erstellt wird. Zur Unterstützung von nativem dynamischem SQL wurden bestimmte Anweisungen in PL/SQL eingefügt oder erweitert:

- **EXECUTE IMMEDIATE:** Bereitet eine Anweisung vor, führt sie aus, gibt Variablen zurück und gibt anschließend Ressourcen frei
- **OPEN-FOR:** Bereitet Anweisungen vor und führt sie über eine Cursorvariable aus
- **FETCH:** Ruft die Ergebnisse geöffneter Anweisungen über die Cursorvariable ab
- **CLOSE:** Schließt den von der Cursorvariablen verwendeten Cursor und gibt die Ressourcen frei

Sie können Bind-Variablen in den dynamischen Parametern der Anweisungen **EXECUTE IMMEDIATE** und **OPEN** verwenden. Natives dynamisches SQL besitzt verschiedene Funktionsmöglichkeiten:

- Dynamische SQL-Anweisungen definieren
- Bind-Variablen **IN**, **IN OUT** und **OUT** verarbeiten, die nicht nach dem Namen, sondern nach der Position gebunden werden

Anweisung EXECUTE IMMEDIATE

Anweisung EXECUTE IMMEDIATE für NDS oder anonyme Blöcke in PL/SQL verwenden:

```
EXECUTE IMMEDIATE dynamic_string
[INTO {define_variable
      [, define_variable] ... | record}]
[USING [IN|OUT|IN OUT] bind_argument
      [, [IN|OUT|IN OUT] bind_argument] ... ],
```

- **INTO** wird für Single Row-Abfragen benutzt und gibt die Variablen oder Records an, in die Spaltenwerte abgerufen werden.
- **USING** enthält alle Bind-Argumente. Der Standardparametermodus ist **IN**.

ORACLE

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

Mithilfe der Anweisung EXECUTE IMMEDIATE können Sie SQL-Anweisungen oder anonyme Blöcke in PL/SQL ausführen. Dazu sind folgende syntaktische Elemente erforderlich:

- *dynamic_string* ist ein Zeichenfolgenausdruck, der für eine dynamische SQL-Anweisung (ohne Abschlusszeichen) oder für einen PL/SQL-Block (mit Abschlusszeichen) steht.
- *define_variable* ist eine PL/SQL-Variable, die den gewählten Spaltenwert speichert.
- *record* ist ein benutzerdefinierter oder %ROWTYPE-Record, der eine gewählte Zeile speichert.
- *bind_argument* ist ein Ausdruck, dessen Wert an die dynamische SQL-Anweisung oder den PL/SQL-Block übergeben wird.
- Die Klausel **INTO** gibt die Variablen oder den Record an, in die oder in den die Spaltenwerte abgerufen werden. Sie wird nur für Single Row-Abfragen verwendet. Für jeden durch die Abfrage abgerufenen Wert muss in der Klausel **INTO** eine entsprechende Variable oder ein entsprechendes Feld mit kompatiblem Typ vorhanden sein.
- Die Klausel **USING** enthält alle Bind-Argumente. Der Standardparametermodus ist **IN**.

Sie können numerische, Zeichen- und Zeichenfolgenliterale, jedoch keine booleschen Literale (TRUE, FALSE und NULL) als Bind-Argumente verwenden.

Hinweis: Verwenden Sie OPEN-FOR, FETCH und CLOSE nur bei Multiple Row-Abfragen. Die Syntax auf der Folie ist nicht vollständig, da auch die Verarbeitung großer Datenmengen (Bulk Processing) unterstützt wird. Dieses Thema wird jedoch in diesem Kurs nicht behandelt.

Verfügbare Methoden zur NDS-Verwendung

Methode	Typ der SQL-Anweisung	Verwendete NDS-SQL-Anweisungen
1. Methode	<i>Anderer Typ als Abfrage ohne Hostvariablen</i>	<code>EXECUTE IMMEDIATE</code> ohne die Klauseln <code>USING</code> und <code>INTO</code>
2. Methode	<i>Anderer Typ als Abfrage mit bekannter Anzahl von Eingabehostvariablen</i>	<code>EXECUTE IMMEDIATE</code> mit einer <code>USING-Klausel</code>
3. Methode	<i>Abfrage mit bekannter Anzahl von SELECT-Listenelementen und Eingabehostvariablen</i>	<code>EXECUTE IMMEDIATE</code> mit den Klauseln <code>USING</code> und <code>INTO</code>
4. Methode	<i>Abfrage mit unbekannter Anzahl von SELECT-Listenelementen oder Eingabehostvariablen</i>	Package <code>DBMS_SQL</code>

ORACLE

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

Die vier verfügbaren NDS-Methoden auf der Folie sind mit aufsteigender Reihenfolge zunehmend allgemein. Dies bedeutet, dass die 2. Methode die 1. Methode umfasst, die 3. Methode die 1. und 2. Methode und die 4. Methode die 1., 2. und 3. Methode. Allerdings bietet jede Methode Vorteile zur Bearbeitung eines bestimmten Typs von SQL-Anweisung, wie im Folgenden erläutert:

1. Methode

Mit dieser Methode kann das Programm eine dynamische SQL-Anweisung annehmen oder erstellen und dann gleich mithilfe des Befehls `EXECUTE IMMEDIATE` ausführen. Die SQL-Anweisung darf keine Abfrage (Anweisung `SELECT`) sein und keine Platzhalter für Eingabehostvariablen enthalten. Die nachfolgenden Hostzeichenfolgen geben beispielsweise Folgendes an:

- `DELETE FROM EMPLOYEES WHERE DEPTNO = 20`
- `GRANT SELECT ON EMPLOYEES TO scott`

Bei der 1. Methode wird die SQL-Anweisung bei jeder Ausführung geparst.

Hinweis

- Zu Anweisungen, die keine Abfragen sind, gehören beispielsweise Data Definition Language-(DDL-)Anweisungen und die Anweisungen `UPDATE`, `INSERT` und `DELETE`.
- Der Begriff *SELECT-Listenelement* umfasst Spaltennamen und Ausdrücke wie `SAL * 1.10` und `MAX(SAL)`.

2. Methode

Mit dieser Methode kann das Programm eine dynamische SQL-Anweisung annehmen oder erstellen und dann mithilfe der Befehle `PREPARE` und `EXECUTE` verarbeiten. Die SQL-Anweisung muss keine Abfrage sein. Die Anzahl der Platzhalter für Eingabehostvariablen und deren Datentypen müssen zur Vorkompilierungszeit bekannt sein. Die folgenden Hostzeichenfolgen fallen beispielsweise in diese Kategorie:

- `INSERT INTO EMPLOYEES (FIRST_NAME, LAST_NAME, JOB_ID) VALUES (:emp_first_name, :emp_last_name, :job_id)`
- `DELETE FROM EMPLOYEES WHERE EMPLOYEE_ID = :emp_number`

Bei der 2. Methode wird die SQL-Anweisung nur einmal geparsst, kann aber viele Male mit verschiedenen Werten für die Hostvariablen ausgeführt werden. SQL-Datendefinitionsanweisungen wie `CREATE` und `GRANT` werden ausgeführt, wenn sie vorbereitet (`prepared`) werden.

3. Methode

Mit dieser Methode kann das Programm eine dynamische Abfrage annehmen oder erstellen und dann mithilfe des Befehls `PREPARE` mit den Cursorbefehlen `DECLARE`, `OPEN`, `FETCH` und `CLOSE` verarbeiten. Die Anzahl der `SELECT`-Listenelemente, die Anzahl der Platzhalter für Eingabehostvariablen und die Datentypen der Eingabehostvariablen müssen zur Vorkompilierungszeit bekannt sein. Die folgenden Hostzeichenfolgen geben beispielsweise Folgendes an:

- `SELECT DEPARTMENT_ID, MIN(SALARY), MAX(SALARY)
FROM EMPLOYEES
GROUP BY DEPARTMENT_ID`
- `SELECT LAST_NAME, EMPLOYEE_ID
FROM EMPLOYEES
WHERE DEPARTMENT_ID = :dept_number`

4. Methode

Mit dieser Methode kann das Programm eine dynamische SQL-Anweisung annehmen oder erstellen und dann mithilfe von Deskriptoren verarbeiten. Ein Deskriptor ist ein Speicherbereich, der vom Programm und Oracle dazu verwendet wird, eine vollständige Beschreibung der Variablen in einer dynamischen SQL-Anweisung zu speichern. Die Anzahl der `SELECT`-Listenelemente, die Anzahl der Platzhalter für Eingabehostvariablen und die Datentypen der Eingabehostvariablen können bis zur Laufzeit unbekannt sein. Die folgenden Hostzeichenfolgen fallen beispielsweise in diese Kategorie:

- `INSERT INTO EMPLOYEES (<unknown>) VALUES (<unknown>)`
- `SELECT <unknown> FROM EMPLOYEES WHERE DEPARTMENT_ID = 20`

Die 4. Methode ist für dynamische SQL-Anweisungen erforderlich, die eine unbekannte Anzahl von `SELECT`-Listenelementen oder Eingabehostvariablen enthalten. Mit dieser Methode wird das Package `DBMS_SQL` verwendet, das später in dieser Lektion noch behandelt wird. Es gibt nur wenige Situationen, in denen die 4. Methode erforderlich ist.

Hinweis

Weitere Informationen über die vier Methoden für dynamisches SQL finden Sie in den folgenden Lektionen des *Pro*C/C++ Programmer's Guide*.

- *Lesson 13, Oracle Dynamic SQL*
- *Lesson 15, Oracle Dynamic SQL: Method 4*

Dynamisches SQL mit DDL-Anweisungen – Beispiele

```
-- Create a table using dynamic SQL

CREATE OR REPLACE PROCEDURE create_table(
    p_table_name VARCHAR2, p_col_specs VARCHAR2) IS
BEGIN
    EXECUTE IMMEDIATE 'CREATE TABLE ' || p_table_name ||
                      ' (' || p_col_specs || ')';
END;
/
```

```
-- Call the procedure

BEGIN
    create_table('EMPLOYEE_NAMES',
                 'id NUMBER(4) PRIMARY KEY, name VARCHAR2(40)');
END;
/
```

ORACLE

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

Die Codebeispiele zeigen die Erstellung einer `create_table`-Prozedur, die den Tabellennamen und die Spaltendefinitionen (Spezifikationen) als Parameter annimmt.

Der Prozederaufruf zeigt die Erstellung einer Tabelle namens `EMPLOYEE_NAMES` mit zwei Spalten:

- Eine ID-Spalte mit einem `NUMBER`-Datentyp als Primärschlüssel
- Eine Namensspalte mit bis zu 40 Zeichen für den Namen des Mitarbeiters

DDL-Anweisungen können mithilfe der auf der Folie gezeigten Syntax ausgeführt werden, unabhängig davon, ob sie dynamisch erstellt oder als Zeichenfolgenliteral angegeben werden. Sie können Anweisungen, die in einer PL/SQL-Zeichenfolgenvariablen gespeichert sind, entsprechend dem nachstehenden Beispiel erstellen und ausführen:

```
CREATE OR REPLACE PROCEDURE add_col(p_table_name VARCHAR2,
                                    p_col_spec    VARCHAR2) IS
    v_stmt VARCHAR2(100) := 'ALTER TABLE ' || p_table_name ||
                           ' ADD ' || p_col_spec;
BEGIN
    EXECUTE IMMEDIATE v_stmt;
END;
/
```

Um eine neue Spalte in eine Tabelle einzufügen, geben Sie Folgendes ein:

```
EXECUTE add_col('employee_names', 'salary number(8,2)')
```

Dynamisches SQL mit DML-Anweisungen

```
-- Delete rows from any table:
CREATE FUNCTION del_rows(p_table_name VARCHAR2)
RETURN NUMBER IS
BEGIN
    EXECUTE IMMEDIATE 'DELETE FROM ' || p_table_name;
    RETURN SQL%ROWCOUNT;
END;
/
BEGIN DBMS_OUTPUT.PUT_LINE(
    del_rows('EMPLOYEE_NAMES') || ' rows deleted.');
END;
/

-- Insert a row into a table with two columns:
CREATE PROCEDURE add_row(p_table_name VARCHAR2,
    p_id NUMBER, p_name VARCHAR2) IS
BEGIN
    EXECUTE IMMEDIATE 'INSERT INTO ' || p_table_name ||
        ' VALUES (:1, :2)' USING p_id, p_name;
END;
```

ORACLE

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

Das erste Codebeispiel auf der Folie definiert eine dynamische SQL-Anweisung mithilfe der 1. Methode, also mit einer Anweisung, die keine Abfrage ist und keine Hostvariablen enthält. Die Beispiele auf der Folie zeigen Folgendes:

- Die Funktion `del_rows` löscht Zeilen aus einer angegebenen Tabelle und gibt mit dem impliziten Attribut `%ROWCOUNT` des SQL-Cursors die Anzahl der gelöschten Zeilen zurück. Die Ausführung der Funktion ist unter dem Beispiel zum Erstellen einer Funktion dargestellt.
- Die Prozedur `add_row` zeigt, wie Eingabewerte für eine dynamische SQL-Anweisung mit der Klausel `USING` bereitgestellt werden. Die Namen der Bind-Variablen `:1` und `:2` sind unwichtig, die Reihenfolge der Parameternamen (`p_id` und `p_name`) in der Klausel `USING` wird den Bind-Variablen jedoch nach der Position in der Reihenfolge ihres Auftretens zugeordnet. Der PL/SQL-Parameter `p_id` wird daher dem Platzhalter `:1` zugeordnet, der Parameter `p_name` dem Platzhaltern oder Bind-Variablen können alphanumerisch sein. Ihnen muss jedoch ein Doppelpunkt vorangestellt werden.

Hinweis: Die Anweisung `EXECUTE IMMEDIATE` bereitet die dynamische SQL-Anweisung vor (durch Parsen) und führt sie sofort aus. Dynamische SQL-Anweisungen werden stets geparsst.

Beachten Sie auch, dass in keinem der Beispiele ein `COMMIT`-Vorgang erfolgt. Somit können die Vorgänge mit einer `ROLLBACK`-Anweisung rückgängig gemacht werden.

Dynamisches SQL mit Single Row-Abfragen – Beispiel

```

CREATE FUNCTION get_emp( p_emp_id NUMBER )
RETURN employees%ROWTYPE IS
    v_stmt VARCHAR2(200);
    v_emprec employees%ROWTYPE;
BEGIN
    v_stmt := 'SELECT * FROM employees ' ||
              'WHERE employee_id = :p_emp_id';
    EXECUTE IMMEDIATE v_stmt INTO v_emprec USING p_emp_id;
    RETURN v_emprec;
END;
/
DECLARE
    v_emprec employees%ROWTYPE := get_emp(100);
BEGIN
    DBMS_OUTPUT.PUT_LINE('Emp: ' || v_emprec.last_name);
END;
/

```

FUNCTION GET_EMP compiled
anonymous block completed
Emp: King

ORACLE

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

Der Code auf der Folie ist ein Beispiel für die Definition einer dynamischen SQL-Anweisung mithilfe der 3. Methode mit einer Single Row-Abfrage, also einer Abfrage mit einer bekannten Anzahl von SELECT-Listenelementen und Eingabehostvariablen.

Das Beispiel zur Single Row-Abfrage zeigt die Funktion `get_emp`, die einen EMPLOYEES-Record in eine in der Klausel `INTO` angegebene Variable abruft. Außerdem wird gezeigt, wie Eingabewerte für die Klausel `WHERE` bereitgestellt werden.

Mit dem anonymen Block wird die Funktion `get_emp` ausgeführt und das Ergebnis in eine lokale EMPLOYEES Record-Variable zurückgegeben.

Das Beispiel könnte so erweitert werden, dass je nach Eingabeparameterwerten alternative WHERE-Klauseln bereitgestellt werden, um es für die dynamische SQL-Verarbeitung besser geeignet zu machen.

Hinweis

- Ein Beispiel zum Thema dynamisches SQL mit Multiple Row-Abfrage, in dem REF CURSORS verwendet werden, finden Sie in `demo_07_13_a` im Ordner `/home/oracle/labs/plpu/demo`.
- Ein Beispiel zur Verwendung von REF CURSORS finden Sie in `demo_07_13_b` im Ordner `/home/oracle/labs/plpu/demo`.
- REF CURSORS werden im Kurs *Oracle Database: Advanced PL/SQL* behandelt.

Anonyme PL/SQL-Blöcke dynamisch ausführen

```

CREATE FUNCTION annual_sal( p_emp_id NUMBER)
RETURN NUMBER IS
    v_plsql varchar2(200) :=
        'DECLARE |||
        ' rec_emp employees%ROWTYPE; ||
        'BEGIN ||
        '    rec_emp := get_emp(:empid); ||
        2   ':res := rec_emp.salary * 12; ||
        'END;';
    v_result NUMBER;
BEGIN
    EXECUTE IMMEDIATE v_plsql
        USING IN p_emp_id, OUT v_result;
    RETURN v_result;
END;
/
EXECUTE DBMS_OUTPUT.PUT_LINE(annual_sal(100))

```

ORACLE

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

PL/SQL-Blöcke dynamisch ausführen

Die Funktion `annual_sal` erstellt dynamisch einen anonymen PL/SQL-Block. Der PL/SQL-Block enthält Bind-Variablen für:

- die Eingabe der Personalnummer mit dem Platzhalter `:empid`
- das Ausgabeergebnis mit der Berechnung des Jahresgehalts der Mitarbeiter mit dem Platzhalter `:res`

Hinweis: Dieses Beispiel zeigt, wie Sie mithilfe der Ergebnissyntax `OUT` (in der Klausel `USING` der Anweisung `EXECUTE IMMEDIATE`) das von dem PL/SQL-Block errechnete Ergebnis abrufen. Sie können die AusgabevARIABLEN der Prozedur und die Rückgabewerte der Funktion auf ähnliche Weise von einem dynamisch ausgeführten, anonymen PL/SQL-Block abrufen.

Das Beispiel auf der Folie generiert folgende Ausgabe:

```

FUNCTION ANNUAL_SAL compiled
anonymous block completed
288000

```

PL/SQL-Code mit nativem dynamischem SQL kompilieren

PL/SQL-Code mit der Anweisung ALTER kompilieren:

- **ALTER PROCEDURE name COMPILE**
- **ALTER FUNCTION name COMPILE**
- **ALTER PACKAGE name COMPILE SPECIFICATION**
- **ALTER PACKAGE name COMPILE BODY**

```
CREATE PROCEDURE compile_plsql(p_name VARCHAR2,
    p_plsql_type VARCHAR2, p_options VARCHAR2 := NULL) IS
    v_stmt varchar2(200) := 'ALTER '|| p_plsql_type |||
                           ' '||| p_name ||| 'COMPILE';
BEGIN
    IF p_options IS NOT NULL THEN
        v_stmt := v_stmt || ' ' ||| p_options;
    END IF;
    EXECUTE IMMEDIATE v_stmt;
END;
/
```

ORACLE

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

Mit der im Beispiel aufgeführten Prozedur `compile_plsql` können Sie unterschiedlichen PL/SQL-Code mithilfe der DDL-Anweisung `ALTER` kompilieren. Es werden vier Grundformen der Anweisung `ALTER` gezeigt, mit denen sich Folgendes kompilieren lässt:

- Prozedur
- Funktion
- Packagespezifikation
- Packagebody

Hinweis: Wenn Sie in der Anweisung `ALTER PACKAGE` die Schlüsselwörter `SPECIFICATION` und `BODY` weglassen, werden Spezifikation und Body kompiliert.

Beispiele für den Prozederaufruf in den vier auf der Folie genannten Fällen:

```
EXEC compile_plsql ('list_employees', 'procedure')
EXEC compile_plsql ('get_emp', 'function')
EXEC compile_plsql ('mypack', 'package', 'specification')
EXEC compile_plsql ('mypack', 'package', 'body')
```

Beispiel zur Kompilierung, wobei `debug` für die Funktion `get_emp` aktiviert ist:

```
EXEC compile_plsql ('get_emp', 'function', 'debug')
```

Lektionsagenda

- Natives dynamisches SQL (NDS) verwenden
- Package `DBMS_SQL` verwenden

ORACLE

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

Package DBMS_SQL

- Das Package **DBMS_SQL** wird verwendet, um dynamisches SQL in Stored Procedures zu erstellen und DDL-Anweisungen zu parsen.
- Zur Ausführung einer dynamischen SQL-Anweisung mit einer unbekannten Anzahl von Eingabe- oder Ausgabevariablen (4. Methode) benötigen Sie das Package **DBMS_SQL**.
- In den meisten Fällen (4. Methode ausgenommen) ist NDS benutzerfreundlicher und erzielt eine bessere Performance als **DBMS_SQL**.
- Das Package **DBMS_SQL** ist beispielsweise in folgenden Situationen erforderlich:
 - Die Liste **SELECT** ist bei der Kompilierung noch nicht bekannt.
 - Die Anzahl der von einer **SELECT**-Anweisung zurückgegebenen Spalten oder ihre Datentypen sind nicht bekannt.

ORACLE

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

Mit **DBMS_SQL** lassen sich Stored Procedures und anonyme PL/SQL-Blöcke erstellen, in denen dynamisches SQL verwendet wird. Sie können z. B. DDL-Anweisungen in PL/SQL ausführen (beispielsweise eine **DROP TABLE**-Anweisung). Die Vorgänge dieses Packages werden unter dem aktuellen Benutzer, nicht dem Packageeigentümer **SYS** durchgeführt.

4. Methode: Die 4. Methode bezieht sich auf Situationen, in denen bei einer dynamischen SQL-Anweisung die Anzahl der für eine Abfrage gewählten Spalten oder die Anzahl der eingestellten Bind-Variablen erst zur Laufzeit bekannt ist. Verwenden Sie in derartigen Fällen das Package **DBMS_SQL**.

Bei der Generierung von dynamischem SQL können Sie entweder das mitgelieferte Package **DBMS_SQL** (für Situationen der 4. Methode) oder natives dynamisches SQL verwenden. Vor Oracle Database 11g waren beide Methoden mit funktionalen Einschränkungen verbunden. In Oracle Database 11g werden dagegen nun beide Möglichkeiten umfassend unterstützt.

Die Features zur Ausführung von dynamischem SQL aus PL/SQL waren in Oracle Database 10g mit bestimmten Einschränkungen verbunden. **DBMS_SQL** wurde für Szenarios der 4. Methode benötigt, konnte jedoch nicht alle Datentypen verarbeiten, und die entsprechende Cursordarstellung konnte nicht von Datenbankclients verwendet werden. Natives dynamisches SQL war für andere Szenarios komfortabler, unterstützte jedoch maximal Anweisungen bis zu einer Größe von 32 KB. Mit Oracle Database 11g fallen diese und andere Einschränkungen weg, sodass dynamisches SQL aus PL/SQL nun umfassend unterstützt wird.

Unterprogramme des Packages DBMS_SQL

Beispiele für Prozeduren und Funktionen des Packages:

- **OPEN_CURSOR**
- **PARSE**
- **BIND_VARIABLE**
- **EXECUTE**
- **FETCH_ROWS**
- **CLOSE_CURSOR**

ORACLE

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

Das Package DBMS_SQL stellt folgende Unterprogramme zur Ausführung von dynamischem SQL bereit:

- OPEN_CURSOR öffnet einen neuen Cursor und gibt eine Cursor-ID zurück.
- PARSE parst die SQL-Anweisung. Jede SQL-Anweisung muss durch einen Aufruf der PARSE-Prozeduren geparsst werden. Dabei wird die Syntax der Anweisung geprüft und die Anweisung mit dem Cursor im Programm verknüpft. Sie können jede DML- oder DDL-Anweisung parsen. DDL-Anweisungen werden beim Parsen sofort ausgeführt.
- BIND_VARIABLE bindet einen gegebenen Wert an eine Bind-Variable, die in der geparssten Anweisung mit ihrem Namen identifiziert wird. Dies ist nicht erforderlich, wenn die Anweisung keine Bind-Variablen enthält.
- EXECUTE führt die SQL-Anweisung aus und gibt die Anzahl der verarbeiteten Zeilen zurück.
- FETCH_ROWS ruft bei einer Abfrage die nächste Zeile ab (bei mehreren Zeilen in einer Schleife zu verwenden).
- CLOSE_CURSOR schließt den angegebenen Cursor.

Hinweis: Wenn Sie mit dem Package DBMS_SQL DDL-Anweisungen ausführen, kann es zu einem Deadlock kommen. Die wahrscheinlichste Ursache hierfür ist, dass das Package verwendet wird, um eine Prozedur zu löschen, die Sie noch benutzen.

Parameter der Prozedur PARSE

Über den Parameter LANGUAGE_FLAG der Prozedur PARSE wird festgelegt, wie Oracle die SQL-Anweisung verarbeitet. Die genaue Verhaltensweise richtet sich nach der Version der Oracle-Datenbank. Der Parameterwert NATIVE (oder 1) bezeichnet die normale Verhaltensform der Datenbank, mit der das Programm verbunden ist.

Ist für den Parameter LANGUAGE_FLAG der Wert V6 (oder 0) eingestellt, bezieht sich dies auf das Verhalten der Oracle-Datenbank, Version 6, bei V7 (oder 2) auf Version 7.

Hinweis: Weitere Informationen finden Sie im Dokument *Oracle Database PL/SQL Packages and Types Reference*.

DBMS_SQL mit DML-Anweisungen – Zeilen löschen

```

CREATE OR REPLACE FUNCTION delete_all_rows
  (p_table_name  VARCHAR2) RETURN NUMBER IS
    v_cur_id      INTEGER;
    v_rows_del    NUMBER;
BEGIN
  v_cur_id := DBMS_SQL.OPEN_CURSOR;
  DBMS_SQLPARSE(v_cur_id,
    'DELETE FROM ' || p_table_name, DBMS_SQL.NATIVE);
  v_rows_del := DBMS_SQL.EXECUTE (v_cur_id);
  DBMS_SQL.CLOSE_CURSOR(v_cur_id);
  RETURN v_rows_del;
END;
/

```

```

CREATE TABLE temp_emp AS SELECT * FROM employees;
BEGIN
  DBMS_OUTPUT.PUT_LINE('Rows Deleted: ' ||
delete_all_rows('temp_emp'));
END;
/

```

ORACLE

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

DBMS_SQL mit DML-Anweisungen

Auf der Folie wird der Tabellenname an die Funktion `delete_all_rows` übergeben. Die Funktion löscht mithilfe von dynamischem SQL Zeilen aus der angegebenen Tabelle und gibt die Anzahl der Zeilen zurück, die nach erfolgreicher Ausführung der Anweisung gelöscht wurden.

Um eine DML-Anweisung dynamisch zu verarbeiten, gehen Sie wie folgt vor:

1. Legen Sie mit `OPEN_CURSOR` einen Speicherbereich für die Verarbeitung der SQL-Anweisung fest.
2. Prüfen Sie mit `PARSE` die Gültigkeit der SQL-Anweisung.
3. Führen Sie die SQL-Anweisung mit der Funktion `EXECUTE` aus. Diese Funktion gibt die Anzahl der verarbeiteten Zeilen zurück.
4. Schließen Sie den Cursor mit `CLOSE_CURSOR`.

