



Oracle Database 12c: PL/SQL Fundamentals

Schulungsunterlagen
D80182DE11
Production 1.1 | November 2014 | D88611

Learn more from Oracle University at oracle.com/education/

Autor	Copyright © 2014, Oracle und/oder verbundene Unternehmen. All rights reserved. Alle Rechte vorbehalten.
Dimpi Rani Sarmah	
Technischer Inhalt und Überarbeitung	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.
Nancy Greenberg Swarnapriya Shridhar KimSeong Loh Miyuki Osato Laszlo Czinkoczki Madhavi Siddireddy Jim Spiller Christopher Wensley	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:
Redaktion	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.
Smita Kommini Malavika Jinka	
Herausgeber	Oracle und Java sind eingetragene Marken der Oracle Corporation und/oder ihrer verbundenen Unternehmen. Andere Namen und Bezeichnungen können Marken ihrer jeweiligen Inhaber sein.
Nita Brozowski Veena Narasimhan	

Inhaltsverzeichnis

1 Einführung

Ziele dieser Lektion 1-2
Kursziele 1-3
Schema **Human Resources** (HR) für diesen Kurs 1-4
Kursagenda 1-5
Informationen zu den Kursaccounts 1-6
In diesem Kurs verwendete Anhänge und Übungen 1-7
PL/SQL-Entwicklungsumgebungen 1-8
Was ist Oracle SQL Developer? 1-9
PL/SQL in SQL*Plus codieren 1-10
Oracle Cloud – Einführung 1-11
Oracle Cloud-Services 1-12
Cloud-Deployment-Modelle 1-13
SQL- und PL/SQL-Dokumentation von Oracle 1-14
Zusätzliche Ressourcen 1-15
Zusammenfassung 1-16
Übungen zu Lektion 1 – Übersicht: Erste Schritte 1-17

2 PL/SQL – Einführung

Ziele 2-2
Agenda 2-3
PL/SQL 2-4
PL/SQL-Laufzeitarchitektur 2-6
PL/SQL – Vorteile 2-7
PL/SQL-Blockstruktur 2-10
Agenda 2-12
Blocktypen 2-13
Programmkonstrukte 2-15
Anonyme Blöcke prüfen 2-17
Anonyme Blöcke ausführen 2-18
Agenda 2-19
Ausgabe von PL/SQL-Blöcken ermöglichen 2-20
Ausgabe von PL/SQL-Blöcken anzeigen 2-21
Quiz 2-22
Zusammenfassung 2-23
Übungen zu Lektion 2 – Übersicht 2-24

3 PL/SQL-Variablen deklarieren

Ziele 3-2

Agenda 3-3

Variablen – Einsatzmöglichkeiten 3-4

Variablennamen – Anforderungen 3-5

Variablen in PL/SQL 3-6

PL/SQL-Variablen deklarieren und initialisieren 3-7

Begrenzungszeichen in Zeichenfolgenliteralen 3-9

Agenda 3-10

Variabtentypen 3-11

PL/SQL-Variablen deklarieren und initialisieren – Richtlinien 3-13

PL/SQL-Variablen deklarieren – Richtlinien 3-14

In diesem Kurs verwendete PL/SQL-Strukturen – Benennungskonventionen 3-15

Skalare Datentypen 3-16

Skalare Basisdatentypen 3-17

Skalare Variablen deklarieren 3-21

Attribut %TYPE 3-22

Variablen mit dem Attribut %TYPE deklarieren 3-24

Boolesche Variablen deklarieren 3-25

Variablen mit LOB-Datentypen 3-26

Zusammengesetzte Datentypen – Records und Collections 3-27

Agenda 3-28

Bind-Variablen 3-29

Bind-Variablen referenzieren 3-31

AUTOPRINT mit Bind-Variablen 3-32

Quiz 3-33

Zusammenfassung 3-34

Übungen zu Lektion 3 – Übersicht 3-35

4 Ausführbare Anweisungen erstellen

Ziele 4-2

Agenda 4-3

Lexikalische Einheiten in PL/SQL-Blöcken 4-4

PL/SQL-Blöcke – Syntax und Richtlinien 4-6

Code kommentieren 4-7

SQL-Funktionen in PL/SQL 4-8

SQL-Funktionen in PL/SQL – Beispiele 4-9

Sequences in PL/SQL-Ausdrücken 4-10

Datentypen konvertieren 4-11

Agenda 4-14

Verschachtelte Blöcke 4-15
Verschachtelte Blöcke – Beispiel 4-16
Variablen – Gültigkeitsbereich und Sichtbarkeit 4-17
Qualifier mit verschachtelten Blöcken 4-19
Aufgabe: Gültigkeitsbereich von Variablen bestimmen 4-20
Agenda 4-22
Operatoren in PL/SQL 4-23
Operatoren in PL/SQL – Beispiele 4-24
Richtlinien für die Programmierung 4-25
Code einrücken 4-26
Quiz 4-27
Zusammenfassung 4-28
Übungen zu Lektion 4 – Übersicht 4-29

5 SQL-Anweisungen in PL/SQL-Blöcken

Ziele 5-2
Agenda 5-3
SQL-Anweisungen in PL/SQL 5-4
SELECT-Anweisungen in PL/SQL 5-5
Daten in PL/SQL abrufen – Beispiel 5-9
Daten in PL/SQL abrufen 5-10
Mehrdeutige Namen 5-11
Benennungskonventionen 5-12
Agenda 5-13
Daten mit PL/SQL bearbeiten 5-14
Daten einfügen – Beispiel 5-15
Daten aktualisieren – Beispiel 5-16
Daten löschen – Beispiel 5-17
Zeilen zusammenführen 5-18
Agenda 5-20
SQL-Cursor 5-21
SQL-Cursorattribute für implizite Cursor 5-23
Quiz 5-25
Zusammenfassung 5-26
Übungen zu Lektion 5 – Übersicht 5-27

6 Kontrollstrukturen erstellen

Ziele 6-2
Ablauf der Ausführung steuern 6-3
Agenda 6-4
IF-Anweisungen 6-5

Einfache IF-Anweisungen 6-7
IF THEN ELSE-Anweisungen 6-8
IF ELSIF ELSE-Klauseln 6-9
NULL-Werte in IF-Anweisungen 6-10
Agenda 6-11
CASE-Ausdrücke 6-12
CASE-Ausdrücke – Beispiel 6-13
Searched CASE-Ausdrücke 6-14
CASE-Anweisungen 6-15
NULL-Werte behandeln 6-16
Logiktabellen 6-17
Boolescher Ausdruck oder logischer Ausdruck? 6-18
Agenda 6-19
Iterative Kontrollstrukturen – LOOP-Anweisungen 6-20
Basisschleifen 6-21
Basisschleifen – Beispiel 6-22
WHILE-Schleifen 6-23
WHILE-Schleifen – Beispiel 6-24
FOR-Schleifen 6-25
FOR-Schleifen – Beispiel 6-27
FOR-Schleifen – Regeln 6-28
Einsatz von Schleifen – Empfehlungen 6-29
Verschachtelte Schleifen und Labels 6-30
Verschachtelte Schleifen und Labels – Beispiel 6-31
CONTINUE-Anweisungen in PL/SQL 6-32
CONTINUE-Anweisungen in PL/SQL – 1. Beispiel 6-33
CONTINUE-Anweisungen in PL/SQL – 2. Beispiel 6-34
Quiz 6-35
Zusammenfassung 6-36
Übungen zu Lektion 6 – Übersicht 6-37

7 Mit zusammengesetzten Datentypen arbeiten

Ziele 7-2
Agenda 7-3
Zusammengesetzte Datentypen 7-4
PL/SQL-Records oder -Collections? 7-5
Agenda 7-6
PL/SQL-Records 7-7
PL/SQL-Records erstellen 7-8
Struktur von PL/SQL-Records 7-9

Attribut %ROWTYPE 7-10
PL/SQL-Records erstellen – Beispiel 7-12
Attribut %ROWTYPE – Vorteile 7-13
Attribut %ROWTYPE – Weiteres Beispiel 7-14
Records mit %ROWTYPE einfügen 7-15
Zeilen in einer Tabelle mit einem Record aktualisieren 7-16
Agenda 7-17
Assoziative Arrays (INDEX BY-Tabellen) 7-18
Assoziative Arrays – Struktur 7-19
Assoziative Arrays erstellen – Schrittfolge 7-20
Assoziative Arrays erstellen und abrufen 7-21
INDEX BY-Tabellen – Methoden 7-22
INDEX BY-Record-Tabellen 7-23
INDEX BY-Record-Tabellen – 2. Beispiel 7-24
Nested Tables 7-25
VARRAYS 7-27
Collection-Typen – Zusammenfassung 7-28
Quiz 7-29
Zusammenfassung 7-30
Übungen zu Lektion 7 – Übersicht 7-31

8 Explizite Cursor

Ziele 8-2
Agenda 8-3
Cursor 8-4
Vorgänge mit expliziten Cursorn 8-5
Explizite Cursor kontrollieren 8-6
Agenda 8-7
Cursor deklarieren 8-8
Cursor öffnen 8-10
Daten aus Cursorn lesen 8-11
Cursor schließen 8-14
Cursor und Records 8-15
Cursor FOR-Schleifen 8-16
Attribute von expliziten Cursorn 8-18
Attribut %ISOPEN 8-19
%ROWCOUNT und %NOTFOUND – Beispiel 8-20
Cursor FOR-Schleifen mit Unterabfragen 8-21
Agenda 8-22
Cursor mit Parametern 8-23

Agenda 8-25
FOR UPDATE-Klauseln 8-26
WHERE CURRENT OF-Klauseln 8-28
Quiz 8-29
Zusammenfassung 8-30
Übungen zu Lektion 8 – Übersicht 8-31

9 Exceptions behandeln

Ziele 9-2
Agenda 9-3
Exceptions – Definition 9-4
Exceptions behandeln – Beispiel 9-5
Exceptions mit PL/SQL 9-6
Exceptions behandeln 9-7
Exception-Typen 9-8
Agenda 9-9
Exceptions abfangen – Syntax 9-10
Exceptions abfangen – Richtlinien 9-12
Vordefinierte Oracle-Serverfehler abfangen 9-13
Nicht vordefinierte Oracle-Serverfehler abfangen 9-16
Nicht vordefinierte Fehler abfangen – Beispiel 9-17
Exceptions abfangen – Funktionen 9-18
Benutzerdefinierte Exceptions abfangen 9-20
Exceptions in Unterblöcken propagieren 9-22
RAISE-Anweisungen 9-23
RAISE_APPLICATION_ERROR-Prozeduren 9-24
Quiz 9-27
Zusammenfassung 9-28
Übungen zu Lektion 9 – Übersicht 9-29

10 Stored Procedures und Stored Functions – Einführung

Ziele 10-2
Agenda 10-3
Prozeduren und Funktionen 10-4
Anonyme Blöcke und Unterprogramme – Unterschiede 10-5
Agenda 10-6
Prozeduren – Syntax 10-7
Prozeduren erstellen 10-8
Prozeduren aufrufen 10-10
Agenda 10-11

Funktionen – Syntax 10-12
Funktionen erstellen 10-13
Funktionen aufrufen 10-14
Parameter an Funktionen übergeben 10-15
Funktionen mit einem Parameter aufrufen 10-16
Quiz 10-17
Zusammenfassung 10-18
Übungen zu Lektion 10 – Übersicht 10-19

Anhang A: Tabellenbeschreibungen und -daten

Anhang B: SQL Developer

Ziele B-2
Was ist Oracle SQL Developer? B-3
SQL Developer – Spezifikationen B-4
SQL Developer 3.2 – Benutzeroberfläche B-5
Datenbankverbindungen erstellen B-7
Datenbankobjekte durchsuchen B-10
Tabellenstruktur anzeigen B-11
Dateien durchsuchen B-12
Schemaobjekte erstellen B-13
Neue Tabellen erstellen – Beispiel B-14
SQL Worksheet – Einsatzmöglichkeiten B-15
SQL-Anweisungen ausführen B-19
SQL-Skripte speichern B-20
Gespeicherte Skriptdateien ausführen – 1. Methode B-21
Gespeicherte Skriptdateien ausführen – 2. Methode B-22
SQL-Code formatieren B-23
Snippets B-24
Snippets – Beispiel B-25
Prozeduren und Funktionen debuggen B-26
Datenbankberichte B-27
Benutzerdefinierte Berichte erstellen B-28
Suchmaschinen und externe Tools B-29
Voreinstellungen festlegen B-30
SQL Developer-Layout zurücksetzen B-32
Data Modeler in SQL Developer B-33
Zusammenfassung B-34