DDL-Anweisungen werden auf ähnliche Weise ausgeführt. Der 3. Schritt ist dabei allerdings optional, da DDL-Anweisungen bei erfolgreichem PARSE, d. h. wenn Syntax und Semantik der Anweisung korrekt sind, sofort ausgeführt werden. Die Verwendung von EXECUTE mit DDL-Anweisungen hat keinerlei Auswirkung. Sie gibt für die Anzahl der verarbeiteten Zeilen den Wert 0 zurück, da DDL-Anweisungen keine Zeilen verarbeiten.

```
table TEMP_EMP created.  
anonymous block completed  
Rows Deleted: 107  
  
table TEMP_EMP dropped.
```

DBMS_SQL mit parametrisierten DML-Anweisungen

```

CREATE PROCEDURE insert_row (p_table_name VARCHAR2,
                            p_id VARCHAR2, p_name VARCHAR2, p_region NUMBER) IS
    v_cur_id      INTEGER;
    v_stmt        VARCHAR2(200);
    v_rows_added  NUMBER;
BEGIN
    v_stmt := 'INSERT INTO '|| p_table_name |||
              ' VALUES (:cid, :cname, :rid)';
    v_cur_id := DBMS_SQL.OPEN_CURSOR;
    DBMS_SQLPARSE(v_cur_id, v_stmt, DBMS_SQL.NATIVE);
    DBMS_SQL.BIND_VARIABLE(v_cur_id, ':cid', p_id);
    DBMS_SQL.BIND_VARIABLE(v_cur_id, ':cname', p_name);
    DBMS_SQL.BIND_VARIABLE(v_cur_id, ':rid', p_region);
    v_rows_added := DBMS_SQL.EXECUTE(v_cur_id);
    DBMS_SQL.CLOSE_CURSOR(v_cur_id);
    DBMS_OUTPUT.PUT_LINE(v_rows_added||' row added');
END;
/

```

ORACLE

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

Im Beispiel auf der Folie wird der DML-Vorgang zum Einfügen einer Zeile in eine angegebene Tabelle ausgeführt. Das Beispiel zeigt den zusätzlichen Schritt, der für die Zuordnung von Werten zu Bind-Variablen erforderlich ist, die in der SQL-Anweisung enthalten sind. So rufen Sie beispielsweise die auf der Folie gezeigte Prozedur auf:

```
EXECUTE insert_row('countries', 'LB', 'Lebanon', 4)
```

Nach dem Parsen der Anweisung müssen Sie die Prozedur `DBMS_SQL.BIND_VARIABLE` aufrufen, um den einzelnen Bind-Variablen in der Anweisung Werte zuzuweisen. Das Binden der Werte muss vor der Ausführung des Codes erfolgen. Um eine `SELECT`-Anweisung dynamisch zu verarbeiten, gehen Sie zwischen dem Öffnen und Schließen des Cursors wie folgt vor:

1. Führen Sie für jede gewählte Spalte `DBMS_SQL.DEFINE_COLUMN` aus.
2. Führen Sie für jede Bind-Variable in der Abfrage `DBMS_SQL.BIND_VARIABLE` aus.
3. Gehen Sie in jeder Zeile folgendermaßen vor:
 - a. Führen Sie `DBMS_SQL.FETCH_ROWS` aus, um eine Zeile abzurufen und die Anzahl der abgerufenen Zeilen zurückzugeben. Stoppen Sie die weitere Verarbeitung, wenn der Wert Null zurückgegeben wird.
 - b. Führen Sie `DBMS_SQL.COLUMN_VALUE` aus, um die einzelnen gewählten Spaltenwerte zur Verarbeitung in jede PL/SQL-Variable abzurufen.

Dieser Codierungsprozess ist zwar nicht komplex, doch dauert seine Erstellung länger und ist fehleranfälliger als die Verwendung von nativem dynamischem SQL.

Quiz

Der vollständige Text der dynamischen SQL-Anweisung ist möglicherweise erst zur Laufzeit bekannt. Die Anweisungsyntax wird daher nicht bei der *Kompilierung*, sondern erst während der *Laufzeit* geprüft.

- a. Richtig
- b. Falsch

ORACLE

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

Richtige Antwort: a

Zusammenfassung

In dieser Lektion haben Sie Folgendes gelernt:

- Ausführungsablauf von SQL-Anweisungen beschreiben
- SQL-Anweisungen dynamisch mit nativem dynamischem SQL (NDS) erstellen und ausführen
- Situationen erkennen, in denen anstelle von NDS das Package `DBMS_SQL` verwendet werden muss, um SQL-Anweisungen dynamisch zu erstellen und auszuführen



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

In dieser Lektion haben Sie gelernt, SQL-Anweisungen dynamisch zu erstellen und mit nativen dynamischen SQL-Anweisungen auszuführen. Die dynamische Ausführung von SQL- und PL/SQL-Code erweitert die Funktionsmöglichkeiten von PL/SQL über Abfrage- und Transaktionsvorgänge hinaus. In früheren Releases der Datenbank konnten Sie mit dem Package `DBMS_SQL` ähnliche Ergebnisse erzielen.

Übungen zu Lektion 7 – Überblick: Natives dynamisches SQL

Diese Übungen behandeln folgende Themen:

- **Package erstellen, das mithilfe von nativem dynamischem SQL eine Tabelle erstellt oder löscht und Zeilen in einer Tabelle füllt, ändert und löscht**
- **Package erstellen, das den PL/SQL-Code in Ihrem Schema kompiliert**

ORACLE

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

Übungen zu Lektion 7 – Überblick

In diesen Übungen erstellen Sie Code, um folgende Aufgaben auszuführen:

- Package erstellen, das mithilfe von nativem dynamischem SQL eine Tabelle erstellt oder löscht und Zeilen in der Tabelle füllt, ändert und löscht
- Package erstellen, das den PL/SQL-Code in Ihrem Schema kompiliert – entweder den gesamten PL/SQL-Code oder nur Code, der in der Tabelle USER_OBJECTS den Status INVALID hat.

Überlegungen zum Design von PL/SQL-Code

8

ORACLE

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

Ziele

Nach Ablauf dieser Lektion haben Sie folgende Ziele erreicht:

- **Standardkonstanten und Exceptions erstellen**
- **Lokale Unterprogramme erstellen und aufrufen**
- **Laufzeitberechtigungen von Unterprogrammen steuern**
- **Autonome Transaktionen ausführen**
- **PL/SQL-Packages und Standalone Stored Subprograms Rollen erteilen**
- **Parameter mit dem Hint NOCOPY per Referenz übergeben**
- **Mit dem Hint PARALLEL ENABLE eine Optimierung vornehmen**
- **Sessionübergreifenden Ergebniscache für PL/SQL-Funktionen verwenden**
- **Klausel DETERMINISTIC mit Funktionen verwenden**
- **Bulk Binding und die Klausel RETURNING mit DML verwenden**



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

In dieser Lektion lernen Sie, Namen von konstanten Werten und Exceptions mithilfe von Packagespezifikationen zu standardisieren. Sie erfahren, wie im Bereich Declaration von PL/SQL-Blöcken Unterprogramme zur lokalen Verwendung in einem Block erstellt werden. Die Beschreibung der Compileranweisung AUTHID soll Ihnen zeigen, wie Sie Laufzeitberechtigungen des PL/SQL-Codes verwalten und mit der Anweisung AUTONOMOUS TRANSACTION unabhängige Transaktionen für Unterprogramme erstellen.

Außerdem werden in dieser Lektion performancebezogene Überlegungen behandelt, die in PL/SQL-Anwendungen verwendet werden können, beispielsweise Bulk Binding-Vorgänge mit einer einzigen SQL-Anweisung, die Klausel RETURNING und die Hints NOCOPY und PARALLEL ENABLE.

Lektionsagenda

- Konstanten und Exceptions standardisieren, lokale Unterprogramme verwenden, Laufzeitberechtigungen von Unterprogrammen steuern und autonome Transaktionen ausführen
- PL/SQL-Packages und Standalone Stored Subprograms Rollen erteilen
- Hints NOCOPY und PARALLEL ENABLE, sessionübergreifenden Ergebniscache für PL/SQL-Funktionen und Klausel DETERMINISTIC verwenden
- Zwischenspeicherung der Funktion für die Rechte des ausführenden Benutzers verwenden
- Bulk Binding und die Klausel RETURNING mit DML verwenden

ORACLE

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

Konstanten und Exceptions standardisieren

Konstanten und Exceptions werden normalerweise mit einem Package ohne Body implementiert (d. h. in einer Packagespezifikation).

- **Vorteile der Standardisierung:**
 - Konsistente Programme entwickeln
 - Code häufiger wiederverwenden
 - Codeverwaltung vereinfachen
 - Unternehmensstandards anwendungsübergreifend implementieren
- **Standardisierung beginnen mit:**
 - Exception-Namen
 - Konstantendefinitionen



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

Wenn mehrere Entwickler eigene Exception Handler in einer Anwendung schreiben, kann dies zu einer inkonsistenten Behandlung von Fehlersituationen führen. Werden bestimmte Standards nicht eingehalten, können verwirrende Situationen entstehen, weil entweder derselbe Fehler mit unterschiedlichen Verfahren behandelt wird oder widersprüchliche, den Benutzer verunsichernde Fehlermeldungen angezeigt werden. Um dies zu vermeiden, können Sie:

- Unternehmensstandards implementieren, die in der gesamten Anwendung ein konsistentes Verfahren zur Fehlerbehandlung gewährleisten
- vordefinierte, allgemeingültige Exception Handler erstellen, die für Konsistenz in der Anwendung sorgen
- Programme schreiben und aufrufen, die konsistente Fehlermeldungen anzeigen

Alle hochwertigen Programmierumgebungen verwenden Benennungs- und Codierungsstandards. Ein geeigneter Ausgangspunkt für Benennungs- und Codierungsstandards in PL/SQL sind allgemein verwendete Konstanten und Exceptions, die in der Anwendungsdomäne vorkommen.

Das PL/SQL-Konstrukt einer Packagespezifikation eignet sich ausgezeichnet für die Unterstützung der Standardisierung, da alle in der Packagespezifikation deklarierten Bezeichner öffentlich sind. Sie sind für die Unterprogramme, die vom Eigentümer des Packages entwickelt werden, sowie für den gesamten Code mit EXECUTE-Rechten für die Packagespezifikation sichtbar.

Exceptions standardisieren

Standardisiertes Fehlerbehandlungspackage erstellen, das alle benannten und vom Programmierer definierten Exceptions enthält, die in der Anwendung verwendet werden sollen

```
CREATE OR REPLACE PACKAGE error_pkg IS
    e_fk_err      EXCEPTION;
    e_seq_nbr_err EXCEPTION;
    PRAGMA EXCEPTION_INIT (e_fk_err, -2292);
    PRAGMA EXCEPTION_INIT (e_seq_nbr_err, -2277);
    ...
END error_pkg;
/
```

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

Das Package `error_pkg` im Beispiel auf der Folie ist ein standardisiertes Exception-Package. Es deklariert eine Gruppe vom Programmierer definierter Exception-Bezeichner. Da viele der in der Oracle-Datenbank vordefinierten Exceptions keinen identifizierenden Namen haben, ordnet das auf der Folie gezeigte Beispieldatei mithilfe der Anweisung `PRAGMA EXCEPTION_INIT` gewählte Exception-Namen einer Fehlernummer der Oracle-Datenbank zu. So können Sie in Ihren Anwendungen mit einem Standardverfahren auf beliebige Exceptions verweisen. Beispiel:

```
BEGIN
    DELETE FROM departments
    WHERE department_id = deptno;
    ...
EXCEPTION
    WHEN error_pkg.e_fk_err THEN
        ...
    WHEN OTHERS THEN
        ...
END;
/
```

Exception-Behandlung standardisieren

Gründe zum Erstellen von Unterprogrammen zur allgemeinen Exception-Behandlung:

- Fehler auf Basis der Exception-Werte **SQLCODE** und **SQLERRM** anzeigen
- Laufzeitfehler bequem mit Parametern im Code protokollieren, um Folgendes zu ermitteln:
 - Die Prozedur, in der der Fehler auftrat
 - Die Position (Zeilennummer) des Fehlers
 - **RAISE_APPLICATION_ERROR** mit Stacktrace-Funktionen, wobei das dritte Argument auf **TRUE** eingestellt ist

ORACLE

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

Eine standardisierte Exception-Behandlung kann als Standalone-Unterprogramm implementiert werden oder als Unterprogramm dem Package hinzugefügt werden, das die Standard-Exceptions definiert. Erstellen Sie ein Package mit folgendem Inhalt:

- Alle benannten Exceptions, die in der Anwendung zu verwenden sind
- Alle unbenannten, vom Programmierer definierten Exceptions, die in der Anwendung zu verwenden sind. Dies sind die Fehlernummern -20000 bis -20999.
- Ein Programm zum Aufruf von **RAISE_APPLICATION_ERROR** auf der Basis von Package-Exceptions
- Ein Programm zum Anzeigen von Fehlern auf der Basis der Werte für **SQLCODE** und **SQLERRM**
- Zusätzliche Objekte wie Fehlerlogtabellen und Programme für den Zugriff auf die Tabellen

Es ist eine gängige Vorgehensweise, Parameter zu verwenden, die den Namen der Prozedur und die Position angeben, an der der Fehler auftrat. Dies erleichtert Ihnen die Verfolgung von Laufzeitfehlern. Eine Alternative ist die Built In-Prozedur **RAISE_APPLICATION_ERROR**. Mit dieser Prozedur können Sie einen Stacktrace der Exceptions führen und damit die Folge der Aufrufe verfolgen, die zu dem Fehler führte. Stellen Sie dazu das dritte, optionale Argument auf **TRUE** ein. Beispiel:

```
RAISE_APPLICATION_ERROR(-20001, 'My first error', TRUE);
```

Dies ist sinnvoll, wenn auf diese Weise mehrere Exceptions ausgelöst werden.

Konstanten standardisieren

Bei Programmen, die lokale Variablen verwenden, deren Werte nicht geändert werden dürfen:

- **Variablen in Konstanten konvertieren, um den Aufwand für Verwaltung und Debugging zu verringern**
- **Zentrale Packagespezifikation mit allen Konstanten erstellen**

```
CREATE OR REPLACE PACKAGE constant_pkg IS
    c_order_received CONSTANT VARCHAR(2) := 'OR';
    c_order_shipped   CONSTANT VARCHAR(2) := 'OS';
    c_min_sal         CONSTANT NUMBER(3)  := 900;
END constant_pkg;
```

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

Definitionsgemäß ändern sich Werte von Variablen, während Werte von Konstanten nicht geändert werden dürfen. Wenn Sie Programme mit lokalen Variablen haben, deren Werte sich nicht ändern (dürfen), konvertieren Sie die Variablen in Konstanten. Dies erleichtert die Verwaltung und das Debugging des Codes.

Erstellen Sie ein einziges, gemeinsam genutztes Package, das alle Konstanten enthält. Dies vereinfacht die Verwaltung und Änderung der Konstanten erheblich. Sie können die Prozedur oder das Package zur Performancesteigerung beim Systemstart laden.

Das Beispiel auf der Folie zeigt das Package `constant_pkg` mit einigen wenigen Konstanten. Verweisen Sie in der Anwendung auf eine beliebige Packagekonstante. Beispiel:

```
BEGIN
    UPDATE employees
        SET salary = salary + 200
        WHERE salary <= constant_pkg.c_min_sal;
END;
/
```

Lokale Unterprogramme

Ein lokales Unterprogramm ist eine am Ende des deklarativen Bereichs definierte Prozedur oder Funktion in einem Unterprogramm.

```
CREATE PROCEDURE employee_sal(p_id NUMBER) IS
    v_emp employees%ROWTYPE;
    FUNCTION tax(p_salary VARCHAR2) RETURN NUMBER IS
    BEGIN
        RETURN p_salary * 0.825;
    END tax;
BEGIN
    SELECT * INTO v_emp
    FROM EMPLOYEES WHERE employee_id = p_id;
    DBMS_OUTPUT.PUT_LINE('Tax: ' || tax(v_emp.salary));
END;
/
EXECUTE employee_sal(100)
```

```
PROCEDURE EMPLOYEE_SAL compiled
anonymous block completed
Tax: 19800
```

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

Lokale Unterprogramme können ein Top-Down-Design fördern. Sie entfernen überflüssigen Code und verringern dadurch die Modulgröße. Dies ist einer der wichtigsten Gründe für das Erstellen von lokalen Unterprogrammen. Wenn ein Modul dieselbe Routine mehrmals benötigt, diese aber von keinem anderen Modul benötigt wird, definieren Sie die Routine als lokales Unterprogramm.

Sie können benannte PL/SQL-Blöcke im deklarativen Bereich beliebiger PL/SQL-Programme, Prozeduren, Funktionen oder anonymer Blöcke definieren, *sofern die Deklaration am Ende des Bereichs Declaration erfolgt*. Lokale Unterprogramme haben folgende Eigenschaften:

- Sie sind nur für den Block zugänglich, in dem sie definiert werden.
- Sie werden zusammen mit den sie einschließenden Blöcken kompiliert.

Die Vorteile von lokalen Unterprogrammen sind:

- Verringerung von sich wiederholendem Code
- Bessere Lesbarkeit und einfache Wartung des Codes
- Weniger Verwaltungsaufwand, da statt zwei Programmen nur ein Programm zu verwalten ist

Das Konzept ist einfach. Auf der Folie wird dies mit einem einfachen Beispiel zur Berechnung der Einkommensteuer aus dem Gehalt eines Mitarbeiters veranschaulicht.

Rechte des Eigentümers und Rechte des ausführenden Benutzers – Vergleich

Rechte des Eigentümers:

- Programme werden mit den Berechtigungen des erstellenden Benutzers ausgeführt.
- Benutzer benötigen keine Berechtigungen für zugrunde liegende Objekte, auf die die Prozedur zugreift. Benutzer benötigen lediglich eine Berechtigung zum Ausführen einer Prozedur.

Rechte des ausführenden Benutzers:

- Programme werden mit den Berechtigungen des aufrufenden Benutzers ausgeführt.
- Benutzer benötigen Berechtigungen für zugrunde liegende Objekte, auf die die Prozedur zugreift.

ORACLE

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

Rechtemodell des Eigentümers

Alle Programme werden standardmäßig mit den Berechtigungen des Benutzers ausgeführt, der das Unterprogramm erstellt hat. Dieses so genannte "Rechtemodell des Eigentümers" hat folgende Eigenschaften:

- Der aufrufende Benutzer hat die Berechtigung, die Prozedur auszuführen, jedoch keine Berechtigungen für die zugrunde liegenden Objekte, auf die die Prozedur zugreift.
- Der Eigentümer benötigt alle erforderlichen Objektberechtigungen für die Objekte, die die Prozedur referenziert.

Wenn beispielsweise der Benutzer Scott ein PL/SQL-Unterprogramm namens `get_employees` erstellt, das anschließend von Sarah aufgerufen wird, erfolgt die Ausführung der Prozedur `get_employees` mit den Berechtigungen für den Eigentümer Scott.

Rechtemodell des ausführenden Benutzers

Beim Rechtemodell des ausführenden Benutzers, das in Oracle8i eingeführt wurde, werden die Programme mit den Berechtigungen des aufrufenden Benutzers ausgeführt. Benutzer einer Prozedur, die mit Rechten des ausführenden Benutzers ausgeführt wird, benötigen Berechtigungen für die zugrunde liegenden Objekte, die die Prozedur referenziert.

Wird beispielsweise Scotts PL/SQL-Unterprogramm `get_employees` von Sarah aufgerufen, erfolgt die Ausführung der Prozedur `get_employees` mit den Berechtigungen von Sarah, der ausführenden Benutzerin.

Rechte des ausführenden Benutzers angeben – AUTHID auf CURRENT_USER einstellen

```
CREATE OR REPLACE PROCEDURE add_dept(
    p_id NUMBER, p_name VARCHAR2) AUTHID CURRENT_USER IS
BEGIN
    INSERT INTO departments
    VALUES (p_id, p_name, NULL, NULL);
END;
```

Bei Verwendung mit Standalone-Funktionen, -Prozeduren oder Packages:

- **Namen in Abfragen, DML, nativem dynamischem SQL und dem Package DBMS_SQL werden im Schema des ausführenden Benutzers aufgelöst.**
- **Aufrufe anderer Packages, Funktionen und Prozeduren werden im Schema des Eigentümers aufgelöst.**



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

Rechte des ausführenden Benutzers angeben

Sie können die Rechte des ausführenden Benutzers für unterschiedliche PL/SQL-Unterprogrammkonstrukte wie folgt einstellen:

```
CREATE FUNCTION name RETURN type AUTHID CURRENT_USER IS...
CREATE PROCEDURE name AUTHID CURRENT_USER IS...
CREATE PACKAGE name AUTHID CURRENT_USER IS...
CREATE TYPE name AUTHID CURRENT_USER IS OBJECT...
```

Der Standardwert AUTHID DEFINER gibt an, dass das Unterprogramm mit den Berechtigungen des jeweiligen Eigentümers ausgeführt wird. Die meisten bereitgestellten PL/SQL-Packages (DBMS_LOB, DBMS_ROWID usw.) verwenden das Rechtemodell des ausführenden Benutzers.

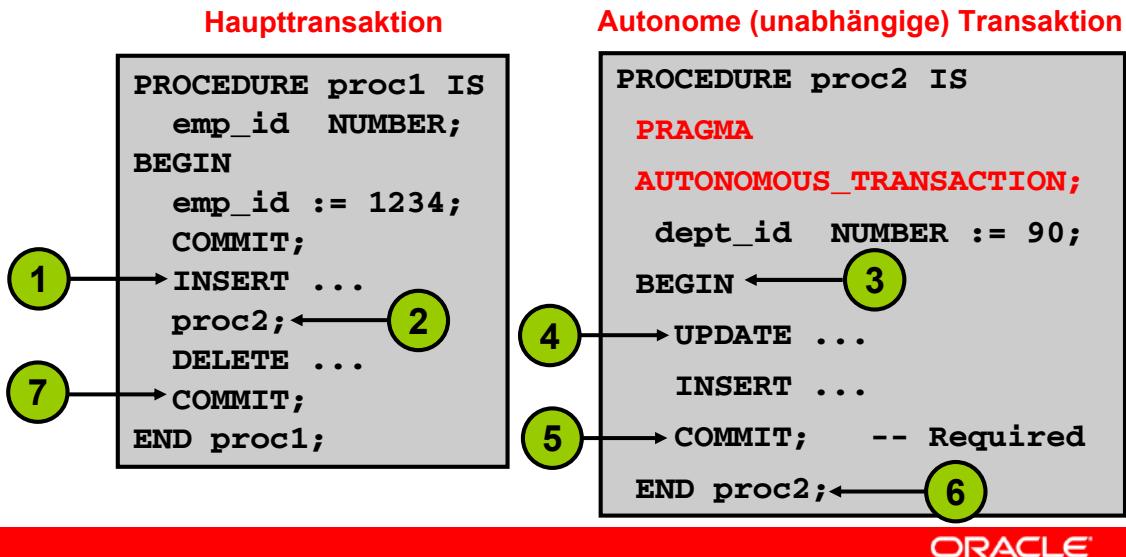
Namensauflösung

Bei Prozeduren mit Rechten des Eigentümers werden alle externen Referenzen im Schema des Eigentümers aufgelöst. Bei Prozeduren mit Rechten des ausführenden Benutzers richtet sich die Auflösung externer Referenzen nach der Art der Anweisung, in der sie sich befinden:

- Namen in Abfragen, DML-Anweisungen, dynamischem SQL und DBMS_SQL werden im Schema des ausführenden Benutzers aufgelöst.
- Alle übrigen Anweisungen wie Aufrufe anderer Packages, Funktionen und Prozeduren werden im Schema des Eigentümers aufgelöst.

Autonome Transaktionen

- Sind unabhängige Transaktionen, die von einer anderen Haupttransaktion gestartet werden
- Werden mit PRAGMA AUTONOMOUS_TRANSACTION angegeben



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

Eine Transaktion umfasst eine Reihe von Anweisungen zur Ausführung einer logischen Arbeitseinheit, die als Ganzes gelingt oder fehlschlägt. Oft startet eine Transaktion eine weitere, die möglicherweise außerhalb des Bereichs der ersten Transaktion operieren muss. Mit anderen Worten: In einer vorhandenen Transaktion ist möglicherweise eine unabhängige Transaktion erforderlich, um Änderungen festzuschreiben oder zurückzurollen, ohne das Ergebnis der startenden Transaktion zu beeinflussen. Beispiel: Bei einer Aktienkauftransaktion müssen die Kundeninformationen unabhängig davon festgeschrieben werden, ob der Aktienkauf als solcher durchgeführt wird. Oder Sie möchten während derselben Transaktion Meldungen in einer Tabelle protokollieren, auch wenn die gesamte Transaktion zurückgerollt wird.

Seit Oracle8i sind autonome Transaktionen zur Erstellung unabhängiger Transaktionen verfügbar. Eine autonome Transaktion ist eine unabhängige Transaktion, die von einer anderen Haupttransaktion gestartet wird. Auf der Folie wird das Verhalten einer autonomen Transaktion dargestellt:

- Die Haupttransaktion beginnt.
- Eine proc2-Prozedur wird aufgerufen, um die autonome Transaktion zu starten.
- Die Haupttransaktion wird unterbrochen.
- Die Aktion der autonomen Transaktion beginnt.
- Die autonome Transaktion endet mit Festschreiben oder Zurückrollen.
- Die Haupttransaktion wird fortgesetzt.
- Die Haupttransaktion endet.

Autonome Transaktionen – Features

- **Sind unabhängig von der Haupttransaktion**
- **Unterbrechen die aufrufende Transaktion, bis die autonomen Transaktionen abgeschlossen sind**
- **Sind keine verschachtelten Transaktionen**
- **Rollen beim Zurückrollen der Haupttransaktion nicht zurück**
- **Machen die Änderungen durch Festschreiben für andere Transaktionen sichtbar**
- **Werden mit separaten Unterprogrammen gestartet und beendet und nicht mit verschachtelten oder anonymen PL/SQL-Blöcken**

ORACLE

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

Autonome Transaktionen weisen folgende Features auf:

- Autonome Transaktionen werden zwar innerhalb einer Transaktion aufgerufen, sind von dieser jedoch unabhängig. Das heißt, es handelt sich nicht um verschachtelte Transaktionen.
- Beim Zurückrollen der Haupttransaktion rollen autonome Transaktionen nicht zurück.
- Änderungen, die eine autonome Transaktion vornimmt, werden durch Festschreiben für andere Transaktionen sichtbar.
- Aufgrund der stackartigen Funktionalität ist stets nur die "oberste" Transaktion zugänglich. Nach dem Abschluss einer autonomen Transaktion wird diese vom Stack genommen, und die aufrufende Transaktion wird fortgesetzt.
- Außer den Ressourcengrenzen gibt es keinerlei Einschränkungen für die Anzahl der rekursiven Aufrufe autonomer Transaktionen.
- Autonome Transaktionen müssen explizit festgeschrieben oder zurückgerollt werden. Andernfalls wird beim Versuch, aus dem autonomen Block zurückzukehren, ein Fehler zurückgegeben.
- Sie können mit `PRAGMA` nicht alle Unterprogramme in einem Package als autonom markieren. Dies ist nur für einzelne Routinen möglich.
- Verschachtelte oder anonyme PL/SQL-Blöcke können nicht als autonom markiert werden.