Anhang C: SQL*Plus

- Ziele C-2
- SQL und SQL*Plus – Interaktion C-3
- SQL-Anweisungen und SQL*Plus-Befehle – Vergleich C-4
- SQL*Plus – Übersicht C-5
- Bei SQL*Plus anmelden C-6
- Tabellenstruktur 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 und START C-12
- Befehl SERVEROUTPUT C-15
- SQL*Plus-Befehl SPOOL C-16
- Befehl AUTOTRACE C-17
- Zusammenfassung C-18

Anhang D: Häufig verwendete SQL-Befehle

- Ziele D-2
- Einfache SELECT-Anweisungen D-3
- SELECT-Anweisungen D-4
- WHERE-Klauseln D-5
- ORDER BY-Klauseln D-6
- GROUP BY-Klauseln D-7
- Data Definition Language (DDL) D-8
- CREATE TABLE-Anweisungen D-9
- ALTER TABLE-Anweisungen D-10
- DROP TABLE-Anweisungen D-11
- GRANT-Anweisungen D-12
- Typen von Berechtigungen D-13
- REVOKE-Anweisungen D-14
- TRUNCATE TABLE-Anweisungen D-15
- Data Manipulation Language (DML) D-16
- INSERT-Anweisungen D-17
- UPDATE-Anweisungen – Syntax D-18
- DELETE-Anweisungen D-19
- Anweisungen zur Transaktionskontrolle D-20
- COMMIT-Anweisungen D-21
- ROLLBACK-Anweisungen D-22

SAVEPOINT-Anweisungen D-23
Joins D-24
Typen von Joins D-25
Mehrdeutige Spaltennamen eindeutig kennzeichnen D-26
Natural Joins D-28
Equi Joins D-29
Records mithilfe von Equi Joins abrufen D-30
Zusätzliche Suchbedingungen mit den Operatoren AND und WHERE D-31
Records mithilfe von Non-Equi Joins abrufen D-32
Records mithilfe von USING-Klauseln abrufen D-33
Records mithilfe von ON-Klauseln abrufen D-34
Left Outer Joins D-35
Right Outer Joins D-36
Full Outer Joins D-37
Self-Joins – Beispiel D-38
Cross Joins D-39
Zusammenfassung D-40

Anhang E: REF CURSOR

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

1

Einführung

ORACLE

Copyright © 2014, Oracle und/oder verbundene Unternehmen. All rights reserved. Alle Rechte vorbehalten.

Ziele dieser Lektion

Nach Ablauf dieser Lektion haben Sie folgende Ziele erreicht:

- Ziele des Kurses erörtern
- Im Kurs verwendetes Datenbankschema `HR` beschreiben
- In diesem Kurs verfügbare Benutzeroberflächen-umgebungen identifizieren
- Funktionen und Vorteile von Oracle Cloud beschreiben
- Verfügbare Anhänge, Dokumentationsunterlagen und andere Ressourcen referenzieren



Copyright © 2014, Oracle und/oder verbundene Unternehmen. All rights reserved. Alle Rechte vorbehalten.

In dieser Lektion erhalten Sie einen allgemeinen Überblick über den Kurs und den Kursablauf. Das im Kurs verwendete Datenbankschema und die verwendeten Tabellen werden vorgestellt. Sie lernen Oracle Cloud und Komponenten wie SQL, PL/SQL und Kompilierungsfeatures kennen. Darüber hinaus erhalten Sie eine Einführung in Tools wie SQL Developer.

Kursziele

Nach Ablauf dieses Kurses haben Sie folgende Ziele erreicht:

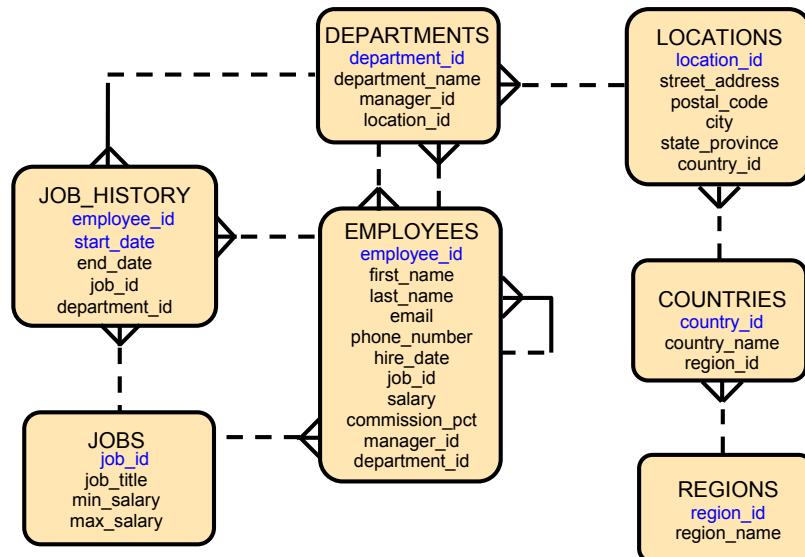
- Programmiererweiterungen identifizieren, um die SQL durch PL/SQL ergänzt wird
- PL/SQL-Code als Schnittstelle zur Datenbank erstellen
- Anonyme PL/SQL-Blöcke entwerfen, die effizient ausgeführt werden
- PL/SQL-Programmierkonstrukte und `IF`-Anweisungen verwenden
- Laufzeitfehler behandeln
- Stored Procedures und Stored Functions beschreiben



Copyright © 2014, Oracle und/oder verbundene Unternehmen. All rights reserved. Alle Rechte vorbehalten.

In diesem Kurs werden die Grundlagen von PL/SQL vorgestellt. Sie lernen die PL/SQL-Syntax, -Blöcke und -Programmierkonstrukte kennen und erfahren, welche Vorteile sich aus der Integration von SQL mit diesen Konstrukten ergeben. Im Kurs wird erklärt, wie Sie PL/SQL-Programmeinheiten erstellen und effizient ausführen. Darüber hinaus erfahren Sie, wie Sie SQL Developer als Entwicklungsumgebung für PL/SQL verwenden und wiederverwendbare Programmeinheiten wie Prozeduren und Funktionen entwickeln.

Schema "Human Resources" (HR) für diesen Kurs



ORACLE

Copyright © 2014, Oracle und/oder verbundene Unternehmen. All rights reserved. Alle Rechte vorbehalten.

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 oder Asien 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.

Kursagenda

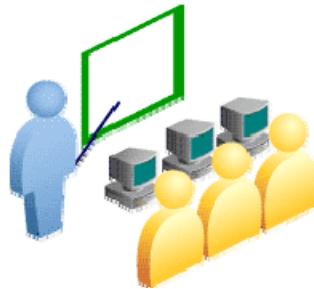
- 1. Tag:
 1. Einführung
 2. PL/SQL – Einführung
 3. PL/SQL-Variablen deklarieren
 4. Ausführbare Anweisungen erstellen
 5. SQL-Anweisungen in PL/SQL-Blöcken
 6. Kontrollstrukturen erstellen
- 2. Tag:
 7. Mit zusammengesetzten Datentypen arbeiten
 8. Explizite Cursor
 9. Exceptions behandeln
 10. Stored Procedures und Stored Functions – Einführung

ORACLE

Copyright © 2014, Oracle und/oder verbundene Unternehmen. All rights reserved. Alle Rechte vorbehalten.

Informationen zu den Kursaccounts

- Es wurden geklonte HR-Account-IDs für Sie eingerichtet.
- Ihre Account-ID lautet ora41.
- Das Kennwort stimmt mit der Account-ID überein.
- Jeder Rechner verfügt über eine vollständige Umgebung und ist demselben Account zugewiesen.



ORACLE

Copyright © 2014, Oracle und/oder verbundene Unternehmen. All rights reserved. Alle Rechte vorbehalten.

In diesem Kurs verwendete Anhänge und Übungen

- Anhang A: Tabellenbeschreibungen und -daten
- Anhang B: SQL Developer
- Anhang C: SQL*Plus
- Anhang D: Häufig verwendete SQL-Befehle
- Anhang E: REF CURSOR
- Übungsband: Übungen und Lösungen

ORACLE

Copyright © 2014, 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 © 2014, Oracle und/oder verbundene Unternehmen. All rights reserved. Alle Rechte vorbehalten.

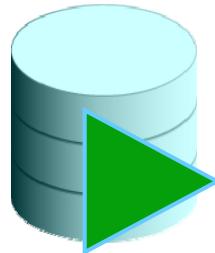
Oracle bietet verschiedene Tools, mit denen Sie PL/SQL-Code erstellen können. In diesem Kurs stehen unter anderem folgende Entwicklungstools zur Verfügung:

- **Oracle SQL Developer:** Grafisches Tool
- **Oracle SQL*Plus:** Fenster- oder Befehlszeilenanwendung

Hinweis: Die im Kursmaterial verwendeten Code- und Bildschirmbeispiele wurden auf Grundlage der Ausgabe in der SQL Developer-Umgebung generiert.

Was ist Oracle SQL Developer?

- Oracle SQL Developer ist ein kostenloses grafisches Tool, das die Produktivität erhöht und Aufgaben der Datenbankentwicklung vereinfacht.
- 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 © 2014, Oracle und/oder verbundene Unternehmen. All rights reserved. Alle Rechte vorbehalten.

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

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

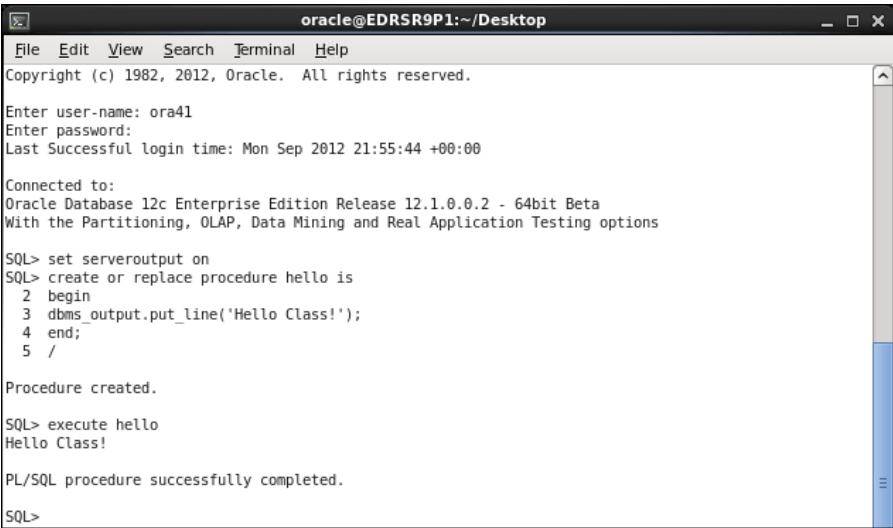
- Datenbankobjekte durchsuchen und verwalten
- SQL-Anweisungen und -Skripte ausführen
- PL/SQL-Anweisungen bearbeiten und debuggen
- Berichte erstellen

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

Anhang B:

In Anhang B dieses Kurses erhalten Sie eine Einführung in den Umgang mit der Benutzeroberfläche von SQL Developer. Informieren Sie sich in Anhang B über das Erstellen einer Datenbankverbindung, die Interaktion mit Daten mittels SQL und PL/SQL und andere Themen. Alternativ können Sie sich die Einführungs-demonstration zu SQL Developer ansehen.

PL/SQL in SQL*Plus codieren



The screenshot shows a terminal window titled "oracle@EDRSR9P1:~/Desktop". The session starts with the Oracle copyright notice. The user logs in as "ora41". The session then connects to an Oracle Database 12c Enterprise Edition Release 12.1.0.0.2 - 64bit Beta. The user creates a new PL/SQL procedure named "hello" with a single line of code that prints "Hello Class!". The procedure is successfully created and executed, displaying the output "Hello Class!".

```
oracle@EDRSR9P1:~/Desktop
Copyright (c) 1982, 2012, Oracle. All rights reserved.

Enter user-name: ora41
Enter password:
Last Successful login time: Mon Sep 2012 21:55:44 +00:00

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

SQL> set serveroutput on
SQL> create or replace procedure hello is
 2 begin
 3   dbms_output.put_line('Hello Class!');
 4 end;
 5 /

Procedure created.

SQL> execute hello
Hello Class!

PL/SQL procedure successfully completed.

SQL>
```

ORACLE

Copyright © 2014, Oracle und/oder verbundene Unternehmen. All rights reserved. Alle Rechte vorbehalten.

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

SQL*Plus:

- ist im Lieferumfang der Datenbank enthalten
- wird auf einem Client und im Datenbankserversystem 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 in Ihrer Session zunächst den Befehl `SET SERVEROUTPUT ON` ausführen.

Hinweise

- Um SQL*Plus in einer Linux-Umgebung zu starten, öffnen Sie ein Terminalfenster und geben den Befehl `sqlplus` ein.
- Informationen zur Verwendung von SQL*Plus finden Sie im Anhang C.