Autonome Transaktionen – Beispiel

```

CREATE TABLE usage (card_id NUMBER, loc NUMBER)
/
CREATE TABLE txn (acc_id NUMBER, amount NUMBER)
/
CREATE OR REPLACE PROCEDURE log_usage (p_card_id NUMBER, p_loc NUMBER)
IS
    PRAGMA AUTONOMOUS_TRANSACTION;
BEGIN
    INSERT INTO usage
    VALUES (p_card_id, p_loc);
    COMMIT;
END log_usage;
/
CREATE OR REPLACE PROCEDURE bank_trans(p_cardnbr NUMBER,p_loc NUMBER) IS
BEGIN
    INSERT INTO txn VALUES (9001, 1000);
    log_usage (p_cardnbr, p_loc);
END bank_trans;
/
EXECUTE bank_trans(50, 2000)

```

ORACLE

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

Autonome Transaktionen

Sie definieren autonome Transaktionen mit `PRAGMA AUTONOMOUS_TRANSACTION`. `PRAGMA` weist den PL/SQL-Compiler an, eine Routine als autonom (unabhängig) zu markieren. In diesem Kontext umfasst der Begriff "Routine" folgende Elemente: anonyme PL/SQL-Blöcke der obersten Ebene (nicht verschachtelt); lokale, Standalone- und in Packages integrierte Funktionen und Prozeduren; Methoden eines SQL-Objekttyps; Datenbanktrigger. Sie können `PRAGMA` an einer beliebigen Stelle des deklarativen Bereichs einer Routine codieren. Die beste Lesbarkeit erzielen Sie jedoch am Anfang des Bereichs `Declaration`.

Im Beispiel auf der Folie protokollieren Sie die Einsatzorte der Bankkarte unabhängig vom Erfolg der einzelnen Transaktionen. Autonome Transaktionen haben bestimmte Vorteile:

- Nach dem Start sind autonome Transaktionen völlig unabhängig. Sie verwenden keine Sperren, Ressourcen oder Festschreibungsabhängigkeiten gemeinsam mit der Haupttransaktion. Dadurch können Sie auch beim Zurückrollen der Haupttransaktion Ereignisse protokollieren, Wiederholungszähler schrittweise erhöhen usw.
- Noch wichtiger ist die Tatsache, dass Sie mithilfe autonomer Transaktionen modulare, wiederverwendbare Softwarekomponenten erstellen können. Stored Procedures beispielsweise können autonome Transaktionen eigenständig starten und beenden. Eine aufrufende Anwendung muss nichts über die autonomen Vorgänge einer Prozedur wissen, und die Prozedur muss den Transaktionskontext der Anwendung nicht kennen. Dadurch sind autonome Transaktionen weniger fehleranfällig und benutzerfreundlicher als normale Transaktionen.

Das Beispiel auf der vorherigen Folie generiert die folgende Ausgabe für die Tabellen TXN und USAGE:

```
table USAGE created.  
table TXN created.  
PROCEDURE LOG_USAGE compiled  
PROCEDURE BANK_TRANS compiled  
anonymous block completed
```

Um den Code auf der vorherigen Seite auszuführen, geben Sie folgenden Code ein:

```
EXECUTE bank_trans(50, 2000)
```

Zeigen Sie im Object Navigator-Baum in der Registerkarte **Data** des Knotens **Tables** die Werte in den Tabellen TXN und USAGE wie folgt an:

The screenshot shows the Oracle Database Object Navigator interface with two tables displayed side-by-side.

TXN Table:

ACC_ID	AMOUNT
1	9001
	1000

USAGE Table:

CARD_ID	LOC
1	50
	2000

Lektionsagenda

- Konstanten und Exceptions standardisieren, lokale Unterprogramme verwenden, Laufzeitberechtigungen von Unterprogrammen steuern und autonome Transaktionen ausführen
- **PL/SQL-Packages und Standalone Stored Subprograms Rollen erteilen**
- Hints **NOCOPY** und **PARALLEL ENABLE**, sessionübergreifenden Ergebniscache für PL/SQL-Funktionen und Klausel **DETERMINISTIC** verwenden
- Bulk Binding und die Klausel **RETURNING** mit DML verwenden

A solid red horizontal bar spanning most of the slide width.

ORACLE

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

PL/SQL-Packages und Standalone Stored Subprograms Rollen erteilen

- **Der SQL-Befehl GRANT erteilt PL/SQL-Packages und Standalone-Unterprogrammen Rollen:**
 - Einheit mit Rechten des ausführenden Benutzers erstellen
 - Einheit mit Rechten des ausführenden Benutzers Rollen erteilen
- **Die Einheit mit Rechten des ausführenden Benutzers wird mit den Berechtigungen des ausführenden Benutzers und den Berechtigungen der Rollen ausgeführt.**



ORACLE

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

Vor Oracle Database 12c wurde eine Einheit mit Rechten des Eigentümers immer mit den Berechtigungen des Eigentümers und eine Einheit mit Rechten des ausführenden Benutzers immer mit den Berechtigungen des ausführenden Benutzers ausgeführt. Um damals eine PL/SQL-Einheit zu erstellen, die alle Benutzer aufrufen konnten, selbst wenn deren Berechtigungen niedriger angesiedelt waren als Ihre eigenen, mussten Sie eine Einheit mit Rechten des Eigentümers verwenden. Diese Einheit wurde immer mit all Ihren Berechtigungen ausgeführt, ungeachtet des Benutzers, der sie aufgerufen hat.

Seit Oracle Database 12c können Sie einzelnen PL/SQL-Packages und Standalone-Unterprogrammen Rollen erteilen. Anstelle einer Einheit mit Rechten des Eigentümers können Sie eine Einheit mit Rechten des ausführenden Benutzers erstellen und dieser dann Rollen erteilen. Die Einheit mit Rechten des ausführenden Benutzers wird mit den Berechtigungen des ausführenden Benutzers und den Berechtigungen der Rollen ausgeführt, jedoch nicht mit eventuellen weiteren Berechtigungen, die Sie besitzen.

Sie erteilen einer Einheit mit Rechten des ausführenden Benutzers Rollen, sodass Benutzer, die über weniger Berechtigungen verfügen als Sie, diese Einheit nur mit den dafür erforderlichen Berechtigungen ausführen können. Es gibt keinen Grund, einer Einheit mit Rechten des Eigentümers Rollen zu erteilen, da deren ausführende Benutzer die Einheit mit all Ihren Berechtigungen ausführen.

Mit dem SQL-Befehl GRANT können Sie PL/SQL-Packages und Standalone Stored Subprograms Rollen erteilen. Die einer PL/SQL-Einheit erteilten Rollen wirken sich nicht auf die Kompilierung aus. Sie beeinflussen die Berechtigungsprüfung von SQL-Anweisungen, die von der Einheit zur Laufzeit abgesetzt werden. Die Einheit wird mit den Berechtigungen ihrer eigenen Rollen und allen anderen derzeit aktivierten Rollen ausgeführt.

Lektionsagenda

- Konstanten und Exceptions standardisieren, lokale Unterprogramme verwenden, Laufzeitberechtigungen von Unterprogrammen steuern und autonome Transaktionen ausführen
- PL/SQL-Packages und Standalone Stored Subprograms Rollen erteilen
- Hints **NOCOPY** und **PARALLEL ENABLE**, sessionübergreifenden Ergebniscache für PL/SQL-Funktionen und Klausel **DETERMINISTIC** verwenden
- Bulk Binding und die Klausel **RETURNING** mit DML verwenden

ORACLE

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

Hint NOCOPY

- Ermöglicht dem PL/SQL-Compiler die Übergabe der Parameter OUT und IN OUT per Referenz anstatt mit einem Wert
- Erhöht die Performance durch geringeren Overhead beim Übergeben von Parametern

```

DECLARE
  TYPE      rec_emp_type IS TABLE OF employees%ROWTYPE;
  rec_emp  rec_emp_type;
  PROCEDURE populate(p_tab IN OUT NOCOPY emptabtype) IS
    BEGIN
      . . .
    END;
BEGIN
  populate(rec_emp);
END;
/

```

ORACLE

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

Beachten Sie, dass PL/SQL-Unterprogramme drei Modi der Parameterübergabe unterstützen: IN, OUT und IN OUT. Standardmäßig gilt:

- Der Parameter IN wird per Referenz übergeben. Ein Zeiger auf den tatsächlichen Parameter IN wird an den entsprechenden formalen Parameter übergeben. Somit referenzieren beide Parameter denselben Speicherort, der den Wert des tatsächlichen Parameters enthält.
- Die Parameter OUT und IN OUT werden mit einem Wert übergeben. Der Wert des tatsächlichen Parameters OUT oder IN OUT wird dabei in die entsprechenden formalen Parameter kopiert. Anschließend werden, sofern das Unterprogramm ordnungsgemäß beendet wird, die den formalen Parametern OUT und IN OUT zugewiesenen Werte in die entsprechenden tatsächlichen Parameter kopiert.

Das Kopieren von Parametern, die für große Datenstrukturen stehen (z. B. Collections, Records oder Objekttypinstanzen), mit den Parametern OUT und IN OUT führt zu einer verlangsamen Ausführung und einem erhöhten Speicherbedarf. Zur Vermeidung dieses Overheads können Sie den Hint NOCOPY angeben, der dem PL/SQL-Compiler die Übergabe derartiger Parameter per Referenz ermöglicht.

Das Beispiel auf der Folie zeigt die Deklaration eines IN OUT-Parameters mit dem Hint NOCOPY.

Auswirkungen des Hints NOCOPY

- **Wenn das Unterprogramm mit einer nicht behandelten Exception beendet wird:**
 - können Sie sich nicht auf die Werte der tatsächlichen Parameter verlassen, die an einen NOCOPY-Parameter übergeben werden
 - werden unvollständige Änderungen nicht "zurückgerollt"
- **Mit dem RPC-(Remote Procedure Call-)Protokoll können Sie Parameter nur mit einem Wert übergeben.**

ORACLE

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

Mit dem Hint NOCOPY erreichen Sie eine bessere Performance, die jedoch zu Lasten einer klar definierten Exception-Semantik geht. Der Hint wirkt sich wie folgt auf die Exception-Behandlung aus:

- Da NOCOPY keine Direktive, sondern ein Hint ist, kann der Compiler NOCOPY-Parameter mit einem Wert oder per Referenz an ein Unterprogramm übergeben. Wenn also das Unterprogramm mit einer nicht behandelten Exception beendet wird, können Sie sich nicht auf die Werte der tatsächlichen NOCOPY-Parameter verlassen.
- Bei Beendigung eines Unterprogramms mit einer nicht behandelten Exception werden die Werte, die den formalen Parametern OUT und IN OUT zugewiesen wurden, standardmäßig nicht in die entsprechenden tatsächlichen Parameter kopiert, und Änderungen scheinen zurückgerollt zu werden. Wenn Sie dagegen NOCOPY angeben, wirken sich die Zuordnungen zu den formalen Parametern auch sofort auf die tatsächlichen Parameter aus. Wenn also das Unterprogramm mit einer nicht behandelten Exception beendet wird, werden die (möglicherweise unvollständigen) Änderungen nicht "zurückgerollt".
- Mit dem RPC-Protokoll können Sie derzeit Parameter nur mit einem Wert übergeben. Somit kann sich die Exception-Semantik beim Partitionieren von Anwendungen ohne Benachrichtigung ändern. Wenn Sie beispielsweise eine lokale Prozedur mit NOCOPY-Parametern an einen Remote-Standort verschieben, werden diese Parameter nicht mehr per Referenz übergeben.

Wann ignoriert der PL/SQL-Compiler den Hint **NOCOPY**?

Der Hint **NOCOPY hat in folgenden Situationen keine Auswirkung:**

- **Der tatsächliche Parameter:**
 - ist ein Element von assoziativen Arrays (INDEX BY-Tabellen)
 - verfügt über ein Constraint (z. B. Anzahl der Nachkommastellen oder NOT NULL)
 - und der formale Parameter sind Records, wobei einer oder beide Records mit %ROWTYPE oder %TYPE deklariert wurden und unterschiedliche Constraints für die entsprechenden Felder in den Records haben
 - benötigt eine implizite Datentypkonvertierung
- **Das Unterprogramm ist am Aufruf einer externen oder Remote-Prozedur beteiligt.**



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

In den folgenden Fällen ignoriert der PL/SQL-Compiler den Hint **NOCOPY** und übergibt die Parameter mit einem Wert (ohne Fehlergenerierung):

- Der tatsächliche Parameter ist ein Element von assoziativen Arrays (INDEX BY-Tabellen). Diese Einschränkung gilt nicht für komplette assoziative Arrays.
- Der tatsächliche Parameter verfügt über ein Constraint (z. B. Anzahl der Nachkommastellen oder NOT NULL). Diese Einschränkung erstreckt sich nicht auf Elemente oder Attribute mit Constraints. Außerdem gilt sie nicht für Zeichenfolgen mit Größenbeschränkungen.
- Der tatsächliche Parameter und der formale Parameter sind Records, wobei einer oder beide Records mit %ROWTYPE oder %TYPE deklariert wurden und unterschiedliche Constraints für die entsprechenden Felder in den Records haben.
- Der tatsächliche Parameter und der formale Parameter sind Records, wobei der tatsächliche Parameter (implizit) als Index einer Cursor FOR-Schleife deklariert wurde und die beiden Parameter unterschiedliche Constraints für die entsprechenden Felder in den Records haben.
- Für die Übergabe des tatsächlichen Parameters ist eine implizite Datentypkonvertierung erforderlich.
- Das Unterprogramm ist am Aufruf einer externen oder Remote-Prozedur beteiligt.

Hint PARALLEL_ENABLE

- Kann in Funktionen als Optimierungs-Hint verwendet werden
- Zeigt an, dass eine Funktion in parallelisierten Abfragen oder DML-Anweisungen verwendet werden kann

```
CREATE OR REPLACE FUNCTION f2 (p_p1 NUMBER)
  RETURN NUMBER PARALLEL_ENABLE IS
BEGIN
  RETURN p_p1 * 2;
END f2;
```

FUNCTION F2 compiled

ORACLE

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

Mit dem Schlüsselwort PARALLEL_ENABLE können Sie in der Syntax eine Funktion deklarieren. Dieser Optimierungs-Hint zeigt an, dass die Funktion in parallelisierten Abfragen oder DML-Anweisungen verwendet werden kann. Das Oracle-Feature für parallele Ausführung verteilt die Ausführung einer SQL-Anweisung auf mehrere Prozesse. Bei Funktionen, die aus einer parallel ausgeführten SQL-Anweisung aufgerufen werden, kann in jedem dieser Prozesse eine separate Kopie ausgeführt werden. Dabei wird jede Kopie nur für die Zeilen aufgerufen, die von dem entsprechenden Prozess verarbeitet werden.

Vor Oracle8i suchte die Parallelisierungsoptimierung in DML-Anweisungen nach Funktionen, bei denen RNDS, WNDS, RNPS und WNPS gemeinsam in einer PRAGMA RESTRICT_REFERENCES-Deklaration angegeben waren. Diese Funktionen, die weder Lese- noch Schreibvorgänge für die Datenbank- oder Packagevariablen durchführten, konnten parallel ausgeführt werden. Bei Funktionen wiederum, die mit einer CREATE FUNCTION-Anweisung definiert wurden, erfolgte eine implizite Codeprüfung daraufhin, ob sie wirklich rein genug waren. Eine parallelisierte Ausführung war gegebenenfalls möglich, obwohl PRAGMA in diesen Funktionen nicht angegeben werden konnte.

Das Schlüsselwort PARALLEL_ENABLE steht in der Deklaration der Funktion hinter dem Rückgabewerttyp, wie im Beispiel auf der Folie gezeigt.

Hinweis: Die Funktion darf keinen Sessionzustand verwenden, beispielsweise Packagevariablen, da die für die parallele Ausführung zuständigen Server diese Variablen möglicherweise nicht gemeinsam verwenden können.

Sessionübergreifender Ergebniscache für PL/SQL-Funktionen

- Bei jedem Aufruf einer zwischengespeicherten PL/SQL-Funktion mit wechselnden Parameterwerten werden sowohl die Parameter als auch ihre Ergebnisse im Ergebniscache abgelegt.
- Der Ergebniscache für Funktionen ist in einer SGA (Shared Global Area) abgelegt und damit für jede Session verfügbar, in der die Anwendung ausgeführt wird.
- Bei künftigen Aufrufen dieser Funktion mit denselben Parametern werden die Ergebnisse aus dem Cache verwendet.
- Performance und Skalierbarkeit werden verbessert.
- Dieses Feature eignet sich für häufig aufgerufene Funktionen mit selten aktualisierten Informationen.

ORACLE

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

In Oracle Database 11g ist erstmals der sessionübergreifende Ergebniscache für PL/SQL-Funktionen verfügbar. Dieser Cachingmechanismus stellt eine sprachunterstützte und systemverwaltete Methode zur Speicherung der Ergebnisse von PL/SQL-Funktionen in einer SGA (Shared Global Area) bereit, die allen Sessions zugänglich ist, in denen die Anwendung ausgeführt wird. Dank dieser effizienten und benutzerfreundlichen Methode brauchen Sie künftig keine eigenen Caches und Cacheverwaltungs-Policies mehr zu entwerfen und zu entwickeln.

Bei jedem Aufruf einer zwischengespeicherten PL/SQL-Funktion mit wechselnden Parameterwerten werden sowohl die Parameter als auch ihre Ergebnisse im Cache abgelegt. Wird dieselbe Funktion nachfolgend mit denselben Parameterwerten erneut aufgerufen, wird das Ergebnis nicht neu berechnet, sondern aus dem Cache abgerufen. Wird ein Datenbankobjekt aktualisiert, das zur Berechnung von zwischengespeicherten Ergebnissen verwendet wurde, wird das Cacheergebnis ungültig und muss erneut berechnet werden.

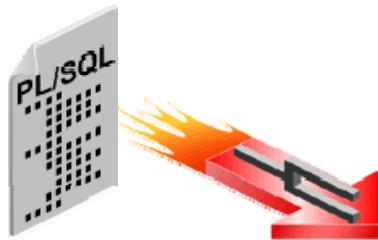
Der Ergebniscache eignet sich für Funktionen, die häufig aufgerufen werden und von Informationen abhängen, die sich nie oder nur selten ändern.

Hinweis: Weitere Informationen zum sessionübergreifenden Ergebniscache für PL/SQL-Funktionen finden Sie in den Kursen *Oracle Database Advanced PL/SQL* und *Oracle Database: SQL und PL/SQL New Features* oder im Dokument *Oracle Database PL/SQL Language Reference*.

Ergebniscache für Funktionen aktivieren

Zwischenspeicherung für Funktionen aktivieren:

- Klausel **RESULT_CACHE** aufnehmen in:
 - Funktionsdeklaration
 - Funktionsdefinition
- Optional eine **RELIES_ON**-Klausel aufnehmen, um alle Tabellen oder Views anzugeben, von denen die Funktionsergebnisse abhängen



ORACLE

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

Die Zwischenspeicherung für eine PL/SQL-Funktion aktivieren Sie mit der Klausel **RESULT_CACHE**. Bei jedem Aufruf einer zwischengespeicherten Funktion prüft das System den Ergebniscache. Ist dort bereits das Ergebnis aus einem früheren Aufruf der Funktion mit denselben Parameterwerten enthalten, gibt das System das zwischengespeicherte Ergebnis an den Aufrufer zurück, ohne den Funktionsbody erneut auszuführen. Andernfalls führt das System den Funktionsbody aus und nimmt das Ergebnis (für diese Parameterwerte) in den Cache auf, bevor die Kontrolle an den Aufrufer zurückgegeben wird.

Im Cache kann sich eine große Anzahl von Ergebnissen für die einzelnen Kombinationen aus Parameterwerten ansammeln, mit denen die betreffende Funktion bislang aufgerufen wurde. Benötigt das System mehr Speicher, werden zwischengespeicherte Ergebnisse in benötigter Anzahl als veraltet gelöscht.

Hinweis: Wenn die Ausführung der Funktion zu einer nicht behandelten Exception führt, wird das Ergebnis der Exception nicht im Cache gespeichert.

Zwischengespeicherte Funktionen deklarieren und definieren – Beispiel

- Zwischenspeicherung für eine Funktion mit der Klausel **RESULT_CACHE** aktivieren
- Beispiel:

```
CREATE OR REPLACE FUNCTION get_hire_date (emp_id NUMBER)
  RETURN VARCHAR
  RESULT_CACHE
  AUTHID CURRENT_USER
IS
  date_hired DATE;
BEGIN
  SELECT hire_date INTO date_hired
  from employees
  WHERE employee_id = emp_id;
  RETURN TO_CHAR(date_hired);
END;
/
SELECT get_hire_date(206) from DUAL;
```

ORACLE

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

Vor Oracle Database 12c konnte das Ergebnis der Funktion für die Rechte des ausführenden Benutzers nicht zwischengespeichert werden. Seit 12c ist dies möglich.

Die Zwischenspeicherung für eine Funktion aktivieren Sie mit der Klausel **RESULT_CACHE**. Bei jedem Aufruf einer zwischengespeicherten Funktion prüft das System den Cache. Ist dort bereits das Ergebnis aus einem früheren Aufruf der Funktion mit denselben Parameterwerten enthalten, gibt das System das zwischengespeicherte Ergebnis an den ausführenden Benutzer zurück, ohne den Funktionsbody erneut auszuführen. Andernfalls führt das System den Funktionsbody aus und nimmt das Ergebnis (für diese Parameterwerte) in den Cache auf, bevor die Kontrolle an den ausführenden Benutzer zurückgegeben wird.

Im Codebeispiel auf der Folie wird die Funktion `get_hire_date` erstellt, die `employee_id` als Parameter annimmt und `hire_date` zurückgibt. Im Beispiel führt die Funktion folgende Aufgaben aus:

- Stellt die Eigenschaft `AUTHID` auf die Rechte des ausführenden Benutzers ein
- Konvertiert mithilfe der Funktion `TO_CHAR` ein `DATE`-Element in ein `VARCHAR`-Element

Da die Funktion hier keine Formatmaske angibt, wird standardmäßig die unter `NLS_DATE_FORMAT` angegebene Formatmaske verwendet.

Fügen Sie `HireDate` wie folgt einen Formatmaskenparameter hinzu:

```
CREATE OR REPLACE FUNCTION get_hire_date (emp_id NUMBER, fmt VARCHAR)
  RETURN VARCHAR
RESULT_CACHE
AUTHID CURRENT_USER
IS
  date_hired DATE;
BEGIN
  SELECT hire_date INTO date_hired
  from employees
  WHERE employee_id = emp_id;
  RETURN TO_CHAR(date_hired, fmt);
END;
```

Klausel DETERMINISTIC mit Funktionen

- **Mithilfe von DETERMINISTIC geben Sie an, dass die Funktion bei jedem Aufruf mit denselben Argumentwerten den gleichen Ergebniswert zurückgibt.**
- **Damit kann der Optimizer überflüssige Funktionsaufrufe vermeiden.**
- **Wurde eine Funktion bereits mit denselben Argumenten aufgerufen, kann der Optimizer das entsprechende Ergebnis verwenden.**
- **Geben Sie DETERMINISTIC nicht für Funktionen an, deren Ergebnis vom Status der Sessionvariablen oder Schemaobjekte abhängt.**

ORACLE

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

Mithilfe der Funktionsklausel DETERMINISTIC können Sie angeben, dass die Funktion bei jedem Aufruf mit denselben Argumentwerten den gleichen Ergebniswert zurückgibt.

Sie müssen dieses Schlüsselwort angeben, wenn Sie beabsichtigen, die Funktion im Ausdruck eines funktionsbasierten Index oder aus der Abfrage einer als REFRESH FAST oder ENABLE QUERY REWRITE markierten Materialized View aufzurufen. Trifft die Oracle-Datenbank in einem solchen Kontext auf eine DETERMINISTIC-Funktion, versucht sie, soweit möglich die bereits berechneten Ergebnisse zu verwenden, anstatt die Funktion erneut auszuführen. Ändern Sie zu einem späteren Zeitpunkt die Semantik der Funktion, müssen Sie alle abhängigen funktionsbasierten Indizes und Materialized Views manuell neu erstellen.

Geben Sie diese Klausel nicht bei der Definition einer Funktion an, die Packagevariablen verwendet oder in einer Weise auf die Datenbank zugreift, die möglicherweise Auswirkungen auf das Rückgabergebnis der Funktion haben kann. Andernfalls werden keine Ergebnisse erfasst, wenn die Oracle-Datenbank die Funktion nicht erneut ausführt.

Hinweis

- Geben Sie DETERMINISTIC nicht für Funktionen an, deren Ergebnis vom Status der Sessionvariablen oder Schemaobjekte abhängt, da die Ergebnisse in diesem Fall von Aufruf zu Aufruf variieren können. Erwägen Sie stattdessen, den Ergebniscache für die Funktion zu aktivieren.
- Weitere Informationen über die Klausel DETERMINISTIC finden Sie im Dokument *Oracle Database SQL Language Reference*.

Lektionsagenda

- Konstanten und Exceptions standardisieren, lokale Unterprogramme verwenden, Laufzeitberechtigungen von Unterprogrammen steuern und autonome Transaktionen ausführen
- Hints **NOCOPY** und **PARALLEL ENABLE**, sessionübergreifenden Ergebniscache für PL/SQL-Funktionen und Klausel **DETERMINISTIC** verwenden
- **Bulk Binding** und die Klausel **RETURNING** mit DML verwenden

ORACLE

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

Klausel RETURNING

- Verbessert die Performance durch Rückgabe von Spaltenwerten für die Anweisungen **INSERT**, **UPDATE** und **DELETE**
- Macht eine **SELECT**-Anweisung überflüssig

```

CREATE OR REPLACE PROCEDURE update_salary(p_emp_id
    NUMBER) IS
    v_name      employees.last_name%TYPE;
    v_new_sal   employees.salary%TYPE;
BEGIN
    UPDATE employees
        SET salary = salary * 1.1
        WHERE employee_id = p_emp_id
    RETURNING last_name, salary INTO v_name, v_new_sal;
    DBMS_OUTPUT.PUT_LINE(v_name || ' new salary is ' ||
        v_new_sal);
END update_salary;
/

```

ORACLE

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

Anwendungen benötigen häufig Angaben zu der von einem SQL-Vorgang betroffenen Zeile, z. B. um Berichte zu generieren oder nachfolgende Aktionen durchzuführen. Die Anweisungen **INSERT**, **UPDATE** und **DELETE** können eine **RETURNING**-Klausel enthalten, die Spaltenwerte aus der betroffenen Zeile in PL/SQL-Variablen oder Hostvariablen zurückgibt. Dadurch entfällt die Notwendigkeit, die Zeile nach einem Vorgang vom Typ **INSERT** oder **UPDATE** bzw. vor einem **DELETE**-Vorgang mit **SELECT** zu wählen. Als Folge davon sind weniger Netzwerkumläufe (Roundtrips), Server-CPU-Zeit, Cursor und Serverspeicher erforderlich.