Oracle Cloud – Einführung

Oracle Cloud ist eine Unternehmenscloud zur geschäftlichen Nutzung. Oracle Public Cloud besteht aus vielen verschiedenen Services, die sich durch einige gemeinsame Eigenschaften auszeichnen:

- On-Demand Self-Service
- Ressourcen-Pooling
- Rasche Elastizität
- Gemessener Service
- Breiter Netzwerkzugriff

www.cloud.oracle.com



ORACLE

Copyright © 2014, Oracle und/oder verbundene Unternehmen. All rights reserved. Alle Rechte vorbehalten.

Oracle Cloud ist eine Unternehmenscloud zur geschäftlichen Nutzung. Sie stellt eine integrierte Sammlung von Cloudservices für Anwendungen und Plattformen bereit, die auf branchenführenden Produkten sowie offenen Java- und SQL-Standards basieren.

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 Cloudservices lassen sich über eine Selfservice-Schnittstelle bereitstellen. Benutzer können ihre Cloudservices auf einer integrierten Entwicklungs- und Deployment-Plattform mit entsprechenden Tools zur schnellen Erweiterung und Erstellung 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. Die beiden größten Vorteile von Cloud-Computing sind Geschwindigkeit und Kosten.

Dabei stechen fünf grundlegende Eigenschaften hervor:

- **On-Demand-Selfservice:** Provisioning, Überwachung, Managementkontrolle
- **Ressourcen-Pooling:** Gemeinsame Nutzung und Abstraktionsebene zwischen Services und ihren Consumern
- **Rasche Elastizität:** Fähigkeit der bedarfsabhängigen vertikalen Skalierung
- **Gemessener Service:** Nutzungsmessung für internes Chargeback (Private Cloud) oder externe Abrechnung (Public Cloud)
- **Breiter Netzwerkzugriff:** Bedeutet in der Regel Zugriff auf jedes Netzwerk-Device über einen Browser

Oracle Cloud-Services

Oracle Cloud bietet folgende drei Typen von Services:

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



ORACLE

Copyright © 2014, 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 zugehörige Software sowie Betriebssysteme, Virtualisierung, Clustering und so weiter. IaaS-Beispiele in der Public Cloud: Amazon Elastic Compute Cloud (EC2) und Amazon Simple Storage Service (S3)

Die Datenbankcloud wird innerhalb der Private Cloud-Umgebung eines Unternehmens als PaaS-Modell erstellt. Die Datenbankcloud bietet On-Demand-Zugriff auf Datenbankservices, der im Selfservice-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.

Cloud-Deployment-Modelle

- **Private Cloud:** Exklusive Verwendung durch eine einzige Organisation; Steuerung, Verwaltung und Hosting in der Regel in privaten Data Centern
- **Public Cloud:** Verwendung durch mehrere Organisationen (Mandanten) auf gemeinsamer Basis; Hosting und Verwaltung durch fremden Serviceprovider
- **Community Cloud:** Verwendung durch eine Gruppe zusammengehöriger Organisationen, die eine gemeinsame Cloud-Computing-Umgebung nutzen möchten
- **Hybrid Cloud:** Verwendung durch eine einzige Organisation, die Private und Public Clouds für eine einzige Anwendung nutzen möchte, um von den Vorteilen beider Clouds zu profitieren

ORACLE®

Copyright © 2014, Oracle und/oder verbundene Unternehmen. All rights reserved. Alle Rechte vorbehalten.

Private Cloud

Hosting und Betrieb von Private Clouds können auch an einen externen Serviceprovider ausgelagert werden, ihre Nutzung bleibt allerdings exklusiv einer einzigen Organisation vorbehalten.

Community Cloud

Beispiel: Communitys können aus den verschiedenen Bereichen des Militärs, aus allen Universitäten einer bestimmten Region oder aus allen Lieferanten eines großen Herstellers gebildet werden.

Hybrid Cloud

Beispiel: In einem "Cloud-Bursting"-Szenario kann eine Organisation die stabile Workload einer Anwendung in einer Private Cloud ausführen. Im Falle einer Workload-Spitze jedoch (etwa am Ende eines Finanzquartals oder während der Urlaubssaison) kann die Anwendung zusätzlich Rechenkapazität einer Public Cloud nutzen. Sobald diese Ressourcen nicht mehr benötigt werden, gibt die Anwendung sie an den öffentlichen Pool zurück.

SQL- und PL/SQL-Dokumentation von Oracle

- *Oracle Database New Features 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 © 2014, Oracle und/oder verbundene Unternehmen. All rights reserved. Alle Rechte vorbehalten.

Navigieren Sie zu <http://www.oracle.com/pls/db121/homepage>. Dort erhalten Sie Zugriff auf die Dokumentationsbibliothek von Oracle Database 12c.

Zusätzliche Ressourcen

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

- 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" auf Oracle Technology Network (OTN):
 - <http://www.oracle.com/technetwork/database/features/plsql/index.html>
- SQL Developer-Onlinehomepage und SQL Developer-Tutorial:
 - http://www.oracle.com/technology/products/database/sql_developer/index.html
 - <http://download.oracle.com/oll/tutorials/SQLDeveloper/index.htm>



Copyright © 2014, Oracle und/oder verbundene Unternehmen. All rights reserved. Alle Rechte vorbehalten.

Zusammenfassung

In dieser Lektion haben Sie Folgendes gelernt:

- Ziele des Kurses erörtern
- Im Kurs verwendetes Datenbankschema `HR` beschreiben
- In diesem Kurs verfügbare Benutzeroberflächen-umgebungen identifizieren
- Funktionen und Vorteile von Oracle Cloud beschreiben
- Verfügbare Anhänge, Dokumentationsunterlagen und andere Ressourcen referenzieren



Copyright © 2014, Oracle und/oder verbundene Unternehmen. All rights reserved. Alle Rechte vorbehalten.

Übungen zu Lektion 1 – Übersicht: Erste Schritte

Diese Übung behandelt folgende Themen:

- SQL Developer starten
- Neue Datenbankverbindung erstellen
- Tabellen des Schemas `HR` durchsuchen
- SQL Developer-Voreinstellung festlegen



Copyright © 2014, Oracle und/oder verbundene Unternehmen. All rights reserved. Alle Rechte vorbehalten.

In dieser Übung führen Sie mit SQL Developer SQL-Anweisungen aus, um Daten im Schema `HR` zu prüfen. Außerdem erstellen Sie einen einfachen anonymen Block.

Hinweis: Für alle schriftlich vorliegenden Übungen wird SQL Developer als Entwicklungsumgebung verwendet. Den Kursteilnehmern wird ebenfalls die Verwendung von SQL Developer empfohlen, allerdings steht ihnen in diesem Kurs als Alternative auch SQL*Plus zur Verfügung.

PL/SQL – Einführung

2

ORACLE

Copyright © 2014, Oracle und/oder verbundene Unternehmen. All rights reserved. Alle Rechte vorbehalten.

Ziele

Nach Ablauf dieser Lektion haben Sie folgende Ziele erreicht:

- Notwendigkeit von PL/SQL beschreiben
- Vorteile von PL/SQL erläutern
- Verschiedene Typen von PL/SQL-Blöcken identifizieren
- Ausgabemeldungen in PL/SQL erläutern



Copyright © 2014, Oracle und/oder verbundene Unternehmen. All rights reserved. Alle Rechte vorbehalten.

In dieser Lektion erhalten Sie eine Einführung in PL/SQL und in die PL/SQL-Programmierkonstrukte. Außerdem lernen Sie die Vorteile von PL/SQL kennen.

Agenda

- Vorteile und Struktur von PL/SQL kennenlernen
- PL/SQL-Blöcke prüfen
- Ausgabemeldungen in PL/SQL generieren

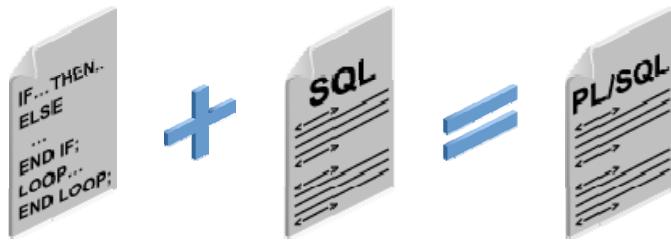
ORACLE

Copyright © 2014, Oracle und/oder verbundene Unternehmen. All rights reserved. Alle Rechte vorbehalten.

PL/SQL

PL/SQL:

- ist die prozedurale Spracherweiterung von SQL
- ist die Standardsprache von Oracle für den Datenzugriff bei relationalen Datenbanken
- integriert prozedurale Konstrukte nahtlos in SQL



ORACLE

Copyright © 2014, Oracle und/oder verbundene Unternehmen. All rights reserved. Alle Rechte vorbehalten.

Structured Query Language (SQL) ist die primäre Sprache für den Zugriff auf und die Bearbeitung von Daten in relationalen Datenbanken. Die wenigen SQL-Befehle sind leicht zu erlernen und zu verwenden.

Beispiel:

```
SELECT first_name, department_id, salary FROM employees;
```

Die obige SQL-Anweisung lässt sich einfach und schnell erstellen. Wenn Sie jedoch Daten ändern möchten, die mithilfe von Bedingungen abgerufen werden, stoßen Sie rasch an die Grenzen von SQL.

Dazu die folgende leicht abgewandelte Aufgabenstellung: Für jeden abgerufenen Mitarbeiter sollen die Abteilungsnummer (**department_id**) und das Gehalt (**salary**) geprüft werden. In Abhängigkeit von der Leistung der Abteilung und vom Gehalt der Mitarbeiter sollen den Mitarbeitern gegebenenfalls unterschiedliche Prämien zugeteilt werden.

Für diese Aufgabe müssen Sie die oben dargestellte SQL-Anweisung ausführen, die Daten sammeln und Logik auf die Daten anwenden.

- Eine Möglichkeit wäre, für jede Abteilung eine SQL-Anweisung zu erstellen, die den Mitarbeitern der Abteilung Prämien zuteilt. Bevor Sie die Prämienhöhe festlegen, müssen Sie jedoch zunächst das Gehalt prüfen. Dies macht die Aufgabe etwas komplizierter.
- Eine effektivere Lösung besteht darin, Bedingungsanweisungen einzubinden. PL/SQL ist für Anforderungen dieser Art konzipiert. Die Sprache bietet eine Programmiererweiterung für das bereits vorhandene SQL.

PL/SQL

PL/SQL:

- stellt eine Blockstruktur für ausführbare Codeeinheiten bereit. Die Verwaltung von Code wird durch diese klar definierte Struktur vereinfacht.
- bietet prozedurale Konstrukte wie:
 - Variablen, Konstanten und Datentypen
 - Kontrollstrukturen wie Bedingungsanweisungen und Schleifen
 - wiederverwendbare Programmblöcke, die einmal erstellt und mehrmals ausgeführt werden

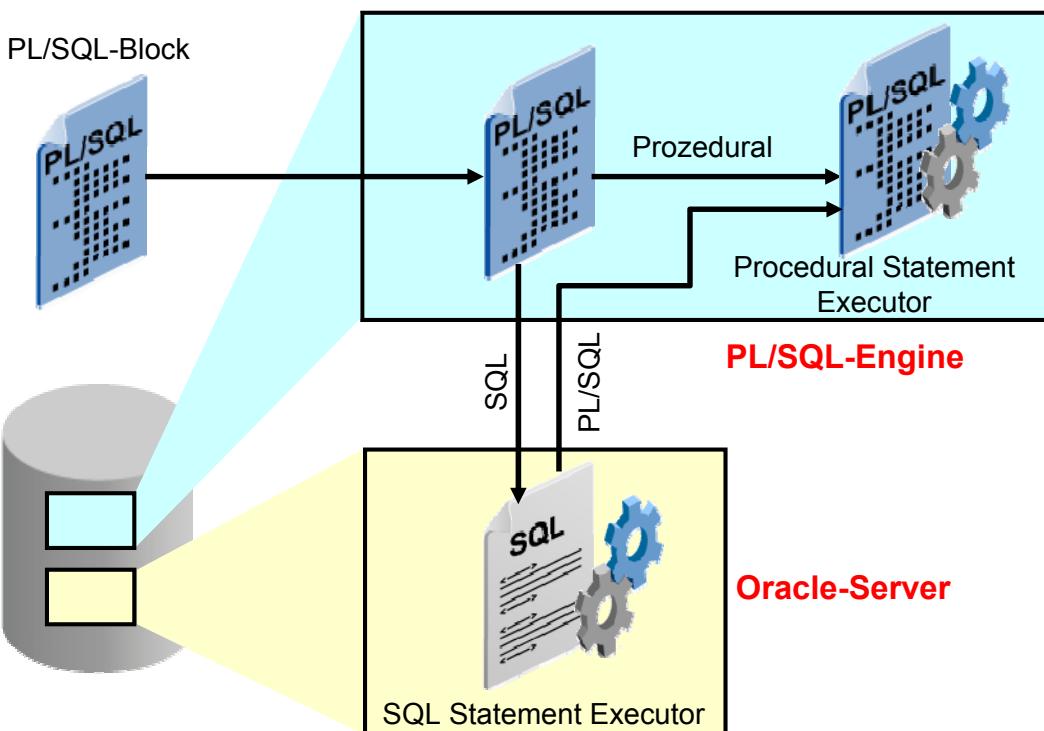
ORACLE

Copyright © 2014, Oracle und/oder verbundene Unternehmen. All rights reserved. Alle Rechte vorbehalten.