Das Beispiel auf der Folie zeigt, wie Sie das Gehalt eines Mitarbeiters aktualisieren und gleichzeitig dessen Nachnamen und neues Gehalt in eine lokale PL/SQL-Variable abrufen. Um das Codebeispiel zu testen, setzen Sie die folgenden Befehle ab, die das Gehalt von **employee_id** 108 vor der Aktualisierung prüfen. Danach wird die Prozedur aufgerufen und die **employee_id** 108 als Parameter übergeben.

```

1 SET SERVEROUTPUT ON
2 /
3 SELECT last_name, salary
4 FROM employees
5 WHERE employee_id = 108;
6 /
7 EXECUTE update_salary(108)

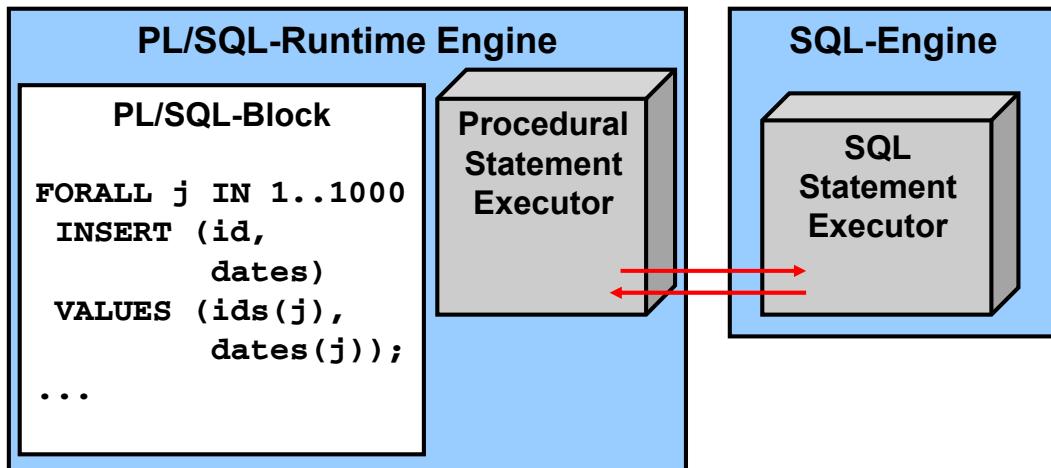
```

LAST_NAME	SALARY
Greenberg	12008

anonymous block completed
Greenberg new salary is 13208.8

Bulk Binding

Bindet ganze Arrays von Werten in einem einzigen Vorgang. Vorgänge vom Typ `FETCH`, `INSERT`, `UPDATE` und `DELETE` müssen somit nicht mehrmals mit einer Schleife ausgeführt werden.



ORACLE

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

Der Oracle-Server führt PL/SQL-Blöcke und -Unterprogramme mit zwei Engines aus:

- Die PL/SQL-Runtime Engine führt prozedurale Anweisungen aus, übergibt SQL-Anweisungen jedoch an die SQL-Engine.
- Die SQL-Engine parst die SQL-Anweisung, führt sie aus und gibt in bestimmten Fällen Daten an die PL/SQL-Engine zurück.

Bei der Ausführung verursacht jede SQL-Anweisung einen Kontextwechsel zwischen den beiden Engines, was zu einer exzessiven SQL-Verarbeitung und damit zu Performanceeinbußen führt. Dies ist typisch für Anwendungen mit einer SQL-Anweisung in einer Schleife, die Werte aus indizierten Collections verwendet. Zu Collections gehören Nested Tables, variable Arrays (VARARRAYs), INDEX BY-Tabellen und Hostarrays.

Eine geringere Anzahl an Kontextwechseln mithilfe von Bulk Binding kann eine erhebliche Performancesteigerung bewirken. Bulk Binding bindet eine ganze Collection mit einem einzigen Aufruf, einem Kontextwechsel, an die SQL-Engine. Beim Bulk Binding wird also die ganze Wertecollection in einem einzigen Kontextwechsel zwischen den beiden Engines hin- und herübertragen. Im Gegensatz dazu erfolgt bei der Wiederholung einer Schleife für jedes Element der Collection ein Kontextwechsel. Je mehr Zeilen von einer SQL-Anweisung betroffen sind, desto größer ist die Performancesteigerung durch Bulk Binding.

Bulk Binding – Syntax und Schlüsselwörter

- Das Schlüsselwort **FORALL** weist die *PL/SQL-Engine* an, Eingabe-Collections vor dem Senden per Bulk Binding an die *SQL-Engine* zu binden.

```
FORALL index IN lower_bound ... upper_bound
  [SAVE EXCEPTIONS]
  sql_statement;
```

- Das Schlüsselwort **BULK COLLECT** weist die *SQL-Engine* an, Ausgabe-Collections vor der Rückgabe per Bulk Binding an die *PL/SQL-Engine* zu binden.

```
... BULK COLLECT INTO
  collection_name [,collection_name] ...
```

ORACLE

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

Verbessern Sie mit Bulk Binding die Performance von:

- DML-Anweisungen, die Elemente von Collections referenzieren
- SELECT-Anweisungen, die Elemente von Collections referenzieren
- Cursor FOR-Schleifen, die Collections und die Klausel RETURNING INTO referenzieren

Das Schlüsselwort **FORALL** weist die *PL/SQL-Engine* an, Eingabe-Collections vor dem Senden an die *SQL-Engine* per Bulk Binding zu binden. Die Anweisung **FORALL** enthält zwar ein Wiederholungsschema, dieses ist jedoch nicht für eine **FOR**-Schleife gedacht.

Das Schlüsselwort **BULK COLLECT** weist die *SQL-Engine* an, Ausgabe-Collections vor der Rückgabe per Bulk Binding an die *PL/SQL-Engine* zu binden. Dadurch können Sie Speicherorte binden, an die SQL die abgerufenen Werte mit einem einzigen Vorgang zurückgeben kann. Die Schlüsselwörter lassen sich somit in den Klauseln **SELECT INTO**, **FETCH INTO** und **RETURNING INTO** verwenden.

Das Schlüsselwort **SAVE EXCEPTIONS** ist optional. Wenn allerdings einige DML-Vorgänge erfolgreich sind und andere nicht, möchten Sie sicherlich die misslungenen Vorgänge protokollieren bzw. entsprechende Berichte darüber erstellen. Mithilfe des Schlüsselwortes **SAVE EXCEPTIONS** können Sie nicht erfolgreiche Vorgänge in einem Cursorattribut namens **%BULK_EXCEPTIONS** speichern. Dies ist eine Collection mit Records, die die Bulk-DML-Wiederholungsnummer und den entsprechenden Fehlercode angeben.

FORALL Exceptions Exceptions mit dem Attribut %BULK_EXCEPTIONS behandeln

Um die Exceptions zu verwalten und das Bulk Binding auch bei Fehlern vollständig durchzuführen, fügen Sie nach den Bindebefehlen und vor der DML-Anweisung das Schlüsselwort `SAVE EXCEPTIONS` in die Anweisung `FORALL` ein.

Alle bei der Ausführung ausgelösten Exceptions werden im Cursorattribut `%BULK_EXCEPTIONS` gespeichert, das eine Collection von Records umfasst. Jeder Record enthält zwei Felder:

`%BULK_EXCEPTIONS (i).ERROR_INDEX` enthält die "Wiederholung" der Anweisung `FORALL`, bei der die Exception ausgelöst wurde, und `%BULK_EXCEPTIONS (i).ERROR_CODE` enthält den entsprechenden Oracle-Fehlercode.

Die in `%BULK_EXCEPTIONS` gespeicherten Werte verweisen auf die zuletzt ausgeführte Anweisung `FORALL`. Ihre untergeordneten Skripte haben einen Bereich von 1 bis `%BULK_EXCEPTIONS.COUNT`.

Hinweis: Weitere Informationen über Bulk Binding und die Behandlung von Bulk Binding Exceptions finden Sie im Dokument *Oracle Database PL/SQL User's Guide and Reference*.

Bulk Binding mit FORALL – Beispiel

```

CREATE PROCEDURE raise_salary(p_percent NUMBER) IS
    TYPE numlist_type IS TABLE OF NUMBER
        INDEX BY BINARY_INTEGER;
    v_id  numlist_type; -- collection
BEGIN
    v_id(1):= 100; v_id(2):= 102; v_id(3):= 104; v_id(4) := 110;
    -- bulk-bind the PL/SQL table
    FORALL i IN v_id.FIRST .. v_id.LAST
        UPDATE employees
            SET salary = (1 + p_percent/100) * salary
            WHERE employee_id = v_id(i);
END;
/

```

```
EXECUTE raise_salary(10)
```

anonymous block completed

ORACLE

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

Hinweis: Bevor Sie das auf der Folie gezeigte Beispiel ausführen können, müssen Sie den Trigger `update_job_history` wie folgt deaktivieren:

```
ALTER TRIGGER update_job_history DISABLE;
```

Im Beispiel auf der Folie erhöht der PL/SQL-Block das Gehalt der Mitarbeiter mit der ID 100, 102, 104 oder 110. Das Bulk Binding der Collection erfolgt mit dem Schlüsselwort `FORALL`. Ohne Bulk Binding hätte der PL/SQL-Block für jeden zu aktualisierenden Mitarbeiter-Record eine SQL-Anweisung an die SQL-Engine gesendet. Wenn viele Mitarbeiter-Records aktualisiert werden müssen, kann die große Anzahl der Kontextwechsel zwischen der PL/SQL-Engine und der SQL-Engine zu Performanceeinbußen führen. Das Schlüsselwort `FORALL` dagegen bindet die Collection per Bulk Binding, um die Performance zu steigern.

Hinweis: Bei Verwendung dieses Features ist kein Schleifenkonstrukt mehr erforderlich.

Zusätzliches Cursorattribut für DML-Vorgänge

Ein anderes, zur Unterstützung von Bulk-Vorgängen hinzugefügtes Cursorattribut ist %BULK_ROWCOUNT. Das Attribut %BULK_ROWCOUNT ist eine zusammengesetzte Struktur zur Verwendung mit der Anweisung FORALL. Dieses Attribut verhält sich wie eine INDEX BY-Tabelle. Sein *i*-tes Element speichert die Anzahl der von der *i*-ten Ausführung einer Anweisung UPDATE oder DELETE verarbeiteten Zeilen. Wenn die *i*-te Ausführung keine Zeilen betrifft, gibt %BULK_ROWCOUNT(*i*) null zurück.

Beispiel:

```

CREATE TABLE num_table (n NUMBER);
DECLARE
    TYPE num_list_type IS TABLE OF NUMBER
        INDEX BY BINARY_INTEGER;
    v_nums num_list_type;
BEGIN
    v_nums(1) := 1;
    v_nums(2) := 3;
    v_nums(3) := 5;
    v_nums(4) := 7;
    v_nums(5) := 11;
    FORALL i IN v_nums.FIRST .. v_nums.LAST
        INSERT INTO v_num_table (n) VALUES (v_nums(i));
    FOR i IN v_nums.FIRST .. v_nums.LAST
        LOOP
            dbms_output.put_line('Inserted ' ||
                SQL%BULK_ROWCOUNT(i) || ' row(s)' ||
                ' on iteration ' || i);
        END LOOP;
    END;
/
DROP TABLE num_table;

```

Die Ergebnisse dieses Beispiels lauten:

```

table NUM_TABLE created.
anonymous block completed
Inserted 1 row(s) on iteration 1
Inserted 1 row(s) on iteration 2
Inserted 1 row(s) on iteration 3
Inserted 1 row(s) on iteration 4
Inserted 1 row(s) on iteration 5

table NUM_TABLE dropped.

```

BULK COLLECT INTO mit Abfragen

Die Anweisung SELECT unterstützt die Syntax von BULK COLLECT INTO.

```
CREATE PROCEDURE get_departments(p_loc NUMBER) IS
  TYPE dept_tab_type IS
    TABLE OF departments%ROWTYPE;
  v_depts dept_tab_type;
BEGIN
  SELECT * BULK COLLECT INTO v_depts
  FROM departments
  WHERE location_id = p_loc;
  FOR i IN 1 .. v_depts.COUNT LOOP
    DBMS_OUTPUT.PUT_LINE(v_depts(i).department_id
      || ' ' || v_depts(i).department_name);
  END LOOP;
END;
```

ORACLE

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

Seit Oracle Database 10g können Sie für SELECT-Anweisungen in PL/SQL die im Beispiel auf der Folie gezeigte Syntax für Bulk Collections verwenden. Auf diese Weise lässt sich eine Zeilensammlung schnell ohne Cursorverfahren abrufen.

Das Beispiel liest alle Abteilungszeilen einer angegebenen Region in eine PL/SQL-Tabelle ein, deren Inhalt mit der auf die Anweisung SELECT folgenden Schleife FOR angezeigt wird.

BULK COLLECT INTO mit Cursorn

Die Anweisung **FETCH** wurde um die Unterstützung der Syntax für **BULK COLLECT INTO** erweitert.

```
CREATE OR REPLACE PROCEDURE get_departments(p_loc NUMBER) IS
  CURSOR cur_dept IS
    SELECT * FROM departments
    WHERE location_id = p_loc;
  TYPE dept_tab_type IS TABLE OF cur_dept%ROWTYPE;
  v_depts dept_tab_type;
BEGIN
  OPEN cur_dept;
  FETCH cur_dept BULK COLLECT INTO v_depts;
  CLOSE cur_dept;
  FOR i IN 1 .. v_depts.COUNT LOOP
    DBMS_OUTPUT.PUT_LINE(v_depts(i).department_id
      || ' ' || v_depts(i).department_name);
  END LOOP;
END;
```

ORACLE

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

Seit Oracle Database 10g können Sie für Cursor in PL/SQL eine Form der Anweisung **FETCH** verwenden, die die im Beispiel auf der Folie gezeigte Syntax für Bulk Collections unterstützt.

Das Beispiel zeigt, wie sich **BULK COLLECT INTO** mit Cursorn einsetzen lässt.

Sie können auch eine **LIMIT**-Klausel hinzufügen, um die Anzahl der in jedem Vorgang abgerufenen Zeilen zu steuern. Das Codebeispiel auf der Folie kann wie folgt geändert werden:

```
CREATE OR REPLACE PROCEDURE get_departments(p_loc NUMBER,
  p_nrows NUMBER) IS
  CURSOR dept_csr IS SELECT *
    FROM departments
    WHERE location_id = p_loc;
  TYPE dept_tabtype IS TABLE OF dept_csr%ROWTYPE;
  depts dept_tabtype;
BEGIN
  OPEN dept_csr;
  FETCH dept_csr BULK COLLECT INTO depts LIMIT nrows;
  CLOSE dept_csr;
  DBMS_OUTPUT.PUT_LINE(depts.COUNT||' rows read');
END;
```

BULK COLLECT INTO mit einer RETURNING-Klausel

```

CREATE OR REPLACE PROCEDURE raise_salary(p_rate NUMBER)
IS
  TYPE emplist_type IS TABLE OF NUMBER;
  TYPE numlist_type IS TABLE OF employees.salary%TYPE
    INDEX BY BINARY_INTEGER;
  v_emp_ids emplist_type :=
    emplist_type(100,101,102,104);
  v_new_sals numlist_type;
BEGIN
  FORALL i IN v_emp_ids.FIRST .. v_emp_ids.LAST
    UPDATE employees
      SET commission_pct = p_rate * salary
    WHERE employee_id = v_emp_ids(i)
    RETURNING salary BULK COLLECT INTO v_new_sals;
  FOR i IN 1 .. v_new_sals.COUNT LOOP
    DBMS_OUTPUT.PUT_LINE(v_new_sals(i));
  END LOOP;
END;

```

ORACLE

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

Bulk Binding kann die Performance von FOR-Schleifen verbessern, die Collections referenzieren und DML zurückgeben. Wenn Sie PL/SQL-Code haben oder schreiben möchten, der dies bewirkt, können Sie das Schlüsselwort FORALL gemeinsam mit den Schlüsselwörtern RETURNING und BULK COLLECT INTO zur Performancesteigerung verwenden.

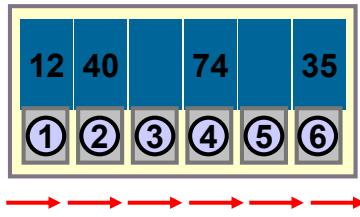
Im Beispiel auf der Folie werden die Gehaltsinformationen (`salary`) aus der Tabelle EMPLOYEES abgerufen und im Array `new_sals` gesammelt. Die Collection `new_sals` wird als Ganzes an die PL/SQL-Engine zurückgegeben.

Das Beispiel auf der Folie zeigt eine unvollständige FOR-Schleife zur Wiederholung durch die neuen, mit dem Vorgang UPDATE erhaltenen Gehaltsdaten und zur Verarbeitung der Ergebnisse.

Bulk Binding in wenig gefüllten Collections

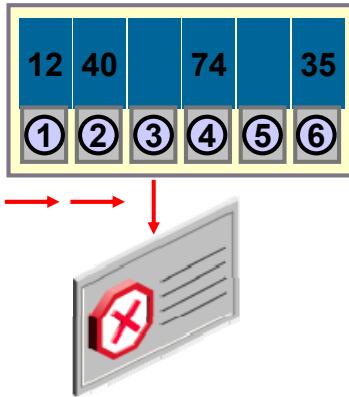
Aktuelle Releases:

```
INDEX BY
PLS_INTEGER
FOR ALL . . .
IN INDICES OF
```



Vor 10g:

```
INDEX BY
BINARY_INTEGER
```



ORACLE

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

FORALL-Unterstützung für wenig gefüllte Collections

- In früheren Releases gab es keine PL/SQL-Unterstützung für die Verwendung von wenig gefüllten Collections mit der Anweisung FORALL.
- Wenn das Schlüsselwort `SAVE EXCEPTIONS` nicht angegeben war, wurde die Anweisung beim ersten gelöschten Element abgebrochen. Auch beim Einsatz von `SAVE EXCEPTION` versuchte die PL/SQL-Engine, die Wiederholungen für alle (vorhandenen und nicht vorhandenen) Elemente auszuführen. Dies führte zu beträchtlichen Performanceeinbußen für den DML-Vorgang bei einem hohen relativen Prozentsatz der gelöschten Elemente.
- Wenn Sie das Schlüsselwort `INDICES` (verfügbar in Oracle Database 10g und höher) verwenden, können Sie eine wenig gefüllte Collection erfolgreich mit der Anweisung FORALL in Schleifen durchlaufen. Diese Syntax bindet wenig gefüllte Collections effizienter und unterstützt auch ein allgemeingültigeres Verfahren, bei dem ein Indexarray für die Wiederholung der Collections angegeben werden kann.
- Durch die Verwendung von wenig gefüllten Collections und Indexarrays in Bulk-Vorgängen lässt sich die Performance verbessern.

Bulk Binding in wenig gefüllten Collections

```
-- The INDICES OF syntax allows the bound arrays
-- themselves to be sparse.

FORALL index_name IN INDICES OF sparse_array_name
    BETWEEN LOWER_BOUND AND UPPER_BOUND -- optional
    SAVE EXCEPTIONS -- optional, but recommended
    INSERT INTO table_name VALUES
        sparse_array(index_name);
    ...

```

```
-- The VALUES OF syntax lets you indicate a subset
-- of the binding arrays.

FORALL index_name IN VALUES OF index_array_name
    SAVE EXCEPTIONS -- optional, but recommended
    INSERT INTO table_name VALUES
        binding_array_name(index_name);
    ...

```

ORACLE

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

FORALL-Unterstützung für wenig gefüllte Collections

- Sie können die Syntax von `INDICES OF` und `VALUES OF` mit der Anweisung `FORALL` verwenden.
- Die Syntax zum Bulk Binding für wenig gefüllte Arrays kann in jeder DML-Syntax eingesetzt werden.
- In der Syntax gilt: Das Indexarray muss dicht gefüllt sein, die Binding-Arrays können dicht oder wenig gefüllt sein, und die indizierten Elemente müssen vorhanden sein.

Bulk Binding in wenig gefüllten Collections

Die typische Anwendung für dieses Feature ist ein Auftragseingabe- und Auftragsabwicklungssystem, in dem:

- Benutzer über das Web Aufträge eingeben
- Aufträge vor der Validierung in einer Staging-Tabelle gespeichert werden
- Daten später auf Grundlage komplexer Geschäftsregeln validiert werden (normalerweise mit PL/SQL programmgesteuert implementiert)
- ungültige Aufträge von gültigen Aufträgen getrennt werden
- gültige Aufträge für die Verarbeitung in eine permanente Tabelle eingefügt werden

ORACLE

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

Dieses Feature kann in jeder Anwendung genutzt werden, die von einer dicht gefüllten PL/SQL-Record-Tabelle oder skalaren Tabelle ausgeht, die mit Bulk Collect gefüllt wurde. Diese wird als Binding-Array verwendet. Ein dicht gefülltes (Pointer-)Array, dessen Elemente die Indizes des Binding-Arrays angeben, wird auf Grundlage der Anwendungslogik in ein wenig gefülltes Array umgewandelt. Dieses Pointer-Array wird dann in der Anweisung `FORALL` verwendet, um Bulk-DML mit den Binding-Arrays auszuführen. Eventuell auftretende Exceptions können gespeichert und im Bereich zur Exception-Behandlung weiter verarbeitet werden, möglicherweise mithilfe einer weiteren `FORALL`-Anweisung.

Bulk Binding mit Indexarray

```
CREATE OR REPLACE PROCEDURE ins_emp2 AS
  TYPE emptab_type IS TABLE OF employees%ROWTYPE;
  v_emp emptab_type;
  TYPE values_of_tab_type IS TABLE OF PLS_INTEGER
    INDEX BY PLS_INTEGER;
  v_num   values_of_tab_type;
  . . .
BEGIN
  . . .
  FORALL k IN VALUES OF v_num
    INSERT INTO new_employees VALUES v_emp(k);
END;
```

ORACLE

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

Sie können eine Index-Collection vom Typ PLS_INTEGER oder BINARY_INTEGER (oder einer ihrer Subtypen) verwenden, deren Werte die Indizes für die am Bulk Binding-DML-Vorgang mit FORALL beteiligten Collections sind. Diese Index-Collections können dann in einer FORALL-Anweisung zur Verarbeitung von Bulk-DML mit der Klausel VALUES OF verwendet werden.

Im Beispiel oben ist V_NUM eine Collection vom Typ PLS_INTEGER. Das Beispiel zeigt die Erstellung einer Prozedur namens INS_EMP2, die pro Anfangsbuchstabe des Nachnamens nur jeweils einen Mitarbeiter angibt. Diese werden dann von der Prozedur mit der Syntax FORALL .. IN VALUES OF in die zuvor erstellte Tabelle NEW_EMPLOYEES eingefügt.

Quiz

Der Hint `NOCOPY` ermöglicht dem PL/SQL-Compiler die Übergabe der Parameter `OUT` und `IN OUT` per Referenz anstatt mit einem Wert. Dies erhöht die Performance durch geringeren Overhead beim Übergeben von Parametern.

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

ORACLE

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

Richtige Antwort: a

PL/SQL-Unterprogramme unterstützen drei Modi der Parameterübergabe: `IN`, `OUT` und `IN OUT`.

Standardmäßig gilt:

- Der Parameter `IN` wird per Referenz übergeben. Ein Zeiger auf den tatsächlichen Parameter `IN` wird an den entsprechenden formalen Parameter übergeben. Somit referenzieren beide Parameter denselben Speicherort, der den Wert des tatsächlichen Parameters enthält.
- Die Parameter `OUT` und `IN OUT` werden mit einem Wert übergeben. Der Wert des tatsächlichen Parameters `OUT` oder `IN OUT` wird dabei in die entsprechenden formalen Parameter kopiert. Anschließend werden, sofern das Unterprogramm ordnungsgemäß beendet wird, die den formalen Parametern `OUT` und `IN OUT` zugewiesenen Werte in die entsprechenden tatsächlichen Parameter kopiert.

Das Kopieren von Parametern, die für große Datenstrukturen stehen (z. B. Collections, Records oder Objekttypinstanzen), mit den Parametern `OUT` und `IN OUT` führt zu einer verlangsamen Ausführung und einem erhöhten Speicherbedarf. Zur Vermeidung dieses Overheads können Sie den Hint `NOCOPY` angeben, der dem PL/SQL-Compiler die Übergabe derartiger Parameter per Referenz ermöglicht.

Zusammenfassung

In dieser Lektion haben Sie Folgendes gelernt:

- Standardkonstanten und Exceptions erstellen
- Lokale Unterprogramme erstellen und aufrufen
- Laufzeitberechtigungen von Unterprogrammen steuern
- Autonome Transaktionen ausführen
- PL/SQL-Packages und Stored Subprograms Rollen erteilen
- Parameter mit dem Hint NOCOPY per Referenz übergeben
- Mit dem Hint PARALLEL_ENABLE eine Optimierung vornehmen
- Sessionübergreifenden Ergebniscache für PL/SQL-Funktionen verwenden
- Klausel DETERMINISTIC mit Funktionen verwenden
- Bulk Binding und die Klausel RETURNING mit DML verwenden



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

Diese Lektion bot einen Einblick in die Verwaltung von PL/SQL-Code durch das Definieren von Konstanten und Exceptions in einer Packagespezifikation. Dadurch wird der Code in hohem Maße wiederverwendbar und standardisiert.

Mithilfe lokaler Unterprogramme können Sie Codeblöcke vereinfachen und in Module unterteilen. Dabei wird die Funktionalität der Unterprogramme mehrmals im lokalen Block verwendet.

Sie können die Sicherheitsberechtigungen von Unterprogrammen für die Laufzeit mit Rechten des Eigentümers oder des ausführenden Benutzers ändern.

Autonome Transaktionen können ohne Auswirkungen auf eine vorhandene Haupttransaktion ausgeführt werden.

Sie wissen jetzt, wie Sie mit dem Hint NOCOPY, Bulk Binding und den RETURNING-Klauseln in SQL-Anweisungen sowie mit dem Hint PARALLEL_ENABLE zur Optimierung von Funktionen die Performance steigern können.

Übungen zu Lektion 8 – Überblick: Überlegungen zum Design von PL/SQL-Code

Diese Übungen behandeln folgende Themen:

- Package erstellen, das Bulk-Abrufvorgänge verwendet**
- Lokales Unterprogramm erstellen, um mit einer autonomen Transaktion einen Geschäftsvorgang zu prüfen**

ORACLE

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

In dieser Übung erstellen Sie ein Package, das einen Bulk-Abruf der Mitarbeiter aus einer bestimmten Abteilung durchführt. Die Daten werden im Package in einer PL/SQL-Tabelle gespeichert. Außerdem stellen Sie eine Prozedur zum Anzeigen des Tabelleninhalts bereit.