PL/SQL definiert eine Blockstruktur zum Erstellen von Code. Diese Struktur erleichtert die Verwaltung und das Debugging von Code, da Ablauf und Ausführung des Programmblöckes leicht verständlich sind.

PL/SQL stellt Features des modernen Softwareengineerings wie Datenkapselung, Exception-Behandlung, Information Hiding sowie objektorientiertes Programmieren zur Verfügung. Damit sind für den Oracle-Server und seine Tools die neuesten Programmiermethoden einsetzbar. PL/SQL enthält alle prozeduralen Konstrukte, die in den 3GL-(Third Generation Language-) Programmiersprachen verfügbar sind.

PL/SQL-Laufzeitarchitektur



ORACLE

Copyright © 2014, Oracle und/oder verbundene Unternehmen. All rights reserved. Alle Rechte vorbehalten.

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

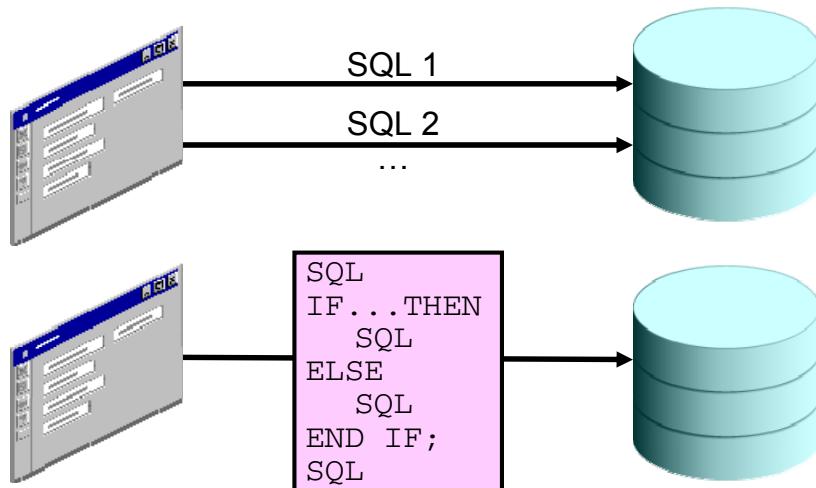
- in der Oracle-Datenbank zur Ausführung von Stored Subprograms
- im Oracle Forms-Client, wenn Client/Server-Anwendungen ausgeführt werden, oder in Oracle Application Server, wenn Oracle Forms Services zur Ausführung von Forms im Web verwendet wird

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 wird die PL/SQL-Umgebung aufgerufen, wenn eine PL/SQL-Funktion in einer SELECT-Anweisung verwendet wird.

Die PL/SQL-Engine ist eine speicherresidente virtuelle Maschine, die 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/Server-Implementierung 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.

PL/SQL – Vorteile

- Integration prozeduraler Konstrukte in SQL
- Bessere Performance



ORACLE

Copyright © 2014, Oracle und/oder verbundene Unternehmen. All rights reserved. Alle Rechte vorbehalten.

Integration prozeduraler Konstrukte in SQL: Der wichtigste Vorteil von PL/SQL ist die Integration der prozeduralen Konstrukte in SQL. SQL ist eine nicht prozedurale Sprache. Durch das Absetzen eines SQL-Befehls teilen Sie dem Datenbankserver mit, was er tun soll. Sie können jedoch nicht angeben, wie er die Aufgabe ausführen soll. PL/SQL integriert Kontroll- und Bedingungsanweisungen in SQL. Dadurch haben Sie eine bessere Kontrolle über Ihre SQL-Anweisungen und deren Ausführung. In dieser Lektion wurde bereits ein Beispiel für die Notwendigkeit der Integration vorgestellt.

Bessere Performance: Ohne PL/SQL könnten Sie SQL-Anweisungen nicht logisch zu einer Einheit zusammenfassen. Wenn Sie eine Anwendung entwickelt haben, die Forms enthält, kann sich diese aus vielen unterschiedlichen Forms mit verschiedenen Feldern zusammensetzen. Falls eine Form Daten weiterleitet, muss möglicherweise eine Reihe von SQL-Anweisungen ausgeführt werden. SQL-Anweisungen werden nacheinander an die Datenbank gesendet. Dies führt zu vielen Netzwerkläufen und zu einem Datenbankaufruf für jede SQL-Anweisung. Der Datenverkehr im Netzwerk nimmt zu, und die Performance ist besonders bei Client/Server-Modellen beeinträchtigt.

Mit PL/SQL können Sie alle SQL-Anweisungen zu einer einzigen Programmeinheit zusammenfassen. Die Anwendung sendet nicht jede SQL-Anweisung einzeln, sondern den gesamten Block an die Datenbank. Die Anzahl der Datenbankaufrufe wird damit erheblich verringert. Wie auf der Folie dargestellt, können Sie die SQL-Anweisungen bei SQL-intensiven Anwendungen mithilfe von PL/SQL-Blöcken gruppieren, bevor sie zur Ausführung an den Oracle-Datenbankserver gesendet werden.

PL/SQL – Vorteile

- Modularisierte Programmentwicklung
- Integration in Oracle-Tools
- Portierbarkeit
- Exception-Behandlung

ORACLE

Copyright © 2014, Oracle und/oder verbundene Unternehmen. All rights reserved. Alle Rechte vorbehalten.

Modularisierte Programmentwicklung: Die Basiseinheit in einem PL/SQL-Programm ist der Block. Blöcke können sequenziell angeordnet oder in anderen Blöcken verschachtelt sein. Die modularisierte Programmentwicklung bietet folgende Vorteile:

- Sie können logisch zusammengehörige Anweisungen in Blöcken gruppieren.
- Sie können Blöcke in größeren Blöcken verschachteln, um leistungsfähige Programme zu erstellen.
- Sie können die Anwendung in kleinere Module gliedern. Bei der Entwicklung komplexer Anwendungen können Sie mit PL/SQL die Anwendung in kleinere, verwaltbare und logisch zusammengehörige Module einteilen.
- Sie können den Code einfach verwalten und debuggen.

In PL/SQL wird die Modularisierung mithilfe von Prozeduren, Funktionen und Packages implementiert. Sie werden in der Lektion "Stored Procedures und Stored Functions – Einführung" erläutert.

Integration in Tools: Die PL/SQL-Engine ist in viele Oracle-Tools wie Oracle Forms und Oracle Reports integriert. In diesen Tools verarbeitet die lokal verfügbare PL/SQL-Engine die prozeduralen Anweisungen. Es werden nur die SQL-Anweisungen an die Datenbank übergeben.

Portierbarkeit: PL/SQL-Programme lassen sich unabhängig von Betriebssystem oder Plattform überall ausführen, wo auch der Oracle-Server ausgeführt wird. Sie müssen die Programme nicht an jede neue Umgebung anpassen. Sie können portierbare Programmpackages und Librarys erstellen, die in unterschiedlichen Umgebungen wiederverwendbar sind.

Exception-Behandlung: PL/SQL ermöglicht Ihnen eine effiziente Exception-Behandlung. Sie können separate Blöcke zur Verarbeitung von Exceptions definieren. Weitere Informationen zur Verarbeitung von Exceptions erhalten Sie in der Lektion "Exceptions behandeln".

PL/SQL verwendet dasselbe Datentypsysteem wie SQL (mit einigen Erweiterungen) und dieselbe Syntax für Ausdrücke.

PL/SQL-Blockstruktur

- **DECLARE (optional)**
 - Variablen, Cursor, benutzerdefinierte Exceptions
- **BEGIN (obligatorisch)**
 - SQL-Anweisungen
 - PL/SQL-Anweisungen
- **EXCEPTION (optional)**
 - Auszuführende Aktionen im Falle von Exceptions
- **END; (obligatorisch)**



ORACLE

Copyright © 2014, Oracle und/oder verbundene Unternehmen. All rights reserved. Alle Rechte vorbehalten.

Die Folie zeigt einen einfachen PL/SQL-Block. Ein PL/SQL-Block besteht aus vier Bereichen:

- **Deklarativer Bereich (optional):** Der deklarative Bereich beginnt mit dem Schlüsselwort `DECLARE` und endet mit Beginn des ausführbaren Bereichs.
- **Ausführbarer Bereich (obligatorisch):** Der ausführbare Bereich beginnt mit dem Schlüsselwort `BEGIN`. Dieser Bereich muss mindestens eine Anweisung enthalten. Der ausführbare Bereich eines PL/SQL-Blockes kann jedoch beliebig viele weitere PL/SQL-Blöcke enthalten.
- **Bereich zur Exception-Behandlung (optional):** Der Exception-Bereich ist im ausführbaren Bereich verschachtelt. Er beginnt mit dem Schlüsselwort `EXCEPTION`.
- **END (obligatorisch):** Am Ende jedes PL/SQL-Blockes muss eine `END`-Anweisung stehen. `END` wird mit einem Semikolon abgeschlossen.

In einem PL/SQL-Block werden die Schlüsselwörter `DECLARE`, `BEGIN` und `EXCEPTION` nicht mit einem Semikolon abgeschlossen. Das Schlüsselwort `END` und alle SQL- sowie PL/SQL-Anweisungen müssen hingegen mit einem Semikolon abgeschlossen werden.

Bereich	Beschreibung	Bestandteil
Deklarativer Bereich <code>(DECLARE)</code>	Enthält die Deklarationen aller Variablen, Konstanten, Cursor und benutzerdefinierten Exceptions, die im ausführbaren Bereich und im Exception-Bereich referenziert werden	Optional
Ausführbarer Bereich <code>(BEGIN ... END)</code>	Enthält SQL-Anweisungen zum Abrufen von Daten aus der Datenbank und PL/SQL-Anweisungen zur Datenmanipulation im Block	Obligatorisch
Exception-Bereich <code>(EXCEPTION)</code>	Legt die Aktionen fest, die ausgeführt werden sollen, wenn Fehler und anormale Bedingungen im ausführbaren Bereich auftreten	Optional

Agenda

- Vorteile und Struktur von PL/SQL kennenlernen
- PL/SQL-Blöcke prüfen
- Ausgabemeldungen in PL/SQL generieren

ORACLE

Copyright © 2014, Oracle und/oder verbundene Unternehmen. All rights reserved. Alle Rechte vorbehalten.

Blocktypen

Anonym

```
[DECLARE]  
  
BEGIN  
  --statements  
  
[EXCEPTION]  
  
END;
```

Prozedur

```
PROCEDURE name  
IS  
BEGIN  
  --statements  
[EXCEPTION]  
END;
```

Funktion

```
FUNCTION name  
RETURN datatype  
IS  
BEGIN  
  --statements  
  RETURN value;  
[EXCEPTION]  
END;
```

ORACLE

Copyright © 2014, Oracle und/oder verbundene Unternehmen. All rights reserved. Alle Rechte vorbehalten.

Ein PL/SQL-Programm besteht aus einem oder mehreren Blöcken. Die Blöcke können klar voneinander getrennt oder ineinander verschachtelt sein.

Es gibt drei Typen von Blöcken, die ein PL/SQL-Programm bilden:

- Anonyme Blöcke
- Prozeduren
- Funktionen

Anonyme Blöcke: Anonyme Blöcke sind unbenannte Blöcke. Sie werden inline deklariert, also an der Stelle in der Anwendung, an der sie auszuführen sind. Die Kompilierung der Blöcke erfolgt bei jeder Ausführung der Anwendung. Die Blöcke werden nicht in der Datenbank gespeichert. Sie werden an die PL/SQL-Engine übergeben und dort zur Laufzeit ausgeführt. Trigger in Oracle Developer-Komponenten bestehen aus solchen Blöcken.

Wenn Sie denselben Block erneut ausführen möchten, müssen Sie ihn neu erstellen. Sie können keine zuvor erstellten Blöcke aufrufen, da die Blöcke anonym und nach der Ausführung nicht mehr vorhanden sind.

Prozeduren: Prozeduren sind benannte Objekte, die SQL- und/oder PL/SQL-Anweisungen enthalten.