Sie fügen dem Package eine Prozedur namens `add_employee` hinzu, um neue Mitarbeiter einzufügen. Diese Prozedur schreibt mit einem lokalen, autonomen Unterprogramm bei jedem Aufruf der Prozedur `add_employee` einen Log-Record, unabhängig davon, ob ein Record erfolgreich hinzugefügt wurde oder nicht.

9

Trigger erstellen

ORACLE

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

Ziele

Nach Ablauf dieser Lektion haben Sie folgende Ziele erreicht:

- **Datenbanktrigger und ihre Verwendung beschreiben**
- **Verschiedene Triggertypen beschreiben**
- **Datenbanktrigger erstellen**
- **Regeln für das Auslösen von Triggern beschreiben**
- **Datenbanktrigger entfernen**
- **Triggerinformationen anzeigen**

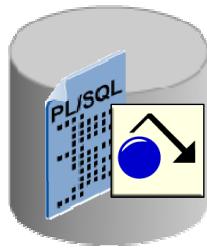
ORACLE

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

In dieser Lektion wird beschrieben, wie Sie Datenbanktrigger erstellen und verwenden.

Was sind Trigger?

- Ein Trigger ist ein PL/SQL-Block, der in der Datenbank gespeichert ist und als Reaktion auf ein bestimmtes Ereignis ausgelöst (ausgeführt) wird.
- Die Oracle-Datenbank führt Trigger automatisch aus, sobald eine vorgegebene Bedingung eintritt.



ORACLE

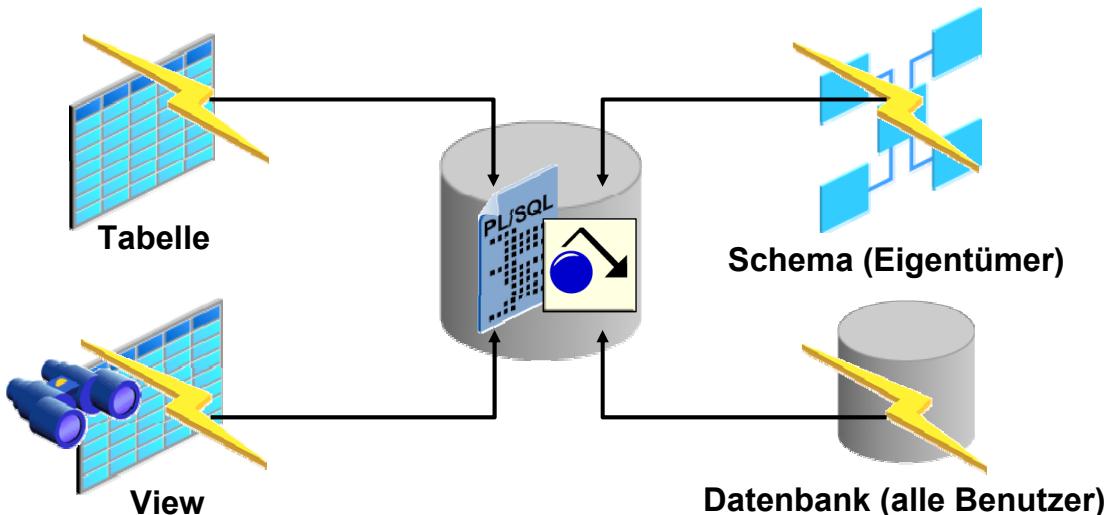
Copyright © 2013, Oracle und/oder verbundene Unternehmen. All rights reserved. Alle Rechte vorbehalten.

Mit Triggern arbeiten – Überblick

Trigger ähneln Stored Procedures. Ein in der Datenbank gespeicherter Trigger enthält PL/SQL in Form eines anonymen Blockes, einer CALL-Anweisung oder eines komplexen Triggerblocks. Allerdings unterscheiden sich Prozeduren und Trigger in der Art ihres Aufrufs. Während Prozeduren explizit von einem Benutzer, einer Anwendung oder einem Trigger ausgeführt werden, werden Trigger implizit von der Oracle-Datenbank ausgelöst, sobald ein bestimmtes auslösendes Ereignis eintritt. Dabei spielt es keine Rolle, welcher Benutzer angemeldet ist oder welche Anwendung verwendet wird.

Trigger definieren

Trigger können für eine Tabelle, eine View, ein Schema (den Schemaeigentümer) oder die Datenbank (alle Benutzer) definiert werden.



ORACLE

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

Triggerereignistypen

Sie können Trigger schreiben, die ausgelöst werden, sobald einer der folgenden Vorgänge in der Datenbank ausgeführt wird:

- Eine Database Manipulation Language-(DML-)Anweisung (`DELETE`, `INSERT` oder `UPDATE`)
- Eine Database Definition Language-(DDL-)Anweisung (`CREATE`, `ALTER` oder `DROP`)
- Ein Datenbankvorgang wie `SERVERERROR`, `LOGON`, `LOGOFF`, `STARTUP` oder `SHUTDOWN`.



ORACLE

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

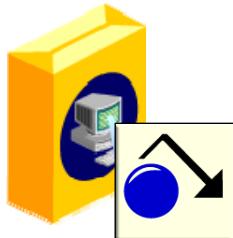
Auslösende Ereignisse oder Anweisungen

Als auslösende Ereignisse bzw. Anweisungen werden die SQL-Anweisungen, Datenbank- oder Benutzerereignisse bezeichnet, die den Trigger zur Ausführung anstoßen. Ein auslösendes Ereignis kann aus einem oder mehreren der folgenden Vorgänge bestehen:

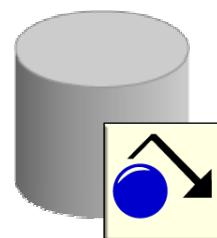
- Anweisung vom Typ `INSERT`, `UPDATE` oder `DELETE` für eine bestimmte Tabelle (oder in einigen Fällen auch eine View)
- Anweisung vom Typ `CREATE`, `ALTER` oder `DROP` für ein beliebiges Schemaobjekt
- Hoch- oder Herunterfahren einer Datenbankinstanz
- Bestimmte oder beliebige Fehlermeldung
- An- oder Abmeldung eines Benutzers

Anwendungs- und Datenbanktrigger

- **Datenbanktrigger (Thema dieses Kurses):**
 - werden ausgelöst, sobald ein DML-, DLL- oder Systemereignis in einem Schema oder einer Datenbank eintritt
- **Anwendungstrigger:**
 - werden ausgelöst, sobald in einer bestimmten Anwendung ein Ereignis eintritt



Anwendungstrigger



Datenbanktrigger

ORACLE

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

Triggertypen

Anwendungstrigger werden implizit immer dann ausgeführt, wenn ein bestimmtes DML-Ereignis innerhalb einer Anwendung eintritt. Beispielsweise werden Trigger häufig in Anwendungen verwendet, die mit Oracle Forms Developer entwickelt wurden.

Die Ausführung von Datenbanktriggern erfolgt implizit, wenn eines der folgenden Ereignisse eintritt:

- DML-Vorgänge in einer Tabelle
- DML-Vorgänge in einer View mit einem INSTEAD OF-Trigger
- DDL-Anweisungen, beispielsweise CREATE und ALTER

Die Trigger werden unabhängig davon ausgeführt, welcher Benutzer angemeldet ist oder welche Anwendung verwendet wird. Datenbanktrigger werden auch implizit ausgeführt, wenn bestimmte Benutzeraktionen oder Aktionen des Datenbanksystems eintreten. (Beispiel: Ein Benutzer meldet sich an, oder der Datenbankadministrator fährt die Datenbank herunter.)

Datenbanktrigger können Systemtrigger für eine Datenbank oder ein Schema sein. (Dieses Thema wird in der nächsten Lektion behandelt.) Im Falle von Datenbanken werden Trigger bei jedem Ereignis für alle Benutzer ausgelöst. Im Falle von Schemas werden die Trigger hingegen bei jedem Ereignis für den jeweiligen Benutzer ausgelöst. Oracle Forms kann ebenfalls Trigger definieren, speichern und ausführen. Die Oracle Forms-Trigger sind jedoch nicht mit den in dieser Lektion behandelten Triggern zu verwechseln.

Trigger implementieren – Szenarios für Geschäftsanwendungen

Sie können Trigger für folgende Zwecke einsetzen:

- **Sicherheit**
- **Auditing**
- **Datenintegrität**
- **Referenzielle Integrität**
- **Tabellenreplikation**
- **Automatisches Berechnen abgeleiteter Daten**
- **Ereignisprotokollierung**

ORACLE

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

Sie entwickeln Datenbanktrigger, um Features zu verbessern, die andernfalls vom Oracle-Server nicht implementiert werden können, oder um Alternativen zu den vom Oracle-Server bereitgestellten Features anzubieten.

- **Sicherheit:** Der Oracle-Server ermöglicht Benutzern oder Rollen den Tabellenzugriff. Trigger lassen einen Tabellenzugriff entsprechend den jeweiligen Datenwerten zu.
- **Auditing:** Der Oracle-Server überwacht die Datenvorgänge in Tabellen. Trigger überwachen die Werte der Datenvorgänge in Tabellen.
- **Datenintegrität:** Der Oracle-Server setzt Integritäts-Constraints durch. Trigger implementieren komplexe Integritätsregeln.
- **Referenzielle Integrität:** Der Oracle-Server erzwingt standardmäßige referenzielle Integritätsregeln. Trigger implementieren eine Nicht-Standardfunktionalität.
- **Tabellenreplikation:** Der Oracle-Server kopiert die Tabellen asynchron in Snapshots. Trigger kopieren die Tabellen synchron in Replikate.
- **Abgeleitete Daten:** Der Oracle-Server berechnet die Werte abgeleiteter Daten manuell. Trigger berechnen die Werte abgeleiteter Daten automatisch.
- **Ereignisprotokollierung:** Der Oracle-Server protokolliert Ereignisse explizit. Trigger protokollieren Ereignisse transparent.

Verfügbare Triggertypen

- **Einfache DML-Trigger**
 - BEFORE
 - AFTER
 - INSTEAD OF
- **Komplexe Trigger**
- **Nicht-DML-Trigger**
 - DDL-Ereignistrigger
 - Datenbankereignistrigger



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

Hinweis

In dieser Lektion werden die Trigger BEFORE, AFTER und INSTEAD OF erläutert. Die übrigen Triggertypen werden in der Lektion "Komplexe, DDL- und Datenbankereignistrigger erstellen" behandelt.

Triggerereignistypen und Triggerbody

- **Der Triggerereignistyp entscheidet darüber, welche DML-Anweisung den Trigger auslöst. Mögliche Ereignisse sind:**
 - **INSERT**
 - **UPDATE [OF column]**
 - **DELETE**
- **Der Triggerbody besteht aus einem PL/SQL-Block oder einem Prozeduraufruf und entscheidet darüber, welche Aktion ausgeführt wird.**

ORACLE

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

Triggerereignistypen und Triggerbody

Das auslösende Ereignis kann eine Anweisung vom Typ **INSERT**, **UPDATE** oder **DELETE** für eine Tabelle sein.

- Wenn das auslösende Ereignis eine **UPDATE**-Anweisung ist, können Sie eine Liste von Spalten aufnehmen und damit angeben, welche Spalten geändert werden müssen, um den Trigger auszulösen. Für die Anweisungen **INSERT** und **DELETE** lässt sich keine Spaltenliste angeben, da diese Anweisungen immer ganze Zeilen betreffen.
 `. . . UPDATE OF salary . . .`
- Das auslösende Ereignis kann eine, zwei oder alle drei der folgenden DML-Vorgänge enthalten.
 `. . . INSERT or UPDATE or DELETE`
 `. . . INSERT or UPDATE OF job_id . . .`

Der Triggerbody definiert die Aktion, das heißt, die bei Eintritt des auslösenden Ereignisses auszuführenden Aufgaben. Der PL/SQL-Block kann SQL- und PL/SQL-Anweisungen enthalten und PL/SQL-Konstrukte definieren, beispielsweise Variablen, Cursor oder Exceptions. Sie können auch eine PL/SQL-Prozedur oder eine Java-Prozedur aufrufen.

Hinweis: Die Größe eines Triggers darf 32 KB nicht überschreiten.

DML-Trigger mit der Anweisung CREATE TRIGGER erstellen

```
CREATE [OR REPLACE] TRIGGER trigger_name
  timing -- when to fire the trigger
  event1 [OR event2 OR event3]
  ON object_name
  [REFERENCING OLD AS old / NEW AS new]
  FOR EACH ROW -- default is statement level trigger
  WHEN (condition)]
  DECLARE]
  BEGIN
    ... trigger_body -- executable statements
  [EXCEPTION . . .]
  END [trigger_name];
```

timing = BEFORE | AFTER | INSTEAD OF

event = INSERT | DELETE | UPDATE | UPDATE OF column_list

ORACLE

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

DML-Trigger erstellen

Die Triggersyntax setzt sich wie folgt zusammen:

- *trigger_name* identifiziert den Trigger eindeutig.
- *timing* gibt an, wann der Trigger im Verhältnis zum auslösenden Ereignis ausgelöst wird.
Werte sind BEFORE, AFTER oder INSTEAD OF.
- *event* identifiziert den DML-Vorgang, der den Trigger auslöst.
Werte sind INSERT, UPDATE [OF column] und DELETE.
- *object_name* gibt die Tabelle oder View an, die mit dem Trigger verknüpft ist.
- Für Row Trigger können Sie Folgendes angeben:
 - Eine REFERENCING-Klausel zum Auswählen von Korrelationsnamen, um die alten und neuen Werte der aktuellen Zeile zu referenzieren (Standardwerte sind OLD und NEW.)
 - FOR EACH ROW, um den Trigger als Row Trigger anzugeben
 - Eine WHEN-Klausel, mit der ein Bedingungsprädikat (in Klammern) angeben wird. Das Bedingungsprädikat wird für jede Zeile ausgewertet, um zu bestimmen, ob der Triggerbody ausgeführt wird.

- *trigger_body* ist die vom Trigger ausgeführte Aktion, die mit einer der folgenden Möglichkeiten implementiert wird:
 - Als anonymer Block, der mit `DECLARE` oder `BEGIN` beginnt und mit `END` endet
 - Mit einer `CALL`-Klausel, die eine Stored Procedure (Standalone oder in ein Package integriert) aufruft. Beispiel:

```
CALL my_procedure;
```

Triggerauslösung angeben (Timing)

Sie können festlegen, ob die Triggeraktion vor oder nach der auslösenden Anweisung ausgeführt werden soll:

- **BEFORE:** Führt den Triggerbody vor dem auslösenden DML-Ereignis für eine Tabelle aus
- **AFTER:** Führt den Triggerbody nach dem auslösenden DML-Ereignis für eine Tabelle aus
- **INSTEAD OF:** Führt den Triggerbody anstelle der auslösenden Anweisung aus. Diese Funktion wird für Views verwendet, die sonst nicht geändert werden können.



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

Triggertiming

BEFORE-Trigger werden häufig in folgenden Situationen verwendet:

- Zur Ermittlung, ob die auslösende Anweisung beendet werden darf. (Dadurch werden unnötige Verarbeitungen vermieden. In Fällen, in denen eine Exception in der auslösenden Aktion ausgelöst wird, kann zudem ein Rollback durchgeführt werden.)
- Zur Ableitung von Spaltenwerten, bevor eine `INSERT`- oder `UPDATE`-Anweisung abgeschlossen ist
- Zur Initialisierung von globalen Variablen oder Flags und zur Validierung komplexer Geschäftsregeln

AFTER-Trigger werden häufig in folgenden Situationen verwendet:

- Zum Abschließen der auslösenden Anweisung, bevor die auslösende Aktion ausgeführt wird
- Zur Durchführung verschiedener Aktionen mit derselben auslösenden Anweisung, wenn bereits ein BEFORE Trigger vorhanden ist

INSTEAD OF-Trigger bieten eine transparente Möglichkeit, Views zu ändern, die nicht zu den direkt mit SQL-DML-Anweisungen bearbeitbaren Views zählen. Sie können entsprechende DML-Anweisungen im Body eines INSTEAD OF-Triggers erstellen, um Aktionen direkt in den zugrunde liegenden View-Tabellen auszuführen.

Sofern sinnvoll, können Sie einzelne Trigger, die zu unterschiedlichen Zeitpunkten ausgelöst werden sollen, auch durch einen einzelnen, komplexen Trigger ersetzen, der die Aktionen explizit in der beabsichtigten Reihenfolge codiert. Wenn zwei oder mehr Trigger für denselben Ausführungszeitpunkt definiert werden und die Reihenfolge ihrer Auslösung von Bedeutung ist, können Sie die Auslösereihenfolge über die Klauseln `FOLLOWERS` und `PRECEDES` steuern.

Trigger auf Anweisungs- bzw. Zeilenebene – Vergleich

Trigger auf Anweisungsebene (Statement Trigger)	Trigger auf Zeilenebene (Row Trigger)
Sind die standardmäßig erstellten Trigger	Verwenden die Klausel FOR EACH ROW bei der Triggererstellung.
Werden einmal für das auslösende Ereignis ausgelöst	Werden einmal für jede vom auslösenden Ereignis betroffene Zeile ausgelöst
Werden einmalig ausgelöst, auch wenn keine Zeilen betroffen sind	Werden nicht ausgelöst, wenn keine Zeilen vom auslösenden Ereignis betroffen sind

ORACLE

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

DML-Triggertypen

Sie können angeben, ob der Trigger einmal für jede Zeile ausgeführt werden soll, die von der auslösenden Anweisung betroffen ist (z. B. bei einer Multiple Row-Anweisung vom Typ **UPDATE**), oder einmalig für die auslösende Anweisung, unabhängig davon, wie viele Zeilen betroffen sind.

Statement Trigger

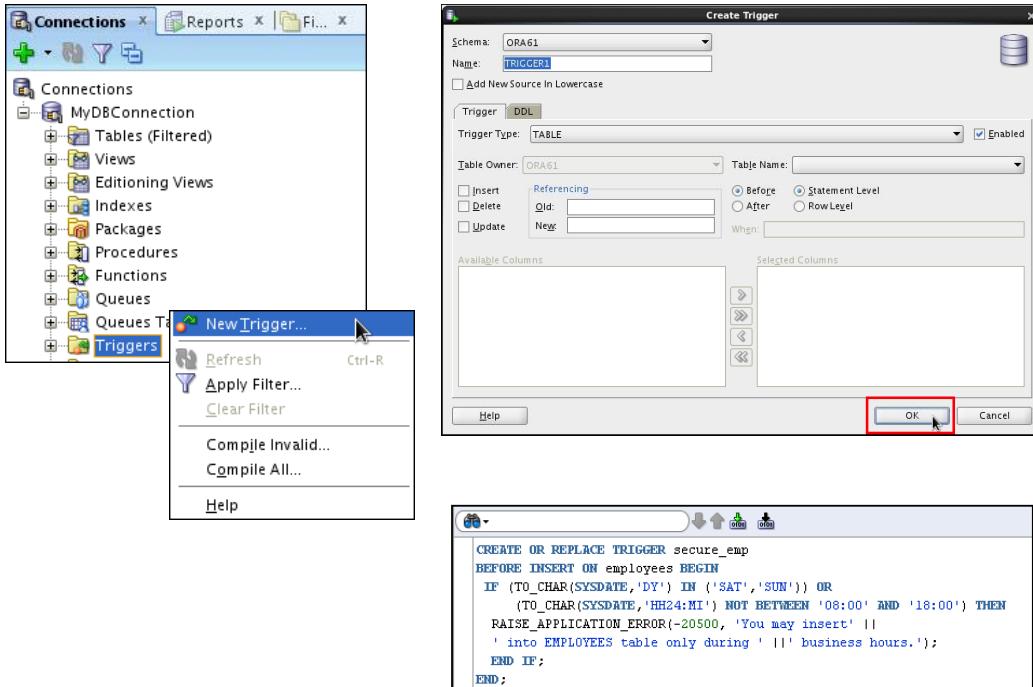
Ein Statement Trigger wird einmal für das auslösende Ereignis ausgelöst, selbst wenn keine Zeilen betroffen sind. Statement Trigger sind nützlich, wenn die Aktion des Triggers nicht von Daten aus betroffenen Zeilen abhängt oder von Daten, die durch das auslösende Ereignis selbst bereitgestellt werden (beispielsweise ein Trigger, der eine komplexe Sicherheitsprüfung für den aktuellen Benutzer durchführt).

Row Trigger

Ein Row Trigger wird jedes Mal ausgelöst, wenn die Tabelle vom auslösenden Ereignis betroffen ist. Wenn das auslösende Ereignis keine Zeilen betrifft, wird der Row Trigger nicht ausgeführt. Row Trigger sind nützlich, wenn die Aktion des Triggers von Daten aus betroffenen Zeilen abhängt oder von Daten, die durch das auslösende Ereignis selbst bereitgestellt werden.

Hinweis: Row Trigger verwenden Korrelationsnamen, um auf die alten und neuen Spaltenwerte der Zeile zuzugreifen, die durch den Trigger verarbeitet wird.

DML-Trigger mit SQL Developer erstellen



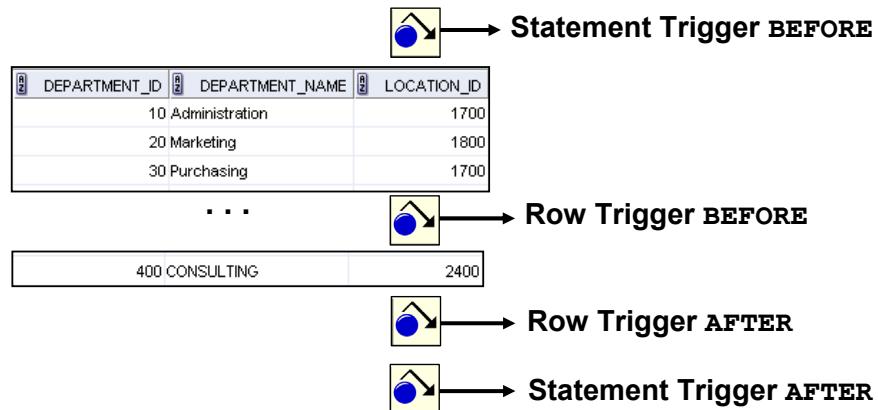
ORACLE

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

Auslösereihenfolge der Trigger – Single Row-Bearbeitung

Auslösereihenfolge für einen Tabellentrigger, wenn eine einzelne Zeile bearbeitet wird:

```
INSERT INTO departments
  (department_id, department_name, location_id)
VALUES (400, 'CONSULTING', 2400);
```



ORACLE

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

Ob Sie einen Statement Trigger oder Row Trigger erstellen, hängt davon ab, ob der Trigger für jede von der auslösenden Anweisung betroffene Zeile ausgelöst werden soll oder lediglich einmal für die auslösende Anweisung, unabhängig von der Anzahl der betroffenen Zeilen.

Wenn die auslösende DML-Anweisung eine einzelne Zeile betrifft, wird sowohl der Statement Trigger als auch der Row Trigger genau einmal ausgelöst.

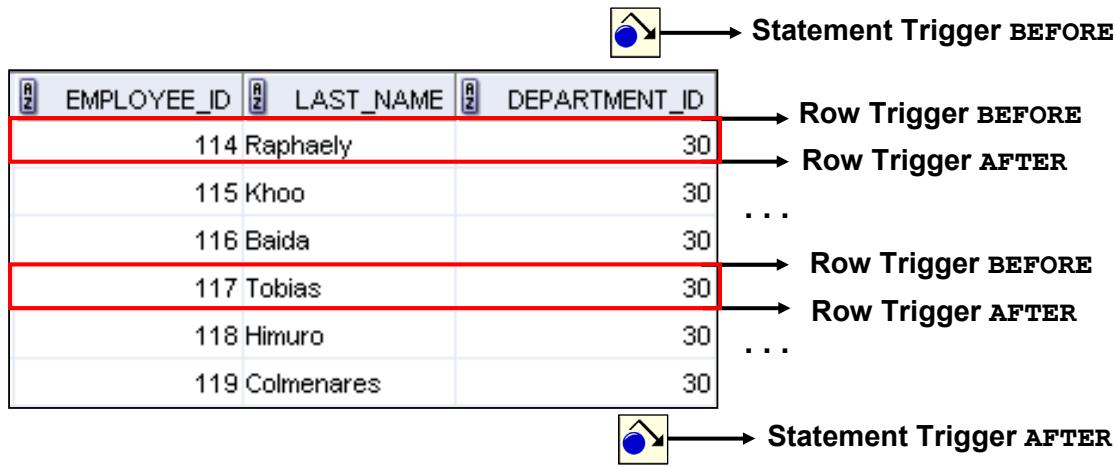
Beispiel

Die SQL-Anweisung auf der Folie unterscheidet nicht zwischen Statement Triggern und Row Triggern, da genau eine Zeile in die Tabelle eingefügt wurde, wobei die angezeigte Anweisungssyntax `INSERT` verwendet wurde.

Auslösereihenfolge der Trigger – Multiple Row-Bearbeitung

Auslösereihenfolge für einen Tabellentrigger, wenn mehrere Zeilen bearbeitet werden:

```
UPDATE employees
  SET salary = salary * 1.1
 WHERE department_id = 30;
```



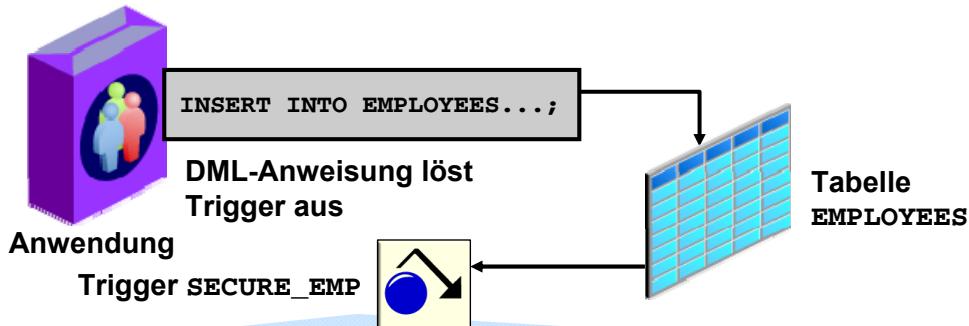
Copyright © 2013, Oracle und/oder verbundene Unternehmen. All rights reserved. Alle Rechte vorbehalten.

Wenn die auslösende DML-Anweisung mehrere Zeilen betrifft, wird ein Statement Trigger genau einmal ausgelöst und ein Row Trigger einmal für jede von der Anweisung betroffene Zeile.

Beispiel

Mit der SQL-Anweisung auf der Folie wird ein Trigger auf Zeilenebene mehrmals entsprechend der Anzahl von Zeilen ausgelöst, die die Klausel WHERE erfüllen, also entsprechend der Anzahl der Mitarbeiter in Abteilung 30.

DML-Statement Trigger erstellen – Beispiel: SECURE_EMP



```
CREATE OR REPLACE TRIGGER secure_emp
BEFORE INSERT ON employees
BEGIN
  IF (TO_CHAR(SYSDATE, 'DY') IN ('SAT','SUN')) OR
    (TO_CHAR(SYSDATE, 'HH24:MI') NOT BETWEEN '08:00' AND '18:00') THEN
    RAISE_APPLICATION_ERROR(-20500, 'You may insert'
    || ' into EMPLOYEES table only during '
    || ' normal business hours.');
  END IF;
END;
```

ORACLE

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

DML-Statement Trigger erstellen

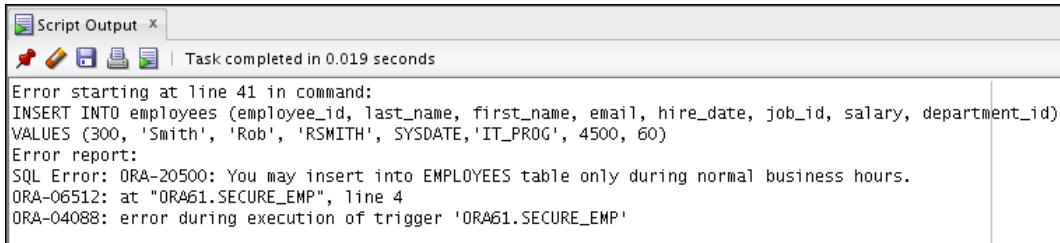
Im Beispiel auf der Folie ist der Datenbanktrigger SECURE_EMP ein BEFORE-Statement Trigger, der die erfolgreiche Ausführung des Vorgangs INSERT verhindert, wenn gegen die Geschäftsbedingung verstößen wird. In diesem Fall beschränkt der Trigger INSERT-Vorgänge in der Tabelle EMPLOYEES auf bestimmte Geschäftszeiten von Montag bis Freitag.

Wenn ein Benutzer am Samstag versucht, eine Zeile in die Tabelle EMPLOYEES einzufügen, wird eine Fehlermeldung ausgegeben. Der Trigger ist nicht erfolgreich, und die auslösende Anweisung wird zurückgerollt. RAISE_APPLICATION_ERROR ist eine serverseitige Built-In-Prozedur, die eine Fehlermeldung an den Benutzer zurückgibt und bewirkt, dass der PL/SQL-Block nicht erfolgreich ist.

Wenn ein Datenbanktrigger nicht erfolgreich ist, wird die auslösende Anweisung automatisch vom Oracle-Server zurückgerollt.

SECURE_EMP-Trigger testen

```
INSERT INTO employees (employee_id, last_name,
    first_name, email, hire_date, job_id, salary,
    department_id)
VALUES (300, 'Smith', 'Rob', 'RSMITH', SYSDATE,
    'IT_PROG', 4500, 60);
```



The screenshot shows the 'Script Output' window from Oracle SQL Developer. It displays the following text:

```
Script Output X
Task completed in 0.019 seconds
Error starting at Line 41 in command:
INSERT INTO employees (employee_id, last_name, first_name, email, hire_date, job_id, salary, department_id)
VALUES (300, 'Smith', 'Rob', 'RSMITH', SYSDATE, 'IT_PROG', 4500, 60)
Error report:
SQL Error: ORA-20500: You may insert into EMPLOYEES table only during normal business hours.
ORA-06512: at "ORAG1.SECURE_EMP", line 4
ORA-04088: error during execution of trigger 'ORAG1.SECURE_EMP'
```

ORACLE

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

SECURE_EMP testen

Um den Trigger zu testen, fügen Sie außerhalb der normalen Geschäftszeiten eine Zeile in die Tabelle EMPLOYEES ein. Wenn Datum und Uhrzeit außerhalb der im Trigger angegebenen Geschäftszeiten liegen, wird die auf der Folie gezeigte Fehlermeldung ausgegeben.

Bedingungsprädikate

```

CREATE OR REPLACE TRIGGER secure_emp BEFORE
INSERT OR UPDATE OR DELETE ON employees
BEGIN
  IF (TO_CHAR(SYSDATE,'DY') IN ('SAT','SUN')) OR
    (TO_CHAR(SYSDATE,'HH24') NOT BETWEEN '08' AND '18') THEN
    IF DELETING THEN RAISE_APPLICATION_ERROR(
      -20502,'You may delete from EMPLOYEES table'|||
      'only during normal business hours.');
    ELSIF INSERTING THEN RAISE_APPLICATION_ERROR(
      -20500,'You may insert into EMPLOYEES table'|||
      'only during normal business hours.');
    ELSIF UPDATING ('SALARY') THEN
      RAISE_APPLICATION_ERROR(-20503, 'You may '|||
      'update SALARY only normal during business hours.');
    ELSE RAISE_APPLICATION_ERROR(-20504,'You may' |||
      ' update EMPLOYEES table only during' |||
      ' normal business hours.');
    END IF;
  END IF;
END;

```

ORACLE

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

Triggerauslösenden DML-Vorgang ermitteln

Wenn unterschiedliche Typen von DML-Vorgängen einen Trigger auslösen können (z. B. ON INSERT OR DELETE OR UPDATE OF Emp_tab), kann der Triggerbody mithilfe der Bedingungsprädikate INSERTING, DELETING und UPDATING prüfen, welcher Anweisungstyp den Trigger ausgelöst hat.

Sie können mehrere auslösende Ereignisse zu einem Ereignis kombinieren, indem Sie die speziellen Bedingungsprädikate INSERTING, UPDATING und DELETING im Triggerbody nutzen.

Beispiel

Sie erstellen einen Trigger, um alle DML-Ereignisse für die Tabelle EMPLOYEES auf bestimmte Geschäftszeiten von Montag bis Freitag (8 bis 18 Uhr) zu beschränken.

DML-Row Trigger erstellen

```
CREATE OR REPLACE TRIGGER restrict_salary
BEFORE INSERT OR UPDATE OF salary ON employees
FOR EACH ROW
BEGIN
    IF NOT (:NEW.job_id IN ('AD_PRES', 'AD_VP'))
        AND :NEW.salary > 15000 THEN
        RAISE_APPLICATION_ERROR (-20202,
            'Employee cannot earn more than $15,000.');
    END IF;
END;
```

```
UPDATE employees
SET salary = 15500
WHERE last_name = 'Russell';
```

```
Error starting at line 1 in command:
UPDATE employees
SET salary = 15500
WHERE last_name = 'Russell'
Error report:
SQL Error: ORA-20202: Employee cannot earn more than $15,000.
ORA-06512: at "ORA62.RESTRICT_SALARY", line 4
ORA-04088: error during execution of trigger 'ORA62.RESTRICT_SALARY'
```

ORACLE

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

Sie können einen BEFORE-Row Trigger erstellen, um zu verhindern, dass der auslösende Vorgang erfolgreich ist, wenn gegen eine bestimmte Bedingung verstößen wird.

Im ersten Beispiel auf der Folie wird ein Trigger erstellt, um zu gewährleisten, dass nur Mitarbeiter mit der Tätigkeits-ID AD_PRES oder AD_VP ein Gehalt von über 15.000 verdienen. Wenn Sie versuchen, das Gehalt des Mitarbeiters Russell mit der Tätigkeits-ID SA_MAN zu aktualisieren, löst der Trigger die auf der Folie gezeigte Exception aus.

Hinweis: Stellen Sie vor Ausführung des ersten Codebeispiels auf der Folie sicher, dass Sie die Trigger `secure_emp` und `secure_employees` deaktivieren.

Qualifizierer OLD und NEW

- Bei der Auslösung eines Triggers auf Zeilenebene erstellt und füllt die PL/SQL-Runtime Engine zwei Datenstrukturen:
 - **OLD:** Speichert die ursprünglichen Werte des vom Trigger verarbeiteten Records
 - **NEW:** Enthält die neuen Werte
- **NEW und OLD weisen die gleiche Struktur auf wie ein Record, in dem die dem Trigger zugeordnete Tabelle mit %ROWTYPE deklariert wurde.**

Datenvorgänge	Alter Wert	Neuer Wert
INSERT	NULL	Eingefügter Wert
UPDATE	Wert vor Aktualisierung	Wert nach Aktualisierung
DELETE	Wert vor dem Löschen	NULL

ORACLE

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

In einem ROW-Trigger können Sie den Wert einer Spalte vor und nach der Datenänderung referenzieren, indem Sie den Qualifizierer **OLD** und **NEW** als Präfix hinzufügen.

Hinweis

- Die Qualifizierer **OLD** und **NEW** sind nur in **ROW** Triggern verfügbar.
- Stellen Sie diesen Qualifizierern in allen SQL- und PL/SQL-Anweisungen einen Doppelpunkt (:) voran.
- Es wird kein Doppelpunkt (:) vorangestellt, wenn die Qualifizierer in einer einschränkenden WHEN-Bedingung referenziert werden.
- Row Trigger können die Performance verringern, wenn Sie viele Aktualisierungen in großen Tabellen ausführen.

Qualifizierer OLD und NEW – Beispiel

```

CREATE TABLE audit_emp (
    user_name      VARCHAR2(30),
    time_stamp     date,
    id             NUMBER(6),
    old_last_name VARCHAR2(25),
    new_last_name VARCHAR2(25),
    old_title      VARCHAR2(10),
    new_title      VARCHAR2(10),
    old_salary     NUMBER(8,2),
    new_salary     NUMBER(8,2) )
/
CREATE OR REPLACE TRIGGER audit_emp_values
AFTER DELETE OR INSERT OR UPDATE ON employees
FOR EACH ROW
BEGIN
    INSERT INTO audit_emp(user_name, time_stamp, id,
        old_last_name, new_last_name, old_title,
        new_title, old_salary, new_salary)
    VALUES (USER, SYSDATE, :OLD.employee_id,
        :OLD.last_name, :NEW.last_name, :OLD.job_id,
        :NEW.job_id, :OLD.salary, :NEW.salary);
END;

```

ORACLE

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

Im Beispiel auf der Folie wird der Trigger AUDIT_EMP_VALUES für die Tabelle EMPLOYEES erstellt. Der Trigger protokolliert die Aktivitäten eines Benutzers in der Tabelle EMPLOYEES und fügt einer Benutzertabelle (AUDIT_EMP) entsprechende Zeilen hinzu. Der Trigger speichert die Werte verschiedener Spalten vor und nach den Datenänderungen und verwendet dabei die Qualifizierer OLD und NEW mit den jeweiligen Spaltennamen.

Qualifizierer OLD und NEW – Beispiel

```

INSERT INTO employees (employee_id, last_name, job_id,
salary, email, hire_date)
VALUES (999, 'Temp emp', 'SA_REP', 6000, 'TEMPEMP',
TRUNC(SYSDATE))
/
UPDATE employees
  SET salary = 7000, last_name = 'Smith'
 WHERE employee_id = 999
/
SELECT *
FROM audit_emp;

```

Query Result X

All Rows Fetched: 2 in 0.005 seconds

	USER_NAME	TIME_STAMP	ID	OLD_LAST_NAME	NEW_LAST_NAME	OLD_TITLE	NEW_TITLE	OLD_SALARY	NEW_SALARY
1	ORA61	20-NOV-12	(null)	(null)	Temp emp	(null)	SA_REP	(null)	6000
2	ORA61	20-NOV-12	999	Temp emp	Smith	SA_REP	SA_REP	6000	7000

ORACLE

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

Qualifizierer OLD und NEW – Beispiel: Tabelle AUDIT_EMP

Sie erstellen einen Trigger für die Tabelle EMPLOYEES, um Zeilen in eine Benutzertabelle namens AUDIT_EMP einzufügen, mit der die Aktivitäten eines Benutzers in der Tabelle EMPLOYEES protokolliert werden. Der Trigger speichert die Werte verschiedener Spalten vor und nach den Datenänderungen und verwendet dabei die Qualifizierer OLD und NEW mit den jeweiligen Spaltennamen.

Durch Einfügen des Mitarbeiter-Records in die Tabelle EMPLOYEES wird folgendes Ergebnis erzielt:

EMPLOYEE_ID	FIRST_NAME	LAST_NAME	EMAIL	PHONE_NUMBER	HIRE_DATE	JOB_ID	SALARY	COMMISSION_PCT	MANAGER_ID	DEPARTMENT_ID
999 (null)	Smith	TEMPEMP	(null)	04-JUN-09	SA_REP	7000	(null)	(null)	(null)	(null)
300 Rob	Smith	RSMITH	(null)	04-JUN-09	IT_PROG	4500	(null)	(null)	(null)	60
206 William	Gietz	WGIETZ	515.123.8181	07-JUN-94	AC_ACCOUNT	8300	(null)	(null)	205	110

...

Durch Aktualisieren des Gehalts für den Mitarbeiter Smith wird folgendes Ergebnis erzielt:

EMPLOYEE_ID	FIRST_NAME	LAST_NAME	EMAIL	PHONE_NUMBER	HIRE_DATE	JOB_ID	SALARY	COMMISSION_PCT	MANAGER_ID	DEPARTMENT_ID
999 (null)	Smith	TEMPEMP	(null)	04-JUN-09	SA_REP	7000	(null)	(null)	(null)	(null)

...

Row Trigger mit der Klausel WHEN bedingt auslösen

```

CREATE OR REPLACE TRIGGER derive_commission_pct
BEFORE INSERT OR UPDATE OF salary ON employees
FOR EACH ROW
WHEN (NEW.job_id = 'SA_REP')
BEGIN
  IF INSERTING THEN
    :NEW.commission_pct := 0;
  ELSIF :OLD.commission_pct IS NULL THEN
    :NEW.commission_pct := 0;
  ELSE
    :NEW.commission_pct := :OLD.commission_pct+0.05;
  END IF;
END;
/

```

ORACLE

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

Row Trigger einschränken – Beispiel

Optional können Sie in die Definition eines Row Triggers eine Triggereinschränkung aufnehmen, indem Sie in einer WHEN-Klausel einen booleschen SQL-Ausdruck angeben. Wenn Sie eine WHEN-Klausel in den Trigger aufnehmen, wird der Ausdruck in der Klausel für jede vom Trigger betroffene Zeile ausgewertet.

Ergibt die Auswertung eines Ausdrucks für eine Zeile TRUE, wird der Triggerbody für diese Zeile ausgeführt. Ergibt die Auswertung des Ausdrucks für eine Zeile dagegen FALSE oder NOT TRUE (unbekannt, z. B. bei Nullwerten), wird der Triggerbody für diese Zeile nicht ausgeführt. Die Auswertung der Klausel WHEN hat keine Auswirkung auf die Ausführung der auslösenden SQL-Anweisung. Mit anderen Worten, die auslösende Anweisung wird nicht zurückgerollt, wenn die Auswertung eines WHEN-Klauselausdrucks FALSE ergibt.

Hinweis: Eine WHEN-Klausel kann nicht in die Definition eines Statement Triggers aufgenommen werden.

Im Beispiel auf der Folie wird ein Trigger für die Tabelle EMPLOYEES erstellt, um die Provision eines Mitarbeiters zu berechnen, wenn eine Zeile in der Tabelle EMPLOYEES eingefügt oder das Gehalt eines Mitarbeiters geändert wird.

Der Qualifizierer NEW darf in der Klausel WHEN nicht mit einem Doppelpunkt als Präfix versehen werden, da sich die Klausel WHEN außerhalb der PL/SQL-Blöcke befindet.

Triggerausführungsmodell – Zusammenfassung

- 1. Führen Sie alle BEFORE-Statement Trigger aus.**
- 2. Verwenden Sie Schleifen für jede von der SQL-Anweisung betroffene Zeile:**
 - a. Führen Sie alle BEFORE-Row Trigger für diese Zeile aus.**
 - b. Führen Sie die DML-Anweisung aus, und prüfen Sie Integritäts-Constraints für diese Zeile.**
 - c. Führen Sie alle AFTER-Row Trigger für diese Zeile aus.**
- 3. Führen Sie alle AFTER-Statement Trigger aus.**

ORACLE

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

Triggerausführungsmodell

Eine einzelne DML-Anweisung kann bis zu vier Triggertypen auslösen:

- Statement Trigger BEFORE und AFTER
- Row Trigger BEFORE und AFTER

Ein auslösendes Ereignis oder eine Anweisung innerhalb des Triggers kann die Prüfung eines oder mehrerer Integritäts-Constraints zur Folge haben. Sie können die Constraint-Prüfung jedoch verzögern, bis ein COMMIT-Vorgang ausgeführt wird.

Trigger können auch andere Trigger auslösen (kaskadierende Trigger).

Alle Aktionen und Prüfungen, die als Folge einer SQL-Anweisung ausgeführt werden, müssen erfolgreich sein. Wenn eine Exception innerhalb eines Triggers ausgelöst und nicht explizit behandelt wird, werden alle aufgrund der ursprünglichen SQL-Anweisung ausgeführten Aktionen zurückgerollt (einschließlich Aktionen, die durch das Auslösen von Triggern ausgeführt wurden). Auf diese Weise wird sichergestellt, dass Trigger niemals Integritäts-Constraints verletzen.

Wenn ein Trigger ausgelöst wird, können die in der Triggeraktion referenzierten Tabellen durch Transaktionen anderer Benutzer geändert werden. In allen Fällen ist ein lesekonsistentes Bild für die geänderten Werte sichergestellt, in denen der Trigger Lesevorgänge (Abfragen) oder Schreibvorgänge (Aktualisierungen) durchführen muss.

Hinweis: Die Integritätsprüfung kann bis zur Ausführung des Vorgangs COMMIT verzögert werden.

Integritäts-Constraints mit After-Triggern implementieren

```
-- Integrity constraint violation error -2991 raised.
UPDATE employees SET department_id = 999
WHERE employee_id = 170;
```

```
CREATE OR REPLACE TRIGGER employee_dept_fk_trg
AFTER UPDATE OF department_id ON employees
FOR EACH ROW
BEGIN
    INSERT INTO departments VALUES(:new.department_id,
        'Dept ' || :new.department_id, NULL, NULL);
EXCEPTION
    WHEN DUP_VAL_ON_INDEX THEN
        NULL; -- mask exception if department exists
END;
/
```

```
-- Successful after trigger is fired
UPDATE employees SET department_id = 999
WHERE employee_id = 170;
```

1 rows updated

ORACLE

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

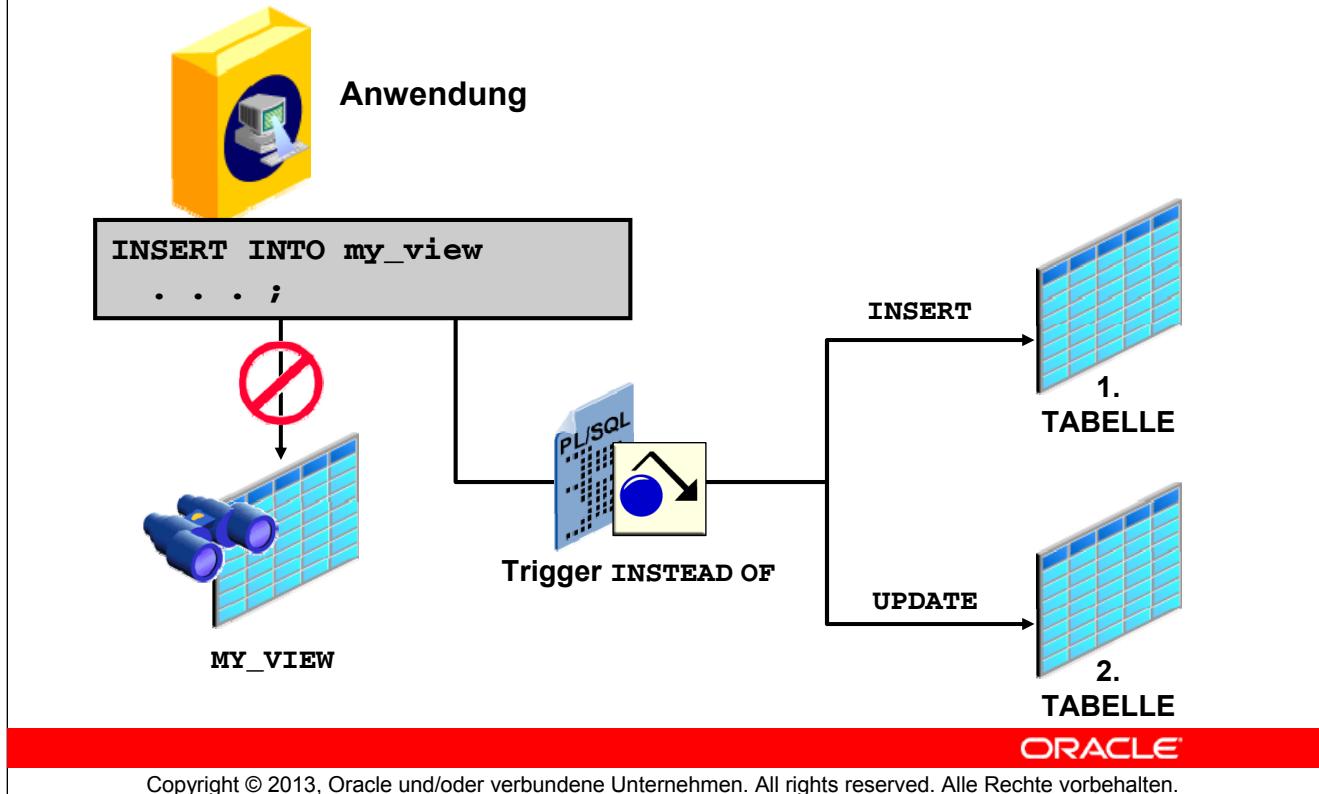
Das Beispiel auf der Folie zeigt eine Situation, in der das Integritäts-Constraint mithilfe eines AFTER-Triggers behandelt werden kann. Die Tabelle EMPLOYEES weist ein FOREIGN KEY Constraint für die Spalte DEPARTMENT_ID der Tabelle DEPARTMENTS auf.

In der ersten SQL-Anweisung wird die DEPARTMENT_ID des Mitarbeiters 170 in 999 geändert. Da eine solche Abteilung in der Tabelle DEPARTMENTS nicht existiert, löst die Anweisung wegen der Verletzung des Integritäts-Constraints die Exception -2291 aus.

Der Trigger EMPLOYEE_DEPT_FK_TRG wird erstellt. Er fügt eine neue Zeile in die Tabelle DEPARTMENTS ein, indem er :NEW.DEPARTMENT_ID für den Wert der DEPARTMENT_ID der neuen Abteilung verwendet. Der Trigger wird ausgelöst, wenn die Anweisung UPDATE die DEPARTMENT_ID von Mitarbeiter 170 in 999 ändert. Die Prüfung des Fremdschlüssel-Constraints verläuft erfolgreich, da der Trigger die Abteilung 999 in die Tabelle DEPARTMENTS eingefügt hat. Aus diesem Grund tritt keine Exception auf, es sei denn, die Abteilung ist bereits vorhanden, wenn der Trigger versucht, die neue Zeile einzufügen. Der Exception Handler fängt die Exception jedoch ab und verbirgt sie, sodass der Vorgang erfolgreich ausgeführt werden kann.

Hinweis: Obwohl das Beispiel auf der Folie aufgrund der beschränkten Daten im Schema HR etwas konstruiert ist, wird doch eines klar: Wenn Sie die Constraint-Prüfung bis zum Commit verzögern, können Sie einen Trigger entwickeln, der diesen Constraint-Fehler erkennt und vor der Commit-Aktion korrigiert.

INSTEAD OF-Trigger



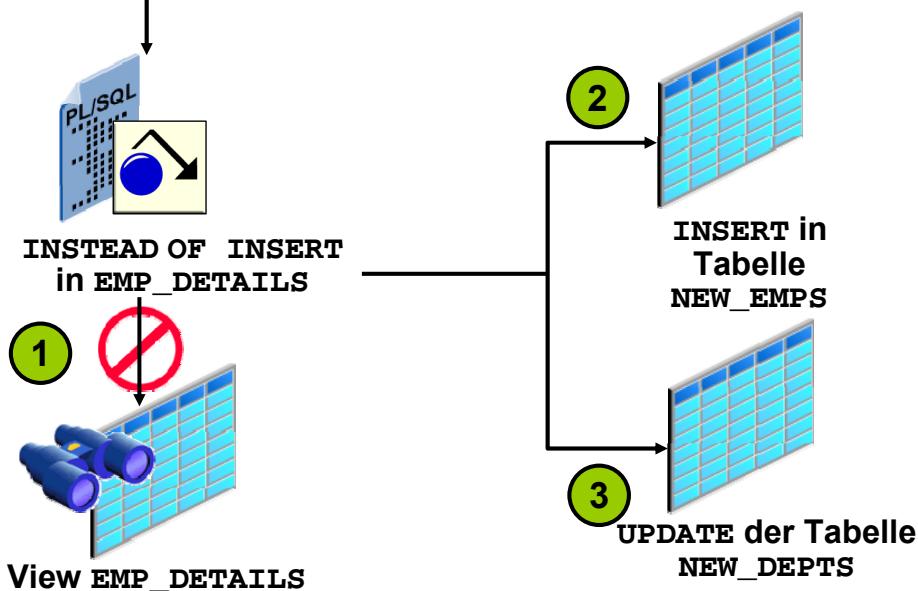
Copyright © 2013, Oracle und/oder verbundene Unternehmen. All rights reserved. Alle Rechte vorbehalten.

Verwenden Sie INSTEAD OF-Triggere, um Daten zu ändern, bei denen eine DML-Anweisung für eine View abgesetzt wird, die implizit nicht aktualisierbar ist. Diese Trigger werden INSTEAD OF-Triggere genannt, da der Oracle-Server im Gegensatz zu anderen Triggertypen nicht die auslösende Anweisung ausführt, sondern den Trigger auslöst. Mit diesen Triggern werden Vorgänge vom Typ INSERT, UPDATE und DELETE direkt für die zugrunde liegenden Tabellen ausgeführt. Sie können Anweisungen vom Typ INSERT, UPDATE und DELETE für eine View erstellen, und der Trigger INSTEAD OF arbeitet unsichtbar im Hintergrund und sorgt für die Ausführung der gewünschten Aktionen. Views können nicht mit normalen DML-Anweisungen geändert werden, wenn die View-Abfrage festgelegte Operatoren, Gruppenfunktionen, Klauseln wie GROUP BY, CONNECT BY und START, den Operator DISTINCT oder Joins enthält. Beispiel: Wenn eine View aus mehreren Tabellen besteht, kann ein INSERT-Vorgang in die View einen INSERT-Vorgang in eine Tabelle und einen UPDATE-Vorgang einer anderen Tabelle mit sich bringen. Daher erstellen Sie einen INSTEAD OF-Trigger, der ausgelöst wird, wenn Sie einen INSERT-Vorgang für die View erstellen. Anstelle des ursprünglichen INSERT-Vorgangs wird der Triggerbody ausgeführt. Dies hat zur Folge, dass Daten in eine Tabelle eingefügt werden und eine andere Tabelle aktualisiert wird.