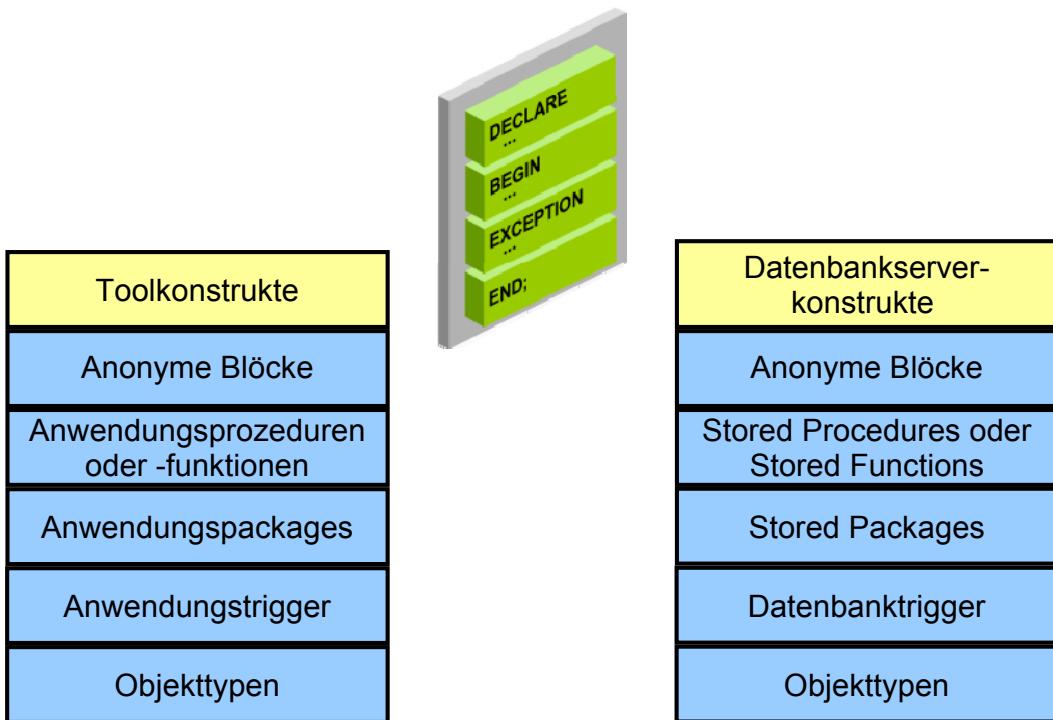
Funktionen: Funktionen sind benannte Objekte, die SQL- und/oder PL/SQL-Anweisungen enthalten. Im Gegensatz zu einer Prozedur gibt eine Funktion einen Wert eines festgelegten Datentyps zurück.

Unterprogramme

Unterprogramme ergänzen anonyme Blöcke. Sie sind benannte PL/SQL-Blöcke, die in der Datenbank gespeichert sind. Da sie benannt und gespeichert sind, können sie jederzeit aufgerufen werden (abhängig von der Anwendung). Sie können Unterprogramme als Prozeduren oder Funktionen deklarieren. In der Regel verwenden Sie Prozeduren zum Durchführen von Aktionen und Funktionen zum Berechnen und Zurückgeben von Werten.

Unterprogramme lassen sich auf Server- oder Anwendungsebene speichern. Mit Komponenten von Oracle Developer (Forms, Reports) können Sie Prozeduren und Funktionen als Teil der Anwendung (als Form oder Bericht) deklarieren und von anderen Prozeduren, Funktionen und Triggern innerhalb derselben Anwendung bei Bedarf aufrufen.

Programmkonstrukte



ORACLE

Copyright © 2014, Oracle und/oder verbundene Unternehmen. All rights reserved. Alle Rechte vorbehalten.

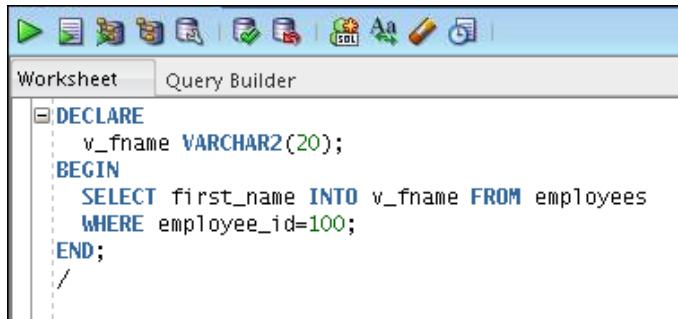
In der folgenden Tabelle sind verschiedene PL/SQL-Programmkonstrukte aufgeführt, die den einfachen PL/SQL-Block verwenden. Die Verfügbarkeit der Programmkonstrukte hängt von der Umgebung ab, in der sie ausgeführt werden.

Programm-konstrukt	Beschreibung	Verfügbarkeit
Anonyme Blöcke	Unbenannte PL/SQL-Blöcke, die in eine Anwendung eingebettet sind oder interaktiv ausgegeben werden	In allen PL/SQL-Umgebungen
Anwendungs-prozeduren oder -funktionen	Benannte PL/SQL-Blöcke, die in einer Oracle Forms Developer-Anwendung oder Shared Library gespeichert sind. Sie können Parameter annehmen und anhand ihres Namens wiederholt aufgerufen werden.	In Komponenten von Oracle Developer-Tools, beispielsweise Oracle Forms Developer, Oracle Reports
Stored Procedures oder Stored Functions	Benannte PL/SQL-Blöcke, die im Oracle-Server gespeichert sind. Sie können Parameter annehmen und anhand ihres Namens wiederholt aufgerufen werden.	Im Oracle-Server oder in Oracle Developer-Tools
Packages (Anwendungs-packages oder Stored Packages)	Benannte PL/SQL-Module, in denen zusammengehörige Prozeduren, Funktionen und Bezeichner gruppiert sind	Im Oracle-Server und in Komponenten von Oracle Developer-Tools, beispielsweise Oracle Forms Developer

Programm-konstrukt	Beschreibung	Verfügbarkeit
Datenbanktrigger	PL/SQL-Blöcke, die einer Datenbanktabelle zugeordnet sind und durch verschiedene Ereignisse automatisch ausgelöst werden	Oracle-Server oder alle Oracle-Tools, die DML absetzen
Anwendungs-trigger	PL/SQL-Blöcke, die entweder einer Datenbanktabelle oder Systemereignissen zugeordnet sind. Sie werden automatisch durch ein DML-Ereignis oder Systemereignis ausgelöst.	Oracle Developer-Tools, beispielsweise Oracle Forms Developer
Objekttypen	Benutzerdefinierte, zusammengesetzte Datentypen, die eine Datenstruktur mit den für die Datenmanipulation notwendigen Funktionen und Prozeduren kapseln	Oracle-Server und Oracle Developer-Tools

Anonyme Blöcke prüfen

Anonymer Block im SQL Developer-Workspace:



The screenshot shows the Oracle SQL Developer interface with the 'Worksheet' tab selected. The code area contains the following PL/SQL block:

```
DECLARE
    v_fname VARCHAR2(20);
BEGIN
    SELECT first_name INTO v_fname FROM employees
    WHERE employee_id=100;
END;
/
```

ORACLE

Copyright © 2014, Oracle und/oder verbundene Unternehmen. All rights reserved. Alle Rechte vorbehalten.

Um einen anonymen Block mit SQL Developer zu erstellen, geben Sie den Block im Workspace ein (siehe Folie).

Beispiel

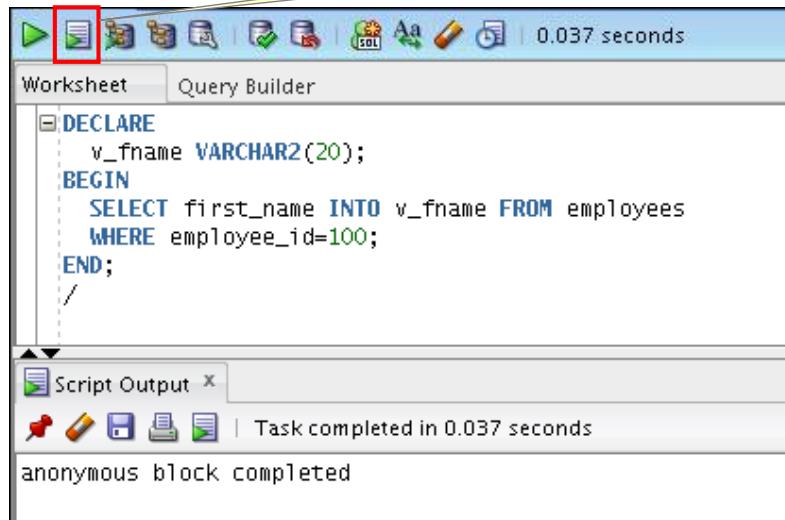
Der Beispielblock weist den deklarativen und den ausführbaren Bereich auf. Die Syntax der Anweisungen im Block müssen Sie nicht beachten. Sie wird im weiteren Verlauf dieses Kurses erläutert.

Der anonyme Block ruft den Vornamen (`first_name`) des Mitarbeiters ab, dessen Personalnummer (`employee_id`) 100 lautet, und speichert ihn in der Variablen `v_fname`.

Anonyme Blöcke ausführen

Anonymen Block durch Klicken auf die Schaltfläche
Run Script ausführen:

Run Script (oder F5)



The screenshot shows the Oracle SQL Developer interface. In the top toolbar, the 'Worksheet' tab is selected. A yellow callout box points to the 'Run Script' button (F5), which is highlighted with a red border. Below the toolbar, the code editor contains the following PL/SQL block:

```
DECLARE
    v_fname VARCHAR2(20);
BEGIN
    SELECT first_name INTO v_fname FROM employees
    WHERE employee_id=100;
END;
/
```

In the bottom right corner of the code editor, there is a status message: "Task completed in 0.037 seconds". Below the code editor is a 'Script Output' window. It has a toolbar with icons for edit, save, print, and copy. The main area of the window displays the message: "anonymous block completed".

ORACLE

Copyright © 2014, Oracle und/oder verbundene Unternehmen. All rights reserved. Alle Rechte vorbehalten.

Klicken Sie zum Ausführen eines anonymen Blockes auf die Schaltfläche **Run Script** (oder drücken Sie F5).

Hinweis: Im Fenster **Script Output** wird nach Ausführung des Blockes die Meldung "anonymous block completed" angezeigt.

Agenda

- Vorteile und Struktur von PL/SQL kennenlernen
- PL/SQL-Blöcke prüfen
- Ausgabemeldungen in PL/SQL generieren

ORACLE

Copyright © 2014, Oracle und/oder verbundene Unternehmen. All rights reserved. Alle Rechte vorbehalten.

Ausgabe von PL/SQL-Blöcken ermöglichen

- Um die Ausgabe in SQL Developer zu ermöglichen, führen Sie vor der Ausführung des PL/SQL-Blockes folgenden Befehl aus:

```
SET SERVEROUTPUT ON
```

- Verwenden Sie im anonymen Block ein vordefiniertes Oracle-Package und dessen Prozedur:

- DBMS_OUTPUT.PUT_LINE

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

ORACLE

Copyright © 2014, 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 sind Sie für die Ein- und Ausgabe auf vordefinierte Oracle-Packages angewiesen. Um eine Ausgabe zu generieren, gehen Sie wie folgt vor:

- Führen Sie den folgenden Befehl aus:

```
SET SERVEROUTPUT ON
```

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

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

Ausgabe von PL/SQL-Blöcken anzeigen

The screenshot shows the Oracle SQL Developer interface. In the top-left pane, there is a code editor window titled 'Worksheet' containing the following PL/SQL script:

```
SET SERVEROUTPUT ON
DECLARE
    v_fname VARCHAR(20);
BEGIN
    SELECT first_name
    INTO v_fname
    FROM employees
    WHERE employee_id = 100;
    DBMS_OUTPUT.PUT_LINE('The First Name of the Employee is ' || v_fname);
END;
```

A yellow callout box points to the 'Run Script' icon in the toolbar above the code editor, with the text: 'F5 drücken, um den Befehl und PL/SQL-Block auszuführen' (Press F5 to execute the command and PL/SQL block). The bottom-right pane, titled 'Script Output', shows the result of the execution:

```
anonymous block completed
The First Name of the Employee is Steven
```

ORACLE

Copyright © 2014, Oracle und/oder verbundene Unternehmen. All rights reserved. Alle Rechte vorbehalten.

Um die Ausgabe für den PL/SQL-Block anzuzeigen, drücken Sie F5 (oder klicken auf das Symbol **Run Script**). Diese Aktion bewirkt Folgendes:

1. Der Befehl `SET SERVEROUTPUT ON` wird ausgeführt.
2. Der anonyme PL/SQL-Block wird ausgeführt.

Die Ausgabe wird in der Registerkarte **Script Output** angezeigt.