Hinweis: Wenn eine View implizit aktualisierbar ist und INSTEAD OF-Trigger enthält, haben die Trigger Priorität. INSTEAD OF-Triggere sind Row Trigger. Die Option CHECK für Views wird nicht erzwungen, wenn INSERT- oder UPDATE-Vorgänge für die View mithilfe von INSTEAD OF-Triggern ausgeführt werden. Der INSTEAD OF-Triggerbody muss die Prüfung erzwingen.

Trigger INSTEAD OF erstellen – Beispiel

```
INSERT INTO emp_details
VALUES (9001,'ABBOTT',3000, 10, 'Administration');
```



ORACLE

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

Sie können einen INSTEAD OF-Trigger erstellen, um die Basistabellen zu verwalten, auf denen eine View basiert.

Das Beispiel auf der Folie zeigt einen Mitarbeiter, der in die View `EMP_DETAILS` eingefügt wird, deren Abfrage auf den Tabellen `EMPLOYEES` und `DEPARTMENTS` basiert. Der Trigger `NEW_EMP_DEPT` (INSTEAD OF) wird anstelle des Vorgangs `INSERT` ausgeführt, der den Trigger auslöst. Der Trigger INSTEAD OF setzt dann die entsprechenden Anweisungen `INSERT` und `UPDATE` für die Basistabellen ab, die von der View `EMP_DETAILS` verwendet werden. Aus diesem Grund wird der neue Mitarbeiter-Record nicht in die Tabelle `EMPLOYEES` eingefügt. Stattdessen werden die folgenden Aktionen ausgeführt:

1. Der Trigger `NEW_EMP_DEPT` INSTEAD OF wird ausgelöst.
2. In die Tabelle `NEW_EMPS` wird eine neue Zeile eingefügt.
3. Die Spalte `DEPT_SAL` der Tabelle `NEW_DEPTS` wird aktualisiert. Das Gehalt für den neuen Mitarbeiter wird zum vorhandenen Gesamtgehalt der Abteilung addiert, in die der neue Mitarbeiter aufgenommen wurde.

Hinweis: Bevor Sie das Beispiel auf der Folie ausführen, müssen Sie die auf den folgenden beiden Seiten gezeigten erforderlichen Strukturen erstellen.

Trigger INSTEAD OF zum Ausführen von DML für komplexe Views erstellen

```

CREATE TABLE new_emps AS
  SELECT employee_id, last_name, salary, department_id
    FROM employees;

CREATE TABLE new_depts AS
  SELECT d.department_id, d.department_name,
         sum(e.salary) dept_sal
    FROM employees e, departments d
   WHERE e.department_id = d.department_id;

CREATE VIEW emp_details AS
  SELECT e.employee_id, e.last_name, e.salary,
         e.department_id, d.department_name
    FROM employees e, departments d
   WHERE e.department_id = d.department_id
  GROUP BY d.department_id, d.department_name;

```

ORACLE

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

Im Beispiel auf der Folie werden zwei neue Tabellen erstellt, NEW_EMPS und NEW_DEPTS, die jeweils auf der Tabelle EMPLOYEES bzw. DEPARTMENTS basieren. Außerdem wird eine View namens EMP_DETAILS aus den Tabellen EMPLOYEES und DEPARTMENTS erstellt.

DML-Anweisungen können im Falle von Views mit komplexen Abfragestrukturen nicht immer direkt ausgeführt werden, um die zugrunde liegenden Tabellen zu beeinflussen. Im Beispiel muss ein INSTEAD OF-Trigger namens NEW_EMP_DEPT erstellt werden, der auf der nächsten Seite zu sehen ist. Der Trigger NEW_DEPT_EMP behandelt DML wie folgt:

- Zeilen werden nicht direkt in die View EMP_DETAILS eingefügt, sondern mit den über die Anweisung INSERT bereitgestellten Datenwerten in die Tabellen NEW_EMPS und NEW_DEPTS.
- Werden Zeilen über die View EMP_DETAILS geändert oder gelöscht, wirkt sich dies auf die entsprechenden Zeilen in den Tabellen NEW_EMPS und NEW_DEPTS aus.

Hinweis: INSTEAD OF-Trigger können nur für Views erstellt werden. Die Timing-Optionen BEFORE und AFTER sind ungültig.

```

CREATE OR REPLACE TRIGGER new_emp_dept
INSTEAD OF INSERT OR UPDATE OR DELETE ON emp_details
FOR EACH ROW
BEGIN
  IF INSERTING THEN
    INSERT INTO new_emps
    VALUES (:NEW.employee_id, :NEW.last_name,
            :NEW.salary, :NEW.department_id);
    UPDATE new_depts
    SET dept_sal = dept_sal + :NEW.salary
    WHERE department_id = :NEW.department_id;
  ELSIF DELETING THEN
    DELETE FROM new_emps
    WHERE employee_id = :OLD.employee_id;
    UPDATE new_depts
    SET dept_sal = dept_sal - :OLD.salary
    WHERE department_id = :OLD.department_id;
  ELSIF UPDATING ('salary') THEN
    UPDATE new_emps
    SET salary = :NEW.salary
    WHERE employee_id = :OLD.employee_id;
    UPDATE new_depts
    SET dept_sal = dept_sal +
                  (:NEW.salary - :OLD.salary)
    WHERE department_id = :OLD.department_id;
  ELSIF UPDATING ('department_id') THEN
    UPDATE new_emps
    SET department_id = :NEW.department_id
    WHERE employee_id = :OLD.employee_id;
    UPDATE new_depts
    SET dept_sal = dept_sal - :OLD.salary
    WHERE department_id = :OLD.department_id;
    UPDATE new_depts
    SET dept_sal = dept_sal + :NEW.salary
    WHERE department_id = :NEW.department_id;
  END IF;
END;
/

```

DEPARTMENT_ID	DEPARTMENT_NAME	DEPT_SAL
10	Administration	7400

1 rows selected

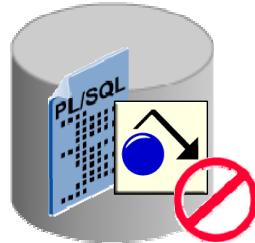
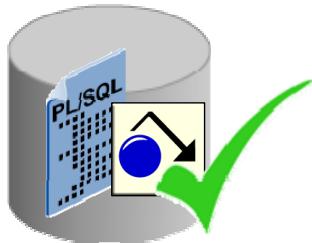
EMPLOYEE_ID	LAST_NAME	SALARY	DEPARTMENT_ID
200	Whalen	4400	10
9001	ABBOTT	3000	10

2 rows selected

Triggerstatus

Die folgenden beiden Triggermodi sind verfügbar:

- **Aktiviert:** Der Trigger führt die entsprechende Aktion aus, wenn eine auslösende Anweisung abgesetzt wird und die Auswertung der Triggereinschränkung (sofern vorhanden) TRUE ergibt (Standard).
- **Deaktiviert:** Der Trigger führt die entsprechende Aktion nicht aus, auch dann nicht, wenn eine auslösende Anweisung abgesetzt wird und die Auswertung der Triggereinschränkung (sofern vorhanden) TRUE ergeben würde.



ORACLE

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

Deaktivierte Trigger erstellen

- Vor Oracle Database 11g waren DML-Anweisungen für eine Tabelle nicht erfolgreich, wenn Sie einen Triggerbody mit einem PL/SQL-Kompilierungsfehler erstellten.
- In Oracle Database 11g können Sie einen deaktivierten Trigger erstellen und diesen erst dann aktivieren, wenn Sie sicher sind, dass die Kompilierung erfolgreich sein wird.

```
CREATE OR REPLACE TRIGGER mytrg
  BEFORE INSERT ON mytable FOR EACH ROW
  DISABLE
BEGIN
  :New.ID := my_seq.Nextval;
  . . .
END;
/
```

ORACLE

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

Vor Oracle Database 11g waren DML-Anweisungen für eine Tabelle nicht erfolgreich, wenn Sie einen Triggerbody mit einem PL/SQL-Kompilierungsfehler erstellten. Die folgende Fehlermeldung wurde angezeigt:

ORA-04098: trigger 'TRG' is invalid and failed re-validation

In Oracle Database 11g können Sie einen deaktivierten Trigger erstellen und diesen erst dann aktivieren, wenn Sie sicher sind, dass die Kompilierung erfolgreich sein wird.

Außerdem können Sie einen Trigger in folgenden Situationen vorübergehend deaktivieren:

- Ein vom Trigger referenziertes Objekt ist nicht verfügbar.
- Sie müssen große Datenmengen laden und möchten dies möglichst schnell ausführen, ohne Trigger auszulösen.
- Sie laden Daten erneut.

Hinweis: Für den Beispielcode auf der Folie wird angenommen, dass bereits eine Sequence namens `my_seq` vorhanden ist.

Trigger mit SQL-Anweisungen

ALTER und DROP verwalten

```
-- Disable or reenable a database trigger:
```

```
ALTER TRIGGER trigger_name DISABLE | ENABLE;
```

```
-- Disable or reenable all triggers for a table:
```

```
ALTER TABLE table_name DISABLE | ENABLE ALL TRIGGERS;
```

```
-- Recompile a trigger for a table:
```

```
ALTER TRIGGER trigger_name COMPILE;
```

```
-- Remove a trigger from the database:
```

```
DROP TRIGGER trigger_name;
```

ORACLE

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

Trigger verwalten

Ein Trigger hat zwei Modi oder Zustände: ENABLED und DISABLED. Wenn ein Trigger neu erstellt wird, ist er standardmäßig aktiviert. Der Oracle-Server prüft die Integritäts-Constraints für aktivierte Trigger und stellt sicher, dass die Trigger sie nicht verletzen können. Außerdem bietet der Oracle-Server lesekonsistente Views für Abfragen und Constraints, verwaltet Abhängigkeiten und stellt einen Two-Phase Commit-Prozess bereit, wenn ein Trigger Remote-Tabellen in einer verteilten Datenbank aktualisiert.

Trigger deaktivieren

Mit dem Befehl ALTER TRIGGER deaktivieren Sie einen Trigger. Mithilfe des Befehls ALTER TABLE können Sie auch alle Trigger für eine Tabelle deaktivieren. Die Deaktivierung von Triggern kann dazu beitragen, die Performance zu steigern oder Datenintegritätsprüfungen beim Laden großer Datenmengen mit Utilities wie SQL*Loader zu vermeiden. Deaktivieren Sie einen Trigger auch dann, wenn er ein Datenbankobjekt referenziert, das derzeit aufgrund einer nicht erfolgreichen Netzwerkverbindung, eines Datenträgerfehlers, einer Offlinedatadatei oder eines Offline-Tablespace nicht verfügbar ist.

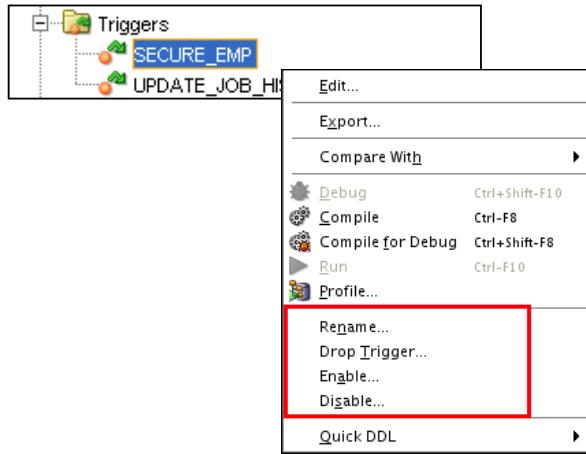
Trigger rekompilieren

Mit dem Befehl ALTER TRIGGER rekompilieren Sie einen ungültigen Trigger explizit.

Trigger entfernen

Nicht länger benötigte Trigger können über eine SQL-Anweisung in SQL Developer oder SQL*Plus entfernt werden. Beim Löschen einer Tabelle werden auch alle damit verbundenen Trigger gelöscht.

Trigger mit SQL Developer verwalten



ORACLE

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

Trigger können im Connections Navigator-Baum im Knoten **Triggers** verwaltet werden. Klicken Sie mit der rechten Maustaste auf einen Triggernamen, und wählen Sie anschließend eine der folgenden Optionen:

- Edit
- Compile
- Compile for Debug
- Rename
- Drop Trigger
- Enable
- Disable

Trigger testen

- Alle triggerauslösenden und nicht triggerauslösenden DML-Vorgänge testen
- Jedes Vorkommen einer WHEN-Klausel testen
- Trigger direkt aus einem grundlegenden DML-Vorgang und indirekt aus einer Prozedur auslösen
- Auswirkung des Triggers auf andere Trigger testen
- Auswirkung anderer Trigger auf den Trigger testen

ORACLE

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

Das Testen von Code kann zeitaufwändig sein. Gehen Sie beim Testen von Triggern wie folgt vor:

- Stellen Sie sicher, dass ein Trigger ordnungsgemäß funktioniert, indem Sie einige Fälle separat testen:
 - Testen Sie zunächst die häufigsten Erfolgsszenarios.
 - Testen Sie die häufigsten Fehlerbedingungen, um zu prüfen, ob sie ordnungsgemäß verwaltet werden.
- Je komplexer der Trigger ist, desto detaillierter wird der Test ausfallen. Beispiel: Wenn Sie einen Row Trigger mit einer WHEN-Klausel angegeben haben, sollten Sie sicherstellen, dass der Trigger ausgelöst wird, wenn die Bedingungen erfüllt sind. Wenn Sie kaskadierende Trigger haben, müssen Sie die Auswirkungen der Trigger untereinander testen, um sicherzustellen, dass Sie die gewünschten Ergebnisse erhalten.
- Führen Sie mithilfe des Packages DBMS_OUTPUT ein Triggerdebugging durch.

Triggerinformationen anzeigen

Sie können die folgenden Triggerinformationen anzeigen:

Data Dictionary View	Beschreibung
USER_OBJECTS	Zeigt Objektinformationen an
USER/ALL/DBA_TRIGGERS	Zeigt Triggerinformationen an
USER_ERRORS	Zeigt PL/SQL-Syntaxfehler für einen Trigger an

ORACLE

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

Die Folie zeigt die Data Dictionary Views, auf die Sie zugreifen können, um Informationen über Trigger abzurufen.

Die View `USER_OBJECTS` enthält den Namen und Status des Triggers sowie das Datum und die Uhrzeit der Triggererstellung.

Die View `USER_ERRORS` enthält Details zu den Kompilierungsfehlern, die während der Kompilierung bei einem Trigger aufgetreten sind. Der Inhalt dieser Views entspricht weitestgehend dem Inhalt von Unterprogrammen.

Die View `USER_TRIGGERS` enthält Details wie den Namen und Typ, das auslösende Ereignis, die Tabelle, für die der Trigger erstellt wird, und den Triggerbody.

Die Anweisung `SELECT Username FROM USER_USERS;` enthält den Namen des Trigger-eigentümers, nicht den Namen des Benutzers, der die Tabelle aktualisiert.

USER_TRIGGERS

DESCRIBE user_triggers

```
DESCRIBE user_triggers
Name          Null Type
-----
TRIGGER_NAME    VARCHAR2(128)
TRIGGER_TYPE     VARCHAR2(16)
TRIGGERING_EVENT VARCHAR2(246)
TABLE_OWNER      VARCHAR2(128)
BASE_OBJECT_TYPE VARCHAR2(18)
TABLE_NAME       VARCHAR2(128)
COLUMN_NAME      VARCHAR2(4000)
REFERENCING_NAMES VARCHAR2(422)
WHEN_CLAUSE      VARCHAR2(4000)
STATUS          VARCHAR2(8)
DESCRIPTION      VARCHAR2(4000)
ACTION_TYPE      VARCHAR2(11)
TRIGGER_BODY      LONG()
CROSSEDITION    VARCHAR2(7)
BEFORE_STATEMENT VARCHAR2(3)
BEFORE_ROW        VARCHAR2(3)
AFTER_ROW         VARCHAR2(3)
AFTER_STATEMENT   VARCHAR2(3)
INSTEAD_OF_ROW    VARCHAR2(3)
FIRE_ONCE         VARCHAR2(3)
APPLY_SERVER_ONLY VARCHAR2(3)
```

```
SELECT trigger_type, trigger_body
FROM user_triggers
WHERE trigger_name = 'SECURE_EMP';
```

ORACLE

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

Ist die Quelldatei nicht verfügbar, können Sie sie über das SQL Worksheet in SQL Developer oder SQL*Plus aus USER_TRIGGERS regenerieren. Außerdem können Sie die Views ALL_TRIGGERS und DBA_TRIGGERS prüfen, die beide eine Zusatzspalte namens OWNER für den Eigentümer des Objekts enthalten. Das zweite Beispiel auf der Folie führt zu folgendem Ergebnis:

TRIGGER_TYPE	TRIGGER_BODY
BEFORE STATEMENT	BEGIN IF (TO_CHAR(SYSDATE, 'DY') IN ('SAT', 'SUN')) OR (TO_CHAR(SYSDATE, 'HH24'))

Quiz

Ein auslösendes Ereignis kann aus einem oder mehreren der folgenden Vorgänge bestehen:

- a. Anweisung vom Typ INSERT, UPDATE oder DELETE für eine bestimmte Tabelle (oder in einigen Fällen auch eine View)**
- b. Anweisung vom Typ CREATE, ALTER oder DROP für ein beliebiges Schemaobjekt**
- c. Hoch- oder Herunterfahren einer Datenbankinstanz**
- d. Bestimmte oder beliebige Fehlermeldung**
- e. An- oder Abmeldung eines Benutzers**

ORACLE

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

Richtige Antworten: a, b, c, d, e

Zusammenfassung

In dieser Lektion haben Sie Folgendes gelernt:

- **Datenbanktrigger erstellen, die von DML-Vorgängen aufgerufen werden**
- **Statement Trigger und Row Trigger erstellen**
- **Regeln zum Auslösen von Datenbanktriggern verwenden**
- **Datenbanktrigger aktivieren, deaktivieren und verwalten**
- **Strategie zum Testen von Triggern entwickeln**
- **Datenbanktrigger entfernen**

ORACLE

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

In dieser Lektion wurde die Erstellung von Datenbanktriggern behandelt, die vor, nach oder anstelle eines angegebenen DML-Vorgangs ausgeführt werden. Trigger sind mit Datenbanktabellen oder Views assoziiert. Das Timing BEFORE und AFTER gilt für DML-Vorgänge in Tabellen. Der Trigger INSTEAD OF wird verwendet, um DML-Vorgänge für eine View durch die entsprechenden DML-Anweisungen für andere Tabellen in der Datenbank zu ersetzen.

Trigger sind standardmäßig aktiviert, können jedoch deaktiviert werden, um ihre Ausführung bis zur erneuten Aktivierung zu unterdrücken. Wenn sich Geschäftsregeln ändern, können Trigger nach Bedarf entfernt oder geändert werden.

Übungen zu Lektion 9 – Überblick: Statement Trigger und Row Trigger erstellen

Diese Übungen behandeln folgende Themen:

- **Row Trigger erstellen**
- **Statement Trigger erstellen**
- **Prozeduren aus einem Trigger aufrufen**

ORACLE

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

Übungen zu Lektion 9 – Überblick

In diesen Übungen erstellen Sie Statement Trigger und Row Trigger. Außerdem erstellen Sie Prozeduren, die aus den Triggern aufgerufen werden.

Komplexe, DDL- und Datenbankereignistrigger erstellen

ORACLE

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

Ziele

Nach Ablauf dieser Lektion haben Sie folgende Ziele erreicht:

- **Komplexe Trigger beschreiben**
- **Sich verändernde Tabellen beschreiben**
- **Trigger für DDL-Anweisungen erstellen**
- **Trigger für Systemereignisse erstellen**
- **Informationen zu Triggern anzeigen**

ORACLE

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

In dieser Lektion wird beschrieben, wie Sie Datenbanktrigger erstellen und verwenden.

Was ist ein komplexer Trigger?

Ein einzelner tabellenbezogener Trigger, mit dem Sie Aktionen für die vier folgenden Ausführungszeitpunkte vorgeben können:

- **Vor der auslösenden Anweisung**
- **Vor jeder Zeile, die von der auslösenden Anweisung betroffen ist**
- **Nach jeder Zeile, die von der auslösenden Anweisung betroffen ist**
- **Nach der auslösenden Anweisung**

ORACLE

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

In Oracle Database 11g können Sie erstmals komplexe Trigger verwenden. Ein komplexer Trigger ist ein einzelner tabellenbezogener Trigger, mit dem Sie Aktionen für die vier folgenden Ausführungszeitpunkte vorgeben können:

- Vor der auslösenden Anweisung
- Vor jeder Zeile, die von der auslösenden Anweisung betroffen ist
- Nach jeder Zeile, die von der auslösenden Anweisung betroffen ist
- Nach der auslösenden Anweisung

Hinweis: Weitere Informationen über Trigger finden Sie im Dokument *Oracle Database PL/SQL Language Reference*.

Mit komplexen Triggern arbeiten

- **Der Body von komplexen Triggern unterstützt einen globalen PL/SQL-Status, auf den der für die jeweiligen Ausführungszeitpunkte relevante Code zugreifen kann.**
- **Der globale Status für komplexe Trigger:**
 - wird beim Start der triggerauslösenden Anweisung erstellt
 - wird nach Abschluss der triggerauslösenden Anweisung gelöscht
- **Ein komplexer Trigger verfügt über einen deklarativen Bereich und je einen Bereich für die einzelnen Ausführungszeitpunkte.**

ORACLE

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

Der Body von komplexen Triggern unterstützt einen globalen PL/SQL-Status, auf den der für die jeweilige Ausführungszeitpunkte relevante Code zugreifen kann. Der globale Status wird automatisch gelöscht, sobald die auslösende Anweisung abgeschlossen ist – selbst dann, wenn die auslösende Anweisung einen Fehler verursacht. Um Fehler in sich verändernden Tabellen zu vermeiden, können Sie zulassen, dass Zeilen, die für eine zweite Tabelle (z. B. eine Historien- oder Auditabelle) bestimmt sind, gesammelt und anschließend global eingefügt werden.

Vor Oracle Database 11g Release 1 (11.1) mussten Sie den globalen Status mit einem Hilfs-package modellieren. Dieser Ansatz war programmieraufwändig und anfällig für Speicherlecks, wenn die auslösende Anweisung einen Fehler verursachte und die Anweisung AFTER nicht ausgelöst wurde. Durch komplexe Trigger wird PL/SQL benutzerfreundlicher und Laufzeit-performance und Skalierbarkeit werden verbessert.

Komplexe Trigger – Vorteile

Sie können komplexe Trigger für folgende Zwecke einsetzen:

- Ansatz programmieren, mit dem die für die einzelnen Ausführungszeitpunkte implementierten Aktionen gemeinsame Daten verwenden
- Zeilen, die für eine zweite Tabelle bestimmt sind, sammeln und in Abständen global einfügen
- Fehler in sich verändernden Tabellen (ORA-04091) vermeiden, indem Zeilen für eine zweite Tabelle gesammelt und anschließend global eingefügt werden

ORACLE

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

Ausführungszeitbereiche von komplexen tabellenbezogenen Triggern

Für Tabellen definierte komplexe Trigger haben einen oder mehrere der folgenden Ausführungszeitbereiche.

Zeitbereiche müssen in der folgenden Reihenfolge angeordnet sein.

Zeitpunkt	Bereich komplexer Trigger
Vor Ausführung der triggerauslösenden Anweisung	Anweisung BEFORE
Nach Ausführung der triggerauslösenden Anweisung	Anweisung AFTER
Vor jeder Zeile, die von der triggerauslösenden Anweisung betroffen ist	BEFORE EACH ROW
Nach jeder Zeile, die von der triggerauslösenden Anweisung betroffen ist	AFTER EACH ROW

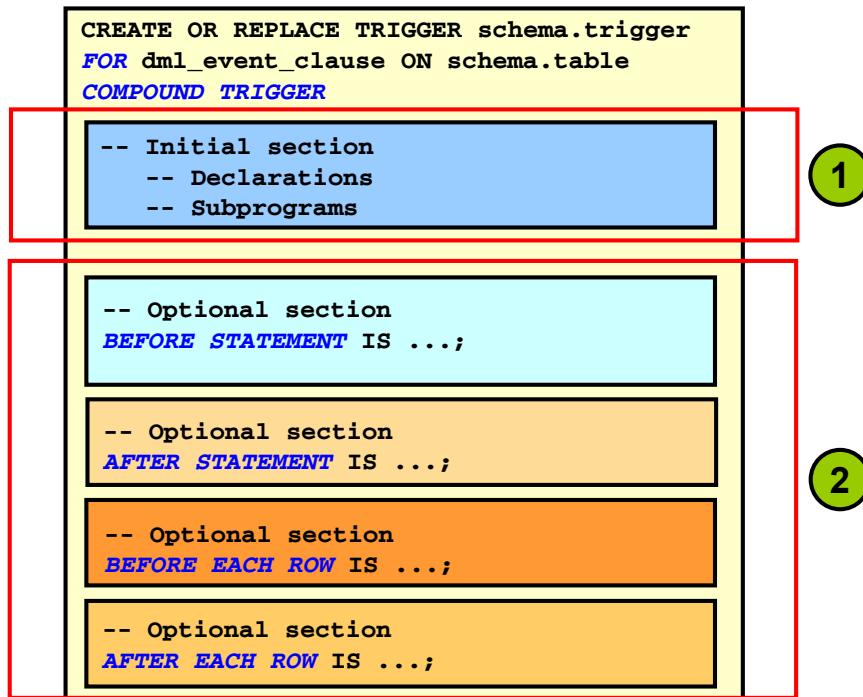
ORACLE

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

Hinweis

Ausführungszeitbereiche müssen in der auf der Folie gezeigten Reihenfolge angeordnet sein.
Fehlt ein Bereich, wird an der entsprechenden Stelle keine Aktion ausgeführt.

Komplexe Trigger für Tabellen – Struktur



ORACLE

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

Ein komplexer Trigger hat zwei Hauptbereiche:

- Einen Anfangsbereich, in dem Variablen und Unterprogramme deklariert werden. Der Code in diesem Bereich wird vor dem Code im optionalen Bereich ausgeführt.
- Einen optionalen Bereich, in dem der Code für mögliche Triggerpunkte definiert ist. Die triggerauslösenden Punkte sind unterschiedlich, je nachdem, ob Sie komplexe Trigger für Tabellen oder Views definieren (siehe Folien auf dieser und der nächsten Seite). Der Code für die triggerauslösenden Punkte muss in der oben gezeigten Reihenfolge angeordnet sein.

Hinweis: Weitere Informationen über komplexe Trigger finden Sie im Dokument *Oracle Database PL/SQL Language Reference*.

Komplexe Trigger für Views – Struktur

```
CREATE OR REPLACE TRIGGER
schema.trigger
FOR dml_event_clause ON schema.view
COMPOUND TRIGGER
    -- Initial section
    -- Declarations
    -- Subprograms
    -- Optional section (exclusive)
    INSTEAD OF EACH ROW IS
    ....;
```

ORACLE

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

Für Views ist ausschließlich eine Klausel vom Typ INSTEAD OF EACH ROW zulässig.