Quiz

Ein PL/SQL-Block *muss* aus den folgenden drei Bereichen bestehen:

- Einem deklarativen Bereich, der mit dem Schlüsselwort `DECLARE` beginnt und mit Beginn des ausführbaren Bereichs endet
- Einem ausführbaren Bereich, der mit dem Schlüsselwort `BEGIN` beginnt und mit `END` endet
- Einem Bereich zur Exception-Behandlung, der mit dem Schlüsselwort `EXCEPTION` beginnt und im ausführbaren Bereich verschachtelt ist
 - a. Richtig
 - b. Falsch

ORACLE

Copyright © 2014, Oracle und/oder verbundene Unternehmen. All rights reserved. Alle Rechte vorbehalten.

Richtige Antwort: b

Ein PL/SQL-Block besteht aus drei Bereichen:

- **Deklarativer Bereich (optional):** Der optionale deklarative Bereich beginnt mit dem Schlüsselwort `DECLARE` und endet mit Beginn des ausführbaren Bereichs.
- **Ausführbarer Bereich (obligatorisch):** Der obligatorische ausführbare Bereich beginnt mit dem Schlüsselwort `BEGIN` und endet mit `END`. Dieser Bereich muss mindestens eine Anweisung enthalten. `END` wird mit einem Semikolon abgeschlossen. Der ausführbare Bereich eines PL/SQL-Blockes kann wiederum beliebig viele weitere PL/SQL-Blöcke enthalten.
- **Bereich zur Exception-Behandlung (optional):** Der optionale Exception-Bereich ist im ausführbaren Bereich verschachtelt. Er beginnt mit dem Schlüsselwort `EXCEPTION`.

Zusammenfassung

In dieser Lektion haben Sie Folgendes gelernt:

- SQL-Anweisungen in PL/SQL-Programmkonstrukte integrieren
- Vorteile von PL/SQL beschreiben
- PL/SQL-Blocktypen unterscheiden
- Ausgabemeldungen in PL/SQL erläutern



Copyright © 2014, Oracle und/oder verbundene Unternehmen. All rights reserved. Alle Rechte vorbehalten.

PL/SQL ist eine Programmiersprache mit Features, die die Funktionalität von SQL erweitern. Die nicht prozedurale Sprache SQL wird durch PL/SQL-Programmierkonstrukte prozedural. PL/SQL-Anwendungen können auf beliebigen Plattformen oder Betriebssystemen ausgeführt werden, auf denen ein Oracle-Server ausgeführt wird. In dieser Lektion haben Sie gelernt, wie einfache PL/SQL-Blöcke erstellt werden.

Übungen zu Lektion 2 – Übersicht

Diese Übung behandelt folgende Themen:

- PL/SQL-Blöcke identifizieren, die erfolgreich ausgeführt werden
- Einfachen PL/SQL-Block erstellen und ausführen



Copyright © 2014, Oracle und/oder verbundene Unternehmen. All rights reserved. Alle Rechte vorbehalten.

Diese Übung vertieft die in dieser Lektion erlernten Grundlagen von PL/SQL.

- Bei der 1. Aufgabe identifizieren Sie PL/SQL-Blöcke, die erfolgreich ausgeführt werden.
- In der 2. Aufgabe erstellen Sie einen einfachen PL/SQL-Block und führen ihn aus.

3

PL/SQL-Variablen deklarieren

ORACLE

Copyright © 2014, Oracle und/oder verbundene Unternehmen. All rights reserved. Alle Rechte vorbehalten.

Ziele

Nach Ablauf dieser Lektion haben Sie folgende Ziele erreicht:

- Gültige und ungültige IDs erkennen
- Verwendungszwecke von Variablen aufzählen
- Variablen deklarieren und initialisieren
- Verschiedene Datentypen auflisten und beschreiben
- Vorteile des Attributs %TYPE identifizieren
- Bind-Variablen deklarieren, verwenden und ausgeben



Copyright © 2014, Oracle und/oder verbundene Unternehmen. All rights reserved. Alle Rechte vorbehalten.

Sie haben bereits einfache PL/SQL-Blöcke und ihre Bereiche kennengelernt. In dieser Lektion werden gültige und ungültige IDs beschrieben. Sie lernen, wie Sie im deklarativen Bereich eines PL/SQL-Blockes Variablen deklarieren und initialisieren. Außerdem werden die verschiedenen Datentypen erläutert. Darüber hinaus lernen Sie das Attribut %TYPE und seine Vorteile kennen.

Agenda

- Variablen – Einführung
- Variable Datentypen und Attribut %TYPE untersuchen
- Bind-Variablen untersuchen

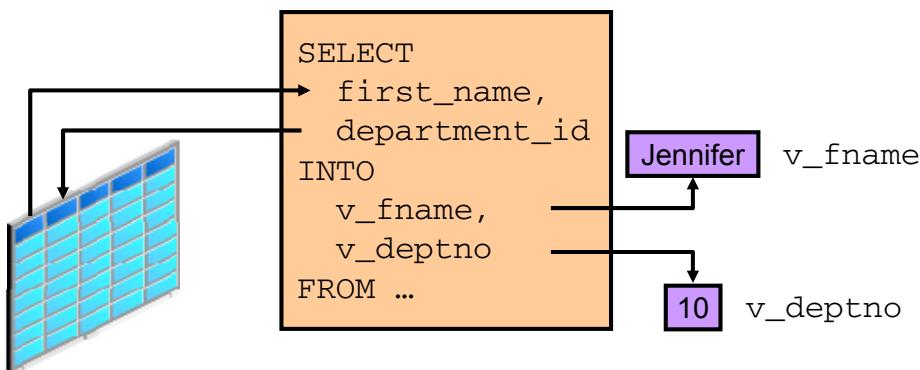
ORACLE

Copyright © 2014, Oracle und/oder verbundene Unternehmen. All rights reserved. Alle Rechte vorbehalten.

Variablen – Einsatzmöglichkeiten

Variablen können für folgende Zwecke verwendet werden:

- Daten temporär speichern
- Gespeicherte Werte bearbeiten
- Wiederverwendbarkeit



ORACLE

Copyright © 2014, Oracle und/oder verbundene Unternehmen. All rights reserved. Alle Rechte vorbehalten.

Mit PL/SQL können Sie Variablen deklarieren und anschließend in SQL-Anweisungen und prozeduralen Anweisungen verwenden.

Variablen werden hauptsächlich verwendet, um Daten zu speichern und gespeicherte Werte zu bearbeiten. Sehen Sie sich die PL/SQL-Anweisung auf der Folie an. Die Anweisung ruft den Vornamen (`first_name`) und die Abteilungsnummer (`department_id`) aus der Tabelle ab. Wenn Sie `first_name` oder `department_id` bearbeiten möchten, müssen Sie den abgerufenen Wert speichern. Für die temporäre Speicherung des Wertes werden Variablen verwendet. Mit dem in diesen Variablen gespeicherten Wert können die Daten ver- und bearbeitet werden. Variablen können beliebige PL/SQL-Objekte wie Variablen, Typen, Cursor und Unterprogramme speichern.

Wiederverwendbarkeit ist ein weiterer Vorteil des Deklarierens von Variablen. Nach ihrer Deklaration können Sie die Variablen in einer Anwendung wiederholt verwenden, indem Sie sie in verschiedenen Anweisungen mehrfach referenzieren.

Variablennamen – Anforderungen

Variablennamen:

- müssen mit einem Buchstaben beginnen
- können Buchstaben oder Zahlen enthalten
- können Sonderzeichen enthalten (wie \$, _ und #)
- dürfen maximal 30 Zeichen lang sein
- dürfen keine reservierten Wörter enthalten



ORACLE

Copyright © 2014, Oracle und/oder verbundene Unternehmen. All rights reserved. Alle Rechte vorbehalten.

Die Regeln zum Benennen von Variablen sind auf der Folie aufgelistet.

Variablen in PL/SQL

Variablen werden:

- im deklarativen Bereich deklariert und (optional) initialisiert
- im ausführbaren Bereich verwendet, wo ihnen neue Werte zugewiesen werden
- als Parameter an PL/SQL-Unterprogramme übergeben
- zum Speichern der Ausgabe von PL/SQL-Unterprogrammen verwendet

ORACLE

Copyright © 2014, Oracle und/oder verbundene Unternehmen. All rights reserved. Alle Rechte vorbehalten.

Sie können Variablen wie folgt verwenden:

- **Im deklarativen Bereich deklarieren und initialisieren:** Sie können Variablen im deklarativen Teil von PL/SQL-Blöcken, Unterprogrammen oder Packages deklarieren. In Deklarationen geben Sie den Speicherplatz, Datentyp und Speicherort für einen Wert an, damit Sie ihn referenzieren können. Deklarationen können zudem einen Ausgangswert zuweisen und das NOT NULL Constraint für die Variable setzen. Vorwärtsreferenzen sind nicht zulässig. Sie müssen eine Variable deklarieren, bevor Sie sie in anderen Anweisungen (auch anderen deklarativen Anweisungen) referenzieren können.
- **Im ausführbaren Bereich verwenden und neue Werte zuweisen:** Sie können den vorhandenen Wert der Variablen im ausführbaren Bereich durch einen neuen Wert ersetzen.
- **Als Parameter an PL/SQL-Unterprogramme übergeben:** Unterprogramme nehmen Parameter an. Sie können Variablen als Parameter an Unterprogramme übergeben.
- **Ausgabe von PL/SQL-Unterprogrammen in Variablen speichern:** Variablen können die von Funktionen zurückgegebenen Werte aufnehmen.

PL/SQL-Variablen deklarieren und initialisieren

Syntax:

```
identifier [CONSTANT] datatype [NOT NULL]
[ := | DEFAULT expr];
```

Beispiele:

```
DECLARE
    v_hiredate      DATE;
    v_location       VARCHAR2(13) := 'Atlanta';
    v_deptno         NUMBER(2) NOT NULL := 10;
    c_comm           CONSTANT NUMBER := 1400;
```

ORACLE

Copyright © 2014, Oracle und/oder verbundene Unternehmen. All rights reserved. Alle Rechte vorbehalten.

Alle PL/SQL-IDs müssen im deklarativen Bereich deklariert sein, damit sie im PL/SQL-Block referenziert werden können. Sie haben die Möglichkeit, einer Variablen einen Ausgangswert zuzuweisen (siehe Folie). Variablen können jedoch auch ohne zugewiesenen Wert deklariert werden. Wenn Sie andere Variablen in einer Deklaration referenzieren, stellen Sie sicher, dass diese in einer vorhergehenden Anweisung bereits separat deklariert wurden.

Für die Syntax gilt:

<i>identifier</i>	Name der Variablen
<i>data type</i>	Skalarer, zusammengesetzter, Referenz- oder LOB-Datentyp (In diesem Kurs werden nur skalare, zusammengesetzte und LOB-Datentypen behandelt.)
CONSTANT	Schränkt die Variable so ein, dass ihr Wert nicht geändert werden kann. (Konstanten müssen initialisiert werden.)
NOT NULL	Schränkt die Variable so ein, dass sie einen Wert enthalten muss (NOT NULL-Variablen müssen initialisiert werden.)
<i>expr</i>	Beliebiger PL/SQL-Ausdruck, kann ein Literal, eine andere Variable oder ein Ausdruck mit Operatoren und Funktionen sein

Hinweis: Neben Variablen können Sie im deklarativen Bereich auch Cursor und Exceptions deklarieren. Informationen zum Deklarieren von Cursoren erhalten Sie in der Lektion "Explizite Cursor". Eine Erläuterung zu Exceptions finden Sie in der Lektion "Exceptions behandeln".

PL/SQL-Variablen deklarieren und initialisieren

1

```
DECLARE
    v_myName  VARCHAR(20);
BEGIN
    DBMS_OUTPUT.PUT_LINE('My name is: ' || v_myName );
    v_myName := 'John';
    DBMS_OUTPUT.PUT_LINE('My name is: ' || v_myName );
END;
/
```

2

```
DECLARE
    v_myName VARCHAR2(20) := 'John';
BEGIN
    v_myName := 'Steven';
    DBMS_OUTPUT.PUT_LINE('My name is: ' || v_myName );
END;
/
```

ORACLE

Copyright © 2014, Oracle und/oder verbundene Unternehmen. All rights reserved. Alle Rechte vorbehalten.

Schauen Sie sich die beiden Codeblöcke auf der Folie an.