Komplexe Trigger – Einschränkungen

- Komplexe Trigger müssen DML-Trigger sein und für Tabellen oder Views definiert werden.
- Der Body von komplexen Triggern muss ein in PL/SQL erstellter komplexer Triggerblock sein.
- Der Body von komplexen Triggern enthält keinen Initialisierungsblock und daher auch keinen Exception-Bereich.
- Exceptions müssen in dem Bereich behandelt werden, in dem sie auftreten. Die Kontrolle kann nicht an einen anderen Bereich übergeben werden.
- :OLD und :NEW dürfen im deklarativen Bereich und in den Bereichen BEFORE STATEMENT und AFTER STATEMENT nicht vorkommen.
- Der Wert von :NEW kann nur im Bereich BEFORE EACH ROW geändert werden.
- Die Auslösereihenfolge von komplexen Triggern ist nur mit der Klausel **FOLLOWs** garantiert.

ORACLE

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

Für komplexe Trigger gelten unter anderem folgende Einschränkungen:

- Der Body von komplexen Triggern muss ein in PL/SQL erstellter komplexer Triggerblock sein.
- Ein komplexer Trigger muss ein DML-Trigger sein.
- Komplexe Trigger müssen für Tabellen oder Views definiert werden.
- Der Body von komplexen Triggern enthält keinen Initialisierungsblock und daher auch keinen Exception-Bereich. Dies ist unproblematisch, da der Bereich BEFORE STATEMENT vor Ausführung der anderen Zeitbereiche immer genau einmal ausgeführt wird.
- Exceptions müssen in dem Bereich behandelt werden, in dem sie auftreten. Die Kontrolle kann nicht an einen anderen Bereich übergeben werden.
- :OLD, :NEW und :PARENT dürfen im deklarativen Bereich und in den Bereichen BEFORE STATEMENT oder AFTER STATEMENT nicht vorkommen.
- Die Auslösereihenfolge von komplexen Triggern ist nur mit der Klausel **FOLLOWs** garantiert.

Triggereinschränkungen für sich verändernde Tabellen

- **Eine sich verändernde Tabelle:**
 - wird von **UPDATE**-, **DELETE**- oder **INSERT**-Anweisungen bearbeitet oder
 - durch **DELETE CASCADE**-Constraints aktualisiert
- **Sich verändernde Tabellen können nicht in der Session abgefragt oder bearbeitet werden, in der die triggerauslösende Anweisung abgesetzt wird.**
- **Diese Einschränkung verhindert inkonsistente Daten bei der Triggerverarbeitung.**
- **Die Einschränkung gilt für alle Trigger, die die Klausel FOR EACH ROW verwenden.**
- **Views, die über den Trigger INSTEAD OF modifiziert werden, zählen nicht zu den sich verändernden Views.**

ORACLE

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

Regeln für Trigger

Für das Lesen und Schreiben von Daten mithilfe von Triggern gelten bestimmte Regeln. Diese Einschränkungen gelten jedoch nur für zeilenorientierte Trigger, es sei denn, ein Statement-Trigger wird als Ergebnis von ON DELETE CASCADE ausgelöst.

Sich verändernde Tabelle

Eine sich verändernde Tabelle (Mutating Table) ist eine Tabelle, die durch eine **UPDATE**-, **DELETE**- oder **INSERT**-Anweisung geändert wird oder infolge einer deklarativen referentiellen Integritätsaktion **DELETE CASCADE** aktualisiert werden muss. Im Fall von **STATEMENT**-Triggern sind Mutating Tables nicht relevant.

Ein Mutating Table-Fehler (**ORA-4091**) tritt auf, wenn ein Trigger auf Zeilenebene versucht, eine Tabelle zu bearbeiten oder zu prüfen, die bereits über eine DML-Anweisung bearbeitet wird.

Die ausgelöste Tabelle selbst und auch alle anderen Tabellen, die diese Tabelle mit dem **FOREIGN KEY**-Constraint referenzieren, sind Mutating Tables. Mit dieser Einschränkung werden inkonsistente Datenmengen bei der Verarbeitung von Row Triggern verhindert.

Sich verändernde Tabellen – Beispiel

```
CREATE OR REPLACE TRIGGER check_salary
BEFORE INSERT OR UPDATE OF salary, job_id
ON employees
FOR EACH ROW
WHEN (NEW.job_id <> 'AD_PRES')
DECLARE
    v_minsalary employees.salary%TYPE;
    v_maxsalary employees.salary%TYPE;
BEGIN
    SELECT MIN(salary), MAX(salary)
    INTO v_minsalary, v_maxsalary
    FROM employees
    WHERE job_id = :NEW.job_id;
    IF :NEW.salary < v_minsalary OR :NEW.salary > v_maxsalary THEN
        RAISE_APPLICATION_ERROR(-20505,'Out of range');
    END IF;
END;
/
```

ORACLE

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

Im Beispiel auf der Folie soll durch den Trigger CHECK_SALARY sichergestellt werden, dass das Gehalt neuer Mitarbeiter in der Tabelle EMPLOYEES und das Gehalt vorhandener Mitarbeiter nach einer Gehalts- oder Tätigkeits-ID-Änderung innerhalb der für die betreffende Tätigkeit festgelegten Gehaltsspanne liegt.

Nach der Aktualisierung eines Mitarbeiter-Records, wird für jede aktualisierte Zeile der Trigger CHECK_SALARY ausgelöst. Der Triggercode führt eine Abfrage in der Tabelle durch, die gerade aktualisiert wird. Daher wird die Tabelle EMPLOYEES als sich verändernde Tabelle oder Mutating Table bezeichnet.

Sich verändernde Tabellen – Beispiel

```
UPDATE employees
SET salary = 3400
WHERE last_name = 'Stiles';
```

The screenshot shows a 'Script Output' window from Oracle SQL Developer. It displays the following text:

```
Task completed in 0.016 seconds
Error starting at line 1 in command:
UPDATE employees
SET salary = 3400
WHERE last_name = 'Stiles'
Error report:
SQL Error: ORA-04091: table ORA$1.EMPLOYEES is mutating, trigger/function may not see it
ORA-06512: at "ORA$1.CHECK_SALARY", line 5
ORA-04088: error during execution of trigger 'ORA$1.CHECK_SALARY'
ORA-04091. 00000 -  "table %s.%s is mutating, trigger/function may not see it"
*Cause: A trigger (or a user defined plsql function that is referenced in
this statement) attempted to look at (or modify) a table that was
in the middle of being modified by the statement which fired it.
*Action: Rewrite the trigger (or function) so it does not read that table.
```

ORACLE

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

Im Beispiel auf der Folie versucht der Triggercode, Daten in einer sich verändernden Tabelle zu lesen oder zu wählen.

Wenn Sie das Gehalt auf einen Betrag zwischen dem vorhandenen Mindestwert und dem vorhandenen Höchstwert einschränken, wird ein Laufzeitfehler ausgegeben. Die Tabelle EMPLOYEES verändert sich (sie mutiert). Daher kann der Trigger keine Daten aus dieser Tabelle lesen.

Auch Funktionen können Mutating Table-Fehler verursachen, wenn sie in einer DML-Anweisung aufgerufen werden.

Mögliche Lösungen

Sie können das Problem sich verändernder Tabellen beispielsweise wie folgt lösen:

- Verwenden Sie einen komplexen Trigger (wie oben erläutert).
- Speichern Sie die Summandaten (die Mindest- und Höchstgehälter) in einer anderen Summentabelle, die durch andere DML-Trigger auf dem aktuellen Stand gehalten wird.
- Speichern Sie die Summandaten in einem PL/SQL-Package, und greifen Sie über das Package auf die Daten zu. Dies ist mit einem BEFORE-Statement Trigger möglich.

Je nach Art des Problems kann die Komplexität einer Lösung zunehmen. Erwägen Sie in diesem Fall, die Regeln in der Anwendung oder der Middle Tier zu implementieren, und vermeiden Sie die Ausführung übermäßig komplexer Geschäftsregeln mit Datenbanktriggern. Eine INSERT-Anweisung generiert im Codebeispiel auf der Folie kein Beispiel für eine sich verändernde Tabelle.

Mutating Table-Fehler mit komplexen Triggern beheben

```

CREATE OR REPLACE TRIGGER check_salary
FOR INSERT OR UPDATE OF salary, job_id
ON employees
WHEN (NEW.job_id <> 'AD_PRES')
COMPOUND TRIGGER

    TYPE salaries_t          IS TABLE OF employees.salary%TYPE;
    min_salaries              salaries_t;
    max_salaries              salaries_t;

    TYPE department_ids_t     IS TABLE OF employees.department_id%TYPE;
    department_ids             department_ids_t;

    TYPE department_salaries_t IS TABLE OF employees.salary%TYPE
                                INDEX BY VARCHAR2(80);
    department_min_salaries   department_salaries_t;
    department_max_salaries   department_salaries_t;

-- example continues on next slide

```

ORACLE

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

Der komplexe Trigger CHECK_SALARY behebt den Fehler in der sich verändernden Tabelle aus dem vorherigen Beispiel. Dazu werden die Werte in PL/SQL-Collections gespeichert und anschließend im Bereich "Before Statement" des komplexen Triggers global eingefügt/aktualisiert. Dieser Code funktioniert ab Oracle Database 11g. Im Beispiel auf der Folie werden PL/SQL-Collections verwendet. Die verwendeten Elementtypen basieren auf den Spalten SALARY und DEPARTMENT_ID aus der Tabelle EMPLOYEES. Um Collections zu erstellen, definieren Sie einen Collection-Typ und deklarieren anschließend Variablen von diesem Typ. Collections werden instanziert, wenn Sie einen Block oder ein Unterprogramm starten, und werden gelöscht, wenn Sie den Block oder das Unterprogramm beenden. `min_salaries` enthält das Mindestgehalt, `max_salaries` das Höchstgehalt der jeweiligen Abteilungen.

`department_ids` enthält die Abteilungs-IDs. Ist dem Mitarbeiter mit dem Mindest- oder Höchstgehalt keine Abteilung zugeordnet, verwenden Sie die NVL-Funktion, um statt `NULL` den Wert -1 für die Abteilungsnummer zu speichern. Anschließend erfassen Sie Mindest- und Höchstgehalt sowie Abteilungsnummer mit einer globalen Einfügeaktion, gruppiert nach Abteilungsnummer für `min_salaries`, `max_salaries` bzw. `department_ids`. Die Anweisung `SELECT` gibt 13 Zeilen zurück. Die Werte für Abteilungsnummern werden als Index für die Tabellen `department_min_salaries` und `department_max_salaries` verwendet. Der Index für diese beiden Tabellen (`VARCHAR2`) entspricht somit den tatsächlichen Abteilungsnummern (`department_ids`).

Mutating Table-Fehler mit komplexen Triggern beheben

```

. . .
BEFORE STATEMENT IS
BEGIN
  SELECT MIN(salary), MAX(salary), NVL(department_id, -1)
  BULK COLLECT INTO min_Salaries, max_salaries, department_ids
  FROM employees
  GROUP BY department_id;
  FOR j IN 1..department_ids.COUNT() LOOP
    department_min_salaries(department_ids(j)) := min_salaries(j);
    department_max_salaries(department_ids(j)) := max_salaries(j);
  END LOOP;
END BEFORE STATEMENT;

AFTER EACH ROW IS
BEGIN
  IF :NEW.salary < department_min_salaries(:NEW.department_id)
  OR :NEW.salary > department_max_salaries(:NEW.department_id) THEN
    RAISE_APPLICATION_ERROR(-20505, 'New Salary is out of acceptable
                                range');
  END IF;
END AFTER EACH ROW;
END check_salary;

```

ORACLE

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

Nach jeder hinzugefügten Zeile wird eine Fehlermeldung angezeigt, wenn das neue Gehalt unter dem Mindestgehalt bzw. über dem Höchstgehalt für diese Abteilung liegt.

Um den neu erstellten komplexen Trigger zu testen, setzen Sie die folgende Anweisung ab:

```

UPDATE employees
SET salary = 3400
WHERE last_name = 'Stiles';

```

1 rows updated

Um sicherzustellen, dass das Gehalt für den Mitarbeiter Stiles aktualisiert wurde, setzen Sie in SQL Developer die folgende Abfrage mit der Taste F9 ab:

```

SELECT employee_id, first_name, last_name, job_id, department_id,
       salary
  FROM employees
 WHERE last_name = 'Stiles';

```

	EMPLOYEE_ID	FIRST_NAME	LAST_NAME	JOB_ID	DEPARTMENT_ID	SALARY
1	138	Stephen	Stiles	ST_CLERK	50	3400

Trigger für DDL-Anweisungen erstellen

```
CREATE [OR REPLACE] TRIGGER trigger_name
BEFORE | AFTER -- Timing
[ddl_event1 [OR ddl_event2 OR ...]]
ON {DATABASE | SCHEMA}
trigger_body
```

DDL-Ereignis – Beispiel	Ausführungszeitpunkt
CREATE	Ein Datenbankobjekt wird mit dem Befehl CREATE erstellt.
ALTER	Ein Datenbankobjekt wird mit dem Befehl ALTER geändert.
DROP	Ein Datenbankobjekt wird mit dem Befehl DROP gelöscht.

ORACLE

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

Sie können einen oder mehrere Typen von DDL-Anweisungen angeben, die den Trigger auslösen. Sofern nicht anders angegeben, lassen sich für diese Ereignisse DATABASE- oder SCHEMA-Trigger erstellen. Außerdem können Sie für das Triggertimeing BEFORE- und AFTER-Werte angeben. Die Oracle-Datenbank löst den Trigger in der vorhandenen Benutzertransaktion aus.

DDL-Vorgänge, die über eine PL/SQL-Prozedur ausgeführt werden, können nicht als auslösendes Ereignis angegeben werden.

Der Triggerbody in der Syntax auf der Folie stellt einen vollständigen PL/SQL-Block dar.

DDL-Triggers werden nur ausgelöst, wenn die folgenden Objekte erstellt werden: Cluster, Funktionen, Indizes, Packages, Prozeduren, Rollen, Sequences, Synonyme, Tabellen, Tablespaces, Trigger, Typen, Views oder Benutzer.

Datenbankereignistrigger erstellen

- **Auslösende Benutzerereignisse:**
 - CREATE, ALTER oder DROP
 - An- oder Abmelden
- **Auslösende Datenbank- oder Systemereignisse:**
 - Datenbank herunter- oder hochfahren
 - Spezifischer (oder beliebiger) ausgelöster Fehler

ORACLE

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

Datenbanktrigger erstellen

Bevor Sie den Triggerbody codieren, legen Sie die Triggerkomponenten fest.

Trigger für Systemereignisse können auf Datenbank- oder Schemaebene definiert werden. So wird beispielsweise ein Trigger zum Herunterfahren einer Datenbank auf der Datenbankebene definiert. Die Trigger für DDL-Anweisungen oder eine Benutzeranmeldung bzw. -abmeldung können ebenfalls auf Datenbank- oder Schemaebene definiert werden. Die Trigger für DML-Anweisungen werden in einer spezifischen Tabelle oder View definiert.

Ein auf Datenbankebene definierter Trigger wird für alle Benutzer ausgelöst. Ein auf Schema- oder Tabellenebene definierter Trigger hingegen wird nur dann ausgelöst, wenn das auslösende Ereignis dieses Schema oder diese Tabelle beinhaltet.

Folgende Ereignisse können einen Trigger auslösen:

- Datendefinitionsanweisung für ein Objekt in der Datenbank oder im Schema
- An- oder Abmelden eines spezifischen (oder beliebigen) Benutzers
- Hoch- oder Herunterfahren einer Datenbank
- Beliebige Fehler

Trigger für Systemereignisse erstellen

```
CREATE [OR REPLACE] TRIGGER trigger_name
BEFORE | AFTER -- timing
[database_event1 [OR database_event2 OR ...]]
ON {DATABASE | SCHEMA}
trigger_body
```

Datenbankereignis	Ausführungszeitpunkt
AFTER SERVERERROR	Ein Oracle-Fehler wird ausgelöst.
AFTER LOGON	Ein Benutzer meldet sich bei der Datenbank an.
BEFORE LOGOFF	Ein Benutzer meldet sich bei der Datenbank ab.
AFTER STARTUP	Die Datenbank wird geöffnet.
BEFORE SHUTDOWN	Die Datenbank wird regulär heruntergefahren.

ORACLE

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

Trigger vom Typ DATABASE oder SCHEMA können für die in der Tabelle auf der Folie genannten Ereignisse erstellt werden. Die Ereignisse SHUTDOWN und STARTUP sind jedoch nur für Trigger vom Typ DATABASE gültig.

Trigger LOGON und LOGOFF – Beispiel

```
-- Create the log_trig_table shown in the notes page
-- first

CREATE OR REPLACE TRIGGER logon_trig
AFTER LOGON ON SCHEMA
BEGIN
    INSERT INTO log_trig_table(user_id,log_date,action)
    VALUES (USER, SYSDATE, 'Logging on');
END;
/
```

```
CREATE OR REPLACE TRIGGER logoff_trig
BEFORE LOGOFF ON SCHEMA
BEGIN
    INSERT INTO log_trig_table(user_id,log_date,action)
    VALUES (USER, SYSDATE, 'Logging off');
END;
/
```

ORACLE

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

Sie können diese Trigger erstellen, um zu überwachen, wie oft Sie sich an- bzw. abmelden. Sie können aber auch einen Bericht erstellen, der festhält, wie lange Sie angemeldet sind. Wenn Sie ON SCHEMA angeben, wird der Trigger nur für den spezifischen Benutzer ausgelöst. Bei Angabe von ON DATABASE wird der Trigger dagegen für alle Benutzer ausgelöst.

Die Definition für log_trig_table in den Beispielen auf der Folie lautet:

```
CREATE TABLE log_trig_table(
    user_id  VARCHAR2(30),
    log_date DATE,
    action   VARCHAR2(40))
/
```

CALL-Anweisungen in Triggern

```
CREATE [OR REPLACE] TRIGGER trigger_name
timing
event1 [OR event2 OR event3]
ON table_name
[REFERENCING OLD AS old | NEW AS new]
[FOR EACH ROW]
[WHEN condition]
CALL procedure_name
/
```

```
CREATE OR REPLACE PROCEDURE log_execution IS
BEGIN
  DBMS_OUTPUT.PUT_LINE('log_execution: Employee Inserted');
END;
/
CREATE OR REPLACE TRIGGER log_employee
BEFORE INSERT ON EMPLOYEES
CALL log_execution -- no semicolon needed
/
```

ORACLE

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

Mit einer CALL-Anweisung können Sie eine Stored Procedure aufrufen und müssen den PL/SQL-Body nicht im Trigger selbst codieren. Die Prozedur kann in PL/SQL, C oder Java implementiert werden.

Der Aufruf kann die Triggerattribute :NEW und :OLD als Parameter referenzieren. Beispiel:

```
CREATE OR REPLACE TRIGGER salary_check
  BEFORE UPDATE OF salary, job_id ON employees
  FOR EACH ROW
  WHEN (NEW.job_id <> 'AD_PRES')
  CALL sal_status(:NEW.job_id, :NEW.salary)
```

Hinweis: Am Ende der Anweisung CALL steht kein Semikolon.

Im vorherigen Beispiel ruft der Trigger die Prozedur sal_status auf. Diese Prozedur vergleicht das neue Gehalt mit der Gehaltsspanne für die neue Tätigkeits-ID aus der Tabelle JOBS.

Datenbankereignistrigger – Vorteile

- **Verbesserte Datensicherheit:**
 - Erweiterte und komplexe Sicherheitsprüfungen
 - Erweitertes und komplexes Auditing
- **Verbesserte Datenintegrität:**
 - Durchsetzung dynamischer Datenintegritäts-Constraints
 - Durchsetzung komplexer referenzieller Integritäts-Constraints
 - Gewährleistung der impliziten gemeinsamen Ausführung zusammengehöriger Vorgänge

ORACLE

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

Sie können Datenbanktrigger für folgende Zwecke einsetzen:

- Als Alternative zu den vom Oracle-Server bereitgestellten Features
- Wenn Ihre Anforderungen komplexer oder einfacher sind als die vom Oracle-Server sichergestellten Anforderungen
- Wenn Ihre Anforderungen vom Oracle-Server nicht sichergestellt werden

Erforderliche Systemberechtigungen zur Triggerverwaltung

Folgende Systemberechtigungen sind für das Verwalten von Triggern erforderlich:

- Die Berechtigungen zum Erstellen, Ändern und Löschen von Triggern in beliebigen Schemas:
 - GRANT CREATE TRIGGER TO ora61
 - GRANT ALTER ANY TRIGGER TO ora61
 - GRANT DROP ANY TRIGGER TO ora61
- Die Berechtigung zum Erstellen eines Triggers für die Datenbank:
 - GRANT ADMINISTER DATABASE TRIGGER TO ora61
- Die Berechtigung EXECUTE (sofern der Trigger Objekte referenziert, die nicht in Ihrem Schema enthalten sind)



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

Für das Erstellen eines Triggers in Ihrem Schema benötigen Sie die Systemberechtigung CREATE TRIGGER. Außerdem müssen Sie Eigentümer der in der auslösenden Anweisung angegebenen Tabelle sein, über die Berechtigung ALTER für die Tabelle in der auslösenden Anweisung verfügen oder die Systemberechtigung ALTER ANY TABLE besitzen. Zum Ändern und Löschen Ihrer Trigger sind keine weiteren Berechtigungen erforderlich.

Wenn das Schlüsselwort ANY verwendet wird, können Sie sowohl Ihre eigenen Trigger als auch die in einem anderen Schema enthaltenen Trigger erstellen, ändern oder löschen. Außerdem können Sie die Trigger einer beliebigen Benutzertabelle zuordnen.

Sie benötigen keine Berechtigungen, um einen Trigger in Ihrem Schema aufzurufen. Trigger werden von den von Ihnen abgesetzten DML-Anweisungen aufgerufen. Wenn Ihr Trigger jedoch auf andere, nicht in Ihrem Schema enthaltene Objekte verweist, muss der Benutzer, der den Trigger erstellt, über die Berechtigung EXECUTE für die referenzierten Prozeduren, Funktionen oder Packages verfügen. Rollen sind nicht ausreichend.

Um einen DATABASE-Trigger zu erstellen, benötigen Sie die Berechtigung ADMINISTER DATABASE TRIGGER. Wenn diese Berechtigung zu einem späteren Zeitpunkt entzogen wird, können Sie den Trigger zwar löschen, aber nicht ändern.

Hinweis: Ähnlich wie bei Stored Procedures verwenden Anweisungen im Triggerbody die Berechtigungen des Triggereigentümers, nicht die Berechtigungen des Benutzers, der den Vorgang zur Auslösung des Triggers ausführt.

Richtlinien für das Entwerfen von Triggern

- **Sie können Trigger entwerfen, um:**
 - zusammengehörige Aktionen auszuführen
 - globale Vorgänge zu zentralisieren
- **Sie dürfen keine Trigger entwerfen, wenn:**
 - die Funktionalität bereits im Oracle-Server vordefiniert ist
 - andere Trigger dadurch dupliziert werden
- **Sie können Stored Procedures erstellen und diese in einem Trigger aufrufen, wenn der PL/SQL-Code sehr lang ist.**
- **Eine übermäßige Verwendung von Triggern kann zu komplexen gegenseitigen Abhängigkeiten führen, die in großen Anwendungen schwer zu verwalten sind.**

ORACLE

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

- Verwenden Sie Trigger, um sicherzustellen, dass für einen bestimmten Vorgang zusammengehörige Aktionen ausgeführt werden. Verwenden Sie Trigger außerdem für zentralisierte globale Vorgänge, die für die Triggeranweisung ausgelöst werden sollen, wobei es keine Rolle spielt, von welchem Benutzer oder welcher Anwendung die Anweisung abgesetzt wird.
- Definieren Sie keine Trigger, die Funktionen duplizieren oder ersetzen, die bereits in der Oracle-Datenbank vordefiniert sind. Implementieren Sie beispielsweise Integritätsregeln mithilfe von deklarativen Constraints und nicht mit Triggern. Gehen Sie wie folgt vor, um die korrekte Reihenfolge beim Entwurf einer Geschäftsregel einzuhalten:
 - Verwenden Sie Built-In Constraints im Oracle-Server, z. B. Primärschlüssel.
 - Entwickeln Sie einen Datenbanktrigger oder eine Anwendung wie ein Servlet oder Enterprise JavaBean (EJB) auf der Middle Tier.
 - Verwenden Sie eine Präsentationsschnittstelle wie Oracle Forms, HTML oder JavaServer Pages (JSP) für Datenpräsentationsregeln.
- Eine übermäßige Verwendung von Triggern kann zu komplexen gegenseitigen Abhängigkeiten führen, die möglicherweise schwer zu verwalten sind. Verwenden Sie Trigger nur, wenn es nötig ist, und vermeiden Sie rekursive und kaskadierende Auswirkungen.
- Vermeiden Sie lange Triggerlogik, indem Sie Stored Procedures oder in einem Package integrierte Prozeduren erstellen, die im Triggerbody aufgerufen werden.
- Datenbanktrigger werden für jeden Benutzer immer dann ausgelöst, wenn das Ereignis für den erstellten Trigger eintritt.

Quiz

Welche der folgenden Aussagen treffen zu?

- a. Ein Trigger wird mit der Anweisung CREATE TRIGGER erstellt.**
- b. Der Quellcode eines Triggers ist im Data Dictionary USER_TRIGGERS enthalten.**
- c. Ein Trigger wird explizit aufgerufen.**
- d. Ein Trigger wird implizit von DML aufgerufen.**
- e. COMMIT, SAVEPOINT und ROLLBACK sind im Zusammenhang mit Triggern nicht zulässig.**

ORACLE

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

Richtige Antworten: a, b, d, e

Zusammenfassung

In dieser Lektion haben Sie Folgendes gelernt:

- **Komplexe Trigger beschreiben**
- **Sich verändernde Tabellen beschreiben**
- **Trigger für DDL-Anweisungen erstellen**
- **Trigger für Systemereignisse erstellen**
- **Informationen zu Triggern anzeigen**



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

Übungen zu Lektion 10 – Überblick: Komplexe, DDL- und Datenbankereignistrigger erstellen

Diese Übungen behandeln folgende Themen:

- **Komplexe Trigger zur Verwaltung von Datenintegritätsregeln erstellen**
- **Trigger erstellen, die eine Exception für eine sich verändernde Tabelle auslösen**
- **Trigger erstellen, die mithilfe des Packagestatus das Problem sich verändernder Tabellen lösen**

ORACLE

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

In diesen Übungen implementieren Sie eine einfache Geschäftsregel, um die Datenintegrität der Gehälter von Mitarbeitern hinsichtlich der gültigen Gehaltsspanne für ihre Tätigkeit sicherzustellen. Sie erstellen für diese Regel einen Trigger.

Während dieses Prozesses wirken sich die neuen Trigger kaskadierend auf Trigger aus, die in den Übungen zur vorherigen Lektion erstellt wurden. Die kaskadierende Auswirkung führt zu einer Exception für die sich verändernde Tabelle JOBS. Anschließend erstellen Sie ein PL/SQL-Package und zusätzliche Trigger, um das Problem der sich verändernden Tabelle zu beheben.