1. Im ersten Block wird die Variable `v_myName` deklariert, jedoch nicht initialisiert. Im ausführbaren Bereich wird der Variablen der Wert `John` zugewiesen.
 - Zeichenfolgenliterale müssen in einfache Anführungszeichen gesetzt werden. Wenn die Zeichenfolge Anführungszeichen wie in "Today's Date" enthält, muss sie wie folgt angegeben werden: '`'Today' 's Date'`'.
 - Der Zuweisungsoperator lautet: "`:=`".
 - Die Prozedur `PUT_LINE` rufen Sie auf, indem Sie die Variable `v_myName` übergeben. Der Wert der Variablen ist verkettet mit der Zeichenfolge '`My name is:`'.
 - Ausgabe dieses anonymen Blockes:

```
anonymous block completed
My name is:
My name is: John
```

2. Im zweiten Block wird die Variable `v_myName` im deklarativen Bereich deklariert und initialisiert. Nach der Initialisierung enthält `v_myName` den Wert `John`. Dieser Wert wird im ausführbaren Bereich des Blockes bearbeitet. Ausgabe des anonymen Blockes:

```
anonymous block completed
My name is: Steven
```

Begrenzungszeichen in Zeichenfolgenliteralen

```
DECLARE
    v_event VARCHAR2(15);
BEGIN
    v_event := q'!Father's day!';
    DBMS_OUTPUT.PUT_LINE('3rd Sunday in June is :
    '|| v_event );
    v_event := q'[Mother's day]';
    DBMS_OUTPUT.PUT_LINE('2nd Sunday in May is :
    '|| v_event );
END;
/
```

Resultierende Ausgabe

anonymous block completed
3rd Sunday in June is : Father's day
2nd Sunday in May is : Mother's day

ORACLE

Copyright © 2014, Oracle und/oder verbundene Unternehmen. All rights reserved. Alle Rechte vorbehalten.

Wenn die Zeichenfolge einen Apostroph (ein einzelnes Anführungszeichen) enthält, müssen Sie das Anführungszeichen wie im folgenden Beispiel doppelt eingeben:

```
v_event VARCHAR2(15) := 'Father''s day';
```

Das erste Anführungszeichen dient als Escape-Zeichen. Dies macht die Zeichenfolge komplex, insbesondere wenn Sie SQL-Anweisungen als Zeichenfolgen verwenden. Die Folie zeigt, wie Sie das Begrenzungszeichen mit der Notation `q'` angeben. Sie können damit jedes Zeichen, das nicht in der Zeichenfolge enthalten ist, als Begrenzungszeichen angeben. Im Beispiel werden Ausrufezeichen (!) und eckige Klammern ([]) als Begrenzungszeichen verwendet. Betrachten Sie das folgende Beispiel:

```
v_event := q'!Father's day!';
```

Vergleichen Sie dieses Beispiel mit dem ersten Beispiel auf der Seite. Sie beginnen die Zeichenfolge mit `q'`, wenn Sie ein Begrenzungszeichen verwenden möchten. Das Zeichen nach der Notation wird als Begrenzungszeichen verwendet. Nach dem Begrenzungszeichen geben Sie die Zeichenfolge und das abschließende Begrenzungszeichen ein und schließen die Notation mit einem einfachen Anführungszeichen. Das folgende Beispiel zeigt, wie Sie eckige Klammern ([]) als Begrenzungszeichen verwenden:

```
v_event := q'[Mother's day]';
```

Agenda

- Variablen – Einführung
- Variable Datentypen und Attribut %TYPE untersuchen
- Bind-Variablen untersuchen



Copyright © 2014, Oracle und/oder verbundene Unternehmen. All rights reserved. Alle Rechte vorbehalten.

Variablenarten

- PL/SQL-Variablen:
 - Skalare Variablen
 - Referenzvariablen
 - Large Object-(LOB-)Variablen
 - Zusammengesetzte Variablen
- Nicht-PL/SQL-Variablen: Bind-Variablen

ORACLE

Copyright © 2014, Oracle und/oder verbundene Unternehmen. All rights reserved. Alle Rechte vorbehalten.

Jede PL/SQL-Variable gehört einem Datentyp an, der ein Speicherformat, Constraints und einen gültigen Wertebereich festlegt. PL/SQL unterstützt verschiedene Kategorien von Datentypen wie skalare Datentypen, Referenzdatentypen, Large Object-(LOB-)Datentypen und zusammengesetzte Datentypen.

- **Skalare Datentypen:** Skalare Datentypen speichern einen einzelnen Wert. Der Wert hängt vom Datentyp der Variablen ab. Beispiel: Die Variable `v_myName`, die im Abschnitt "PL/SQL-Variablen deklarieren und initialisieren" dieser Lektion als Beispiel verwendet wird, weist den Typ `VARCHAR2` auf. Sie kann daher einen Zeichenfolgenwert speichern. PL/SQL unterstützt auch boolesche Variablen.
- **Referenzdatentypen:** Referenzdatentypen speichern Werte, die als *Zeiger* bezeichnet werden und auf einen Speicherort zeigen.
- **LOB-Datentypen:** LOB-Datentypen speichern Werte, die als *Positionsanzeiger* bezeichnet werden. Diese Werte geben den Speicherort von Large Objects (zum Beispiel Grafiken) an, die außerhalb der Tabelle gespeichert sind.
- **Zusammengesetzte Datentypen:** Zusammengesetzte Datentypen stehen bei Verwendung von PL/SQL-Collection- und Record-Variablen zur Verfügung. Diese enthalten interne Elemente, die Sie wie individuelle Variablen behandeln können.

Zu den Nicht-PL/SQL-Variablen gehören in Precompiler-Programmen deklarierte Variablen der Hostsprache, Maskenfelder in Forms-Anwendungen sowie Hostvariablen. Im weiteren Verlauf dieser Lektion werden die Hostvariablen näher erläutert.

Weitere Informationen zu LOBs finden Sie im Dokument *PL/SQL User's Guide and Reference*.

Variablenarten



ORACLE

Copyright © 2014, Oracle und/oder verbundene Unternehmen. All rights reserved. Alle Rechte vorbehalten.

Die Folie enthält Beispiele für die folgenden Datentypen:

- "TRUE" steht für einen booleschen Wert.
- "15-JAN-09" stellt den Datentyp DATE dar.
- Das Bild steht für ein BLOB.
- Der Text in der Sprechblase steht für den Datentyp VARCHAR2 oder CLOB.
- "256120.08" stellt den Datentyp NUMBER mit Dezimalstellen dar.
- Die Filmrolle steht für ein BFILE.
- Der Ortsname *Atlanta* steht für den Datentyp VARCHAR2.

PL/SQL-Variablen deklarieren und initialisieren – Richtlinien

- Benennungskonventionen einhalten
- Aussagekräftige IDs für Variablen verwenden
- Variablen initialisieren, die als NOT NULL und CONSTANT definiert sind
- Variablen mit dem Zuweisungsoperator (:=) oder dem Schlüsselwort DEFAULT initialisieren:

```
v_myName VARCHAR2(20) := 'John' ;
```

```
v_myName VARCHAR2(20) DEFAULT 'John' ;
```

- Zur besseren Lesbarkeit und Codeverwaltung nur eine ID pro Zeile deklarieren

ORACLE

Copyright © 2014, Oracle und/oder verbundene Unternehmen. All rights reserved. Alle Rechte vorbehalten.

Beachten Sie beim Deklarieren von PL/SQL-Variablen folgende Richtlinien:

- Halten Sie konsistente Benennungskonventionen ein. Verwenden Sie beispielsweise name für Variablen und c_name für Konstanten. Analog können Sie Variablen mit v_fname benennen. Wichtig ist, dass Sie bei Ihrer Benennungskonvention auf Konsistenz achten, damit sich die Elemente leichter identifizieren lassen.
- Verwenden Sie aussagekräftige und geeignete IDs für Variablen. Wählen Sie beispielsweise salary und sal_with_commission anstelle von salary1 und salary2.
- Wenn Sie das NOT NULL Constraint verwenden, müssen Sie beim Deklarieren der Variablen einen Wert zuweisen.
- Beim Deklarieren von Konstanten muss der Typspezifikation das Schlüsselwort CONSTANT vorangestellt werden. Die folgende Deklaration benennt eine Konstante vom Typ NUMBER und weist der Konstanten den Wert "50.000" zu. Konstanten müssen bei ihrer Deklaration initialisiert werden, da sonst ein Kompilierungsfehler auftritt. Nach der Initialisierung lässt sich der Wert einer Konstanten nicht mehr ändern.

```
sal CONSTANT NUMBER := 50000.00;
```

PL/SQL-Variablen deklarieren – Richtlinien

- Keine Spaltennamen als IDs verwenden

```
DECLARE
    employee_id  NUMBER(6);
BEGIN
    SELECT employee_id
    INTO   employee_id
    FROM   employees
    WHERE      last_name = 'Kochhar';
END;
/
```

A yellow triangular warning sign with a red exclamation mark in the center. It is positioned to the right of the code snippet, specifically pointing towards the two occurrences of 'employee_id' in the SELECT statement.

- Constraint NOT NULL verwenden, wenn die Variable einen Wert speichern muss

ORACLE

Copyright © 2014, Oracle und/oder verbundene Unternehmen. All rights reserved. Alle Rechte vorbehalten.

- Initialisieren Sie die Variable mit einem Ausdruck. Verwenden Sie dafür den Zuweisungsoperator (:=) oder das reservierte Wort DEFAULT. Wenn Sie keinen Ausgangswert zuweisen, enthält die neue Variable standardmäßig NULL, bis Sie einen Wert zuweisen. Um einer Variablen einen Wert (erneut) zuzuweisen, erstellen Sie eine PL/SQL-Zuweisungsanweisung. Allerdings ist das Initialisieren aller Variablen eine empfohlene Vorgehensweise bei der Programmierung.
- Gleiche Objektnamen sind zulässig, solange die Objekte in unterschiedlichen Blöcken definiert sind. Objekte mit gleichen Objektnamen können Sie mit Labels kennzeichnen.
- Vermeiden Sie die Verwendung von Spaltennamen als IDs. Wenn PL/SQL-Variablen in SQL-Anweisungen denselben Namen wie eine Spalte tragen, geht der Oracle-Server davon aus, dass die Spalte referenziert wird. Obwohl das Codebeispiel auf der Folie funktioniert, ist Code, der für Datenbanktabellen und Variablen denselben Namen enthält, schwer zu lesen und zu verwalten.
- Verwenden Sie das NOT NULL Constraint, wenn die Variable einen Wert enthalten muss. Sie können einer als NOT NULL definierten Variablen keine NULL-Werte zuweisen. Auf das NOT NULL Constraint muss eine Initialisierungsklausel folgen.

```
pincode VARCHAR2(15) NOT NULL := 'Oxford';
```

In diesem Kurs verwendete PL/SQL-Strukturen – Benennungskonventionen

PL/SQL-Struktur	Konvention	Beispiel
Variable	v_variable_name	v_rate
Konstante	c_constant_name	c_rate
Parameter eines Unterprogramms	p_parameter_name	p_id
Bind-Variable (Hostvariable)	b_bind_name	b_salary
Cursor	cur_cursor_name	cur_emp
Record	rec_record_name	rec_emp
Typ	type_name_type	ename_table_type
Exception	e_exception_name	e_products_invalid
Datei-Handle	f_file_handle_name	f_file

ORACLE

Copyright © 2014, 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.

Skalare Datentypen

- Speichern einen einzelnen Wert
- Besitzen keine internen Komponenten

TRUE

15-JAN-09

256120.08

Atlanta

Das Verlangen des Faulen
regt sich vergebens,
das Verlangen der Fleißigen
wird befriedigt.

ORACLE

Copyright © 2014, Oracle und/oder verbundene Unternehmen. All rights reserved. Alle Rechte vorbehalten.

PL/SQL bietet eine Reihe vordefinierter Datentypen. Zur Auswahl stehen unter anderem: INTEGER, FLOATING POINT, CHARACTER, BOOLEAN, DATE, COLLECTION und LOB. In dieser Lektion werden die grundlegenden Typen vorgestellt, die in PL/SQL-Programmen häufig verwendet werden.

Skalare Datentypen speichern einen einzelnen Wert und besitzen keine internen Komponenten. Sie können in vier Kategorien klassifiziert werden: NUMBER, CHARACTER, DATE und BOOLEAN. Die Datentypen CHARACTER und NUMBER besitzen Subtypen, die einem Constraint einen Basistyp zuweisen. Die Typen INTEGER und POSITIVE sind beispielsweise Subtypen des Basistyps NUMBER.

Weitere Informationen zu skalaren Datentypen (sowie eine vollständige Liste) finden Sie im Dokument *PL/SQL User's Guide and Reference*.

Skalare Basisdatentypen

- CHAR [(maximum_length)]
- VARCHAR2 (maximum_length)
- NUMBER [(precision, scale)]
- BINARY_INTEGER
- PLS_INTEGER
- BOOLEAN
- BINARY_FLOAT
- BINARY_DOUBLE

ORACLE

Copyright © 2014, Oracle und/oder verbundene Unternehmen. All rights reserved. Alle Rechte vorbehalten.

Datentyp	Beschreibung
CHAR [(maximum_length)]	Basistyp für Daten mit Zeichenfolgen fester Länge bis 32.767 Byte. Wenn Sie keine maximale Länge angeben, wird als Standardlänge 1 Byte festgelegt.
VARCHAR2 (maximum_length)	Basistyp für Daten mit Zeichenfolgen variabler Länge bis 32.767 Byte. Für VARCHAR2-Variablen und -Konstanten gibt es keine Standardgröße.
NUMBER [(precision, scale)]	Ziffern mit einer festgelegten Anzahl von Nachkommastellen (scale) und Gesamtstellen (precision). Die Anzahl der Gesamtstellen kann zwischen 1 und 38 betragen. Der gültige Wertebereich für die Nachkommastellenzahl ist -84 bis 127.
BINARY_INTEGER	Basistyp für Ganzzahlen zwischen -2.147.483.647 und 2.147.483.647

Datentyp	Beschreibung
PLS_INTEGER	Basistyp für Ganzzahlen mit Vorzeichen zwischen -2.147.483.648 und 2.147.483.647. PLS_INTEGER-Werte erfordern weniger Speicherplatz und lassen sich schneller verarbeiten als NUMBER-Werte. In Oracle Database 11g und Oracle Database 12c sind die Datentypen PLS_INTEGER und BINARY_INTEGER identisch. Die arithmetischen Vorgänge für PLS_INTEGER- und BINARY_INTEGER-Werte sind schneller als für NUMBER-Werte.
BOOLEAN	Basistyp, der einen der drei möglichen Werte für logische Berechnungen speichert: TRUE, FALSE und NULL
BINARY_FLOAT	Stellt Gleitkommazahlen im IEEE 754-Format dar. Zum Speichern des Wertes sind 5 Byte erforderlich.
BINARY_DOUBLE	Stellt Gleitkommazahlen im IEEE 754-Format dar. Zum Speichern des Wertes sind 9 Byte erforderlich.

Skalare Basisdatentypen

- DATE
- TIMESTAMP
- TIMESTAMP WITH TIME ZONE
- TIMESTAMP WITH LOCAL TIME ZONE
- INTERVAL YEAR TO MONTH
- INTERVAL DAY TO SECOND

ORACLE

Copyright © 2014, Oracle und/oder verbundene Unternehmen. All rights reserved. Alle Rechte vorbehalten.

Datentyp	Beschreibung
DATE	Basistyp für Datum und Uhrzeit. DATE-Werte beinhalten die am aktuellen Tag seit Mitternacht verstrichene Zeit in Sekunden. Der Datumsbereich liegt zwischen 4712 v. Chr. und 9999 n. Chr.
TIMESTAMP	Der Datentyp TIMESTAMP, eine Erweiterung von DATE, speichert Jahr, Monat, Tag, Stunde, Minute, Sekunde und Sekundenbruchteil. Die Syntax lautet: TIMESTAMP [(precision)], wobei der optionale Parameter precision die Stellenanzahl der Sekundenbruchteile angibt (ganze Zahl zwischen 0 und 9). Die Standardeinstellung lautet 6.
TIMESTAMP WITH TIME ZONE	Der Datentyp TIMESTAMP WITH TIME ZONE, eine Erweiterung von TIMESTAMP, beinhaltet eine Zeitzonenverschiebung. Die Zeitzonenverschiebung ist die Differenz (in Stunden und Minuten) zwischen der lokalen und der UTC-Zeit (Universal Time Coordinate), früher bekannt als Greenwich Mean Time. Die Syntax lautet TIMESTAMP [(precision)] WITH TIME ZONE, wobei der optionale Parameter precision die Stellenanzahl der Sekundenbruchteile angibt (ganze Zahl zwischen 0 und 9). Die Standardeinstellung lautet 6.

Datentyp	Beschreibung
TIMESTAMP WITH LOCAL TIME ZONE	<p>Der Datentyp TIMESTAMP WITH LOCAL TIME ZONE, eine Erweiterung von TIMESTAMP, beinhaltet eine Zeitzonenverschiebung. Die Zeitzonenverschiebung ist die Differenz (in Stunden und Minuten) zwischen der lokalen und der UTC-Zeit (Universal Time Coordinate), früher bekannt als Greenwich Mean Time. Die Syntax lautet <code>TIMESTAMP [(precision)] WITH LOCAL TIME ZONE</code>, wobei der optionale Parameter <code>precision</code> die Stellenanzahl der Sekundenbruchteile angibt (ganze Zahl zwischen 0 und 9). Für die Gesamtstellenanzahl muss ein Ganzzahlliteral zwischen 0 und 9 angegeben werden. Die Angabe einer symbolischen Konstanten oder Variablen ist nicht möglich. Die Standardeinstellung lautet 6.</p> <p>Der Unterschied zum Datentyp TIMESTAMP WITH TIME ZONE liegt darin, dass in Datenbankspalten eingegebene Zeitzonenwerte der Datenbank normalisiert werden und die Zeitzonendifferenz nicht in der Spalte gespeichert wird. Wenn Sie den Wert abrufen, gibt der Oracle-Server den Wert in der Zeitzone Ihrer lokalen Session zurück.</p>
INTERVAL YEAR TO MONTH	<p>Mit dem Datentyp INTERVAL YEAR TO MONTH speichern und bearbeiten Sie Zeiträume, die Jahre oder Monate umfassen. Die Syntax lautet <code>INTERVAL YEAR [(precision)] TO MONTH</code>, wobei <code>precision</code> die Stellenanzahl im Feld für das Jahr angibt. Für die Gesamtstellenanzahl muss ein Ganzzahlliteral zwischen 0 und 4 angegeben werden. Die Angabe einer symbolischen Konstanten oder Variablen ist nicht möglich.</p> <p>Die Standardeinstellung lautet 2.</p>
INTERVAL DAY TO SECOND	<p>Mit dem Datentyp INTERVAL DAY TO SECOND speichern und bearbeiten Sie Zeiträume, die Tage, Stunden, Minuten und Sekunden umfassen. Die Syntax lautet <code>INTERVAL DAY [(precision1)] TO SECOND [(precision2)]</code>, wobei <code>precision</code> und <code>precision2</code> die Stellenanzahl im Feld für das Jahr beziehungsweise für die Sekunden angibt. Für die Gesamtstellenanzahl muss in beiden Fällen ein Ganzzahlliteral zwischen 0 und 9 angegeben werden. Die Angabe einer symbolischen Konstanten oder Variablen ist nicht möglich. Die Standardeinstellungen lauten 2 und 6.</p>

Skalare Variablen deklarieren

Beispiele:

```
DECLARE
    v_emp_job          VARCHAR2(9);
    v_count_loop       BINARY_INTEGER := 0;
    v_dept_total_sal  NUMBER(9,2) := 0;
    v_orderdate        DATE := SYSDATE + 7;
    c_tax_rate         CONSTANT NUMBER(3,2) := 8.25;
    v_valid            BOOLEAN NOT NULL := TRUE;
    ...
```

ORACLE

Copyright © 2014, Oracle und/oder verbundene Unternehmen. All rights reserved. Alle Rechte vorbehalten.

Die Beispiele für die Variablen-deklaration auf der Folie sind wie folgt definiert.

- **v_emp_job:** Variable, die die Positionsbezeichnung eines Mitarbeiters speichert
- **v_count_loop:** Variable, die die Iterationen einer Schleife zählt. Anfangswert: 0
- **v_dept_total_sal:** Variable, die die Gesamtsumme der Gehälter einer Abteilung berechnet. Anfangswert: 0
- **v_orderdate:** Variable, die das Lieferdatum eines Auftrags speichert. Anfangswert: Eine Woche nach dem aktuellen Datum.
- **c_tax_rate:** Konstante für den Steuersatz, der sich im gesamten PL/SQL-Block nie ändert. Einstellung: 8 . 25
- **v_valid:** Flag, das angibt, ob eine Dateneingabe gültig oder ungültig ist. Anfangswert: TRUE

Attribut %TYPE

- Dient zum Deklarieren einer Variablen entsprechend:
 - der Definition einer Datenbankspalte
 - einer anderen deklarierten Variablen
- Erhält als Präfix:
 - den Namen der Datenbanktabelle und -spalte
 - den Namen der deklarierten Variablen

ORACLE

Copyright © 2014, Oracle und/oder verbundene Unternehmen. All rights reserved. Alle Rechte vorbehalten.

PL/SQL-Variablen sind in der Regel so deklariert, dass sie in einer Datenbank gespeicherte Daten aufnehmen und bearbeiten. Wenn Sie PL/SQL-Variablen zum Speichern von Spaltenwerten deklarieren, müssen Sie sicherstellen, dass die Variablen den richtigen Datentyp und die richtige Gesamtstelligenzahl aufweisen. Ist dies nicht der Fall, tritt während der Ausführung ein PL/SQL-Fehler auf. Wenn Sie große Unterprogramme entwickeln müssen, kann dies zeitaufwändig sein und leicht zu Fehlern führen.

Anstatt den Datentyp und die Gesamtstelligenzahl von Variablen hartzudcodieren, können Sie Variablen mithilfe des Attributs %TYPE entsprechend einer zuvor deklarierten Variablen oder Datenbankspalte deklarieren. Das Attribut %TYPE wird meist verwendet, wenn der in der Variablen gespeicherte Wert von einer Tabelle in der Datenbank abgeleitet wird. Wenn Sie eine Variable mithilfe des Attributs %TYPE deklarieren, müssen Sie ihm den Namen der Datenbanktabelle und der Spalte voranstellen. Wenn Sie eine zuvor deklarierte Variable referenzieren, stellen Sie der zu deklarierenden Variablen den Variablennamen der zuvor deklarierten Variablen voran.

Attribut %TYPE – Vorteile

- Sie können Fehler vermeiden, die durch nicht übereinstimmende Datentypen oder falsche Gesamtstelzenzahlen verursacht werden.
- Die Hartcodierung des Datentyps einer Variablen lässt sich umgehen.
- Sie können die Variablen Deklaration beibehalten, wenn sich die Spaltendefinition ändert. Der PL/SQL-Block kann Fehler ausgeben, wenn eine Tabellenspalte geändert wird, für die Sie bereits eine Variable ohne das Attribut %TYPE deklariert haben. Wenn Sie das Attribut %TYPE verwenden, ermittelt PL/SQL den Datentyp und die Größe der Variablen bei der Komplilierung des Blockes. Dies gewährleistet, dass eine solche Variable immer mit der Spalte kompatibel ist, die zum Auffüllen der Variablen verwendet wird.

Variablen mit dem Attribut %TYPE deklarieren

Syntax:

```
identifier      table.column_name%TYPE;
```

Beispiele:

```
...
  v_emp_lname      employees.last_name%TYPE;
...
```

```
...
  v_balance        NUMBER(7,2);
  v_min_balance   v_balance%TYPE := 1000;
...
```

ORACLE

Copyright © 2014, Oracle und/oder verbundene Unternehmen. All rights reserved. Alle Rechte vorbehalten.

Deklarieren Sie die Variablen zum Speichern des Nachnamens eines Mitarbeiters. Die Variable `v_emp_lname` ist mit demselben Datentyp wie die Spalte `last_name` in der Tabelle `employees` definiert. Das Attribut `%TYPE` gibt den Datentyp einer Datenbankspalte an.

Deklarieren Sie Variablen so, dass sie den Kontostand eines Bankkontos sowie das Mindestguthaben von 1.000 speichern. Die Variable `v_min_balance` wird so definiert, dass sie denselben Datentyp wie die Variable `v_balance` hat. Das Attribut `%TYPE` gibt den Datentyp einer Variablen an.

Das NOT NULL Constraint für Datenbankspalten gilt nicht für Variablen, die mit `%TYPE` deklariert wurden. Daher können Sie einer Variablen, die Sie mit dem Attribut `%TYPE` deklariert haben und die eine als NOT NULL definierte Datenbankspalte verwendet, den Wert NULL zuordnen.

Boolesche Variablen deklarieren

- Einer booleschen Variablen können nur die Werte TRUE, FALSE und NULL zugewiesen werden.
- Bedingungsausdrücke verwenden die logischen Operatoren AND und OR sowie den unären Operator NOT, um die Variablenwerte zu prüfen.
- Die Variablen ergeben stets TRUE, FALSE oder NULL.
- Arithmetische Ausdrücke, Zeichenfolgen- und Datumsausdrücke können boolesche Werte zurückgeben.

ORACLE®

Copyright © 2014, Oracle und/oder verbundene Unternehmen. All rights reserved. Alle Rechte vorbehalten.

Mit PL/SQL können Sie Variablen in SQL-Anweisungen und prozeduralen Anweisungen vergleichen. Diese Vergleiche, die als boolesche Ausdrücke bezeichnet werden, bestehen aus einfachen oder komplexen Ausdrücken, die durch relationale Operatoren getrennt sind. In einer SQL-Anweisung können Sie mithilfe von booleschen Ausdrücken die Zeilen in einer Tabelle angeben, die von der Anweisung betroffen sind. In prozeduralen Anweisungen sind boolesche Ausdrücke die Basis für IF-Anweisungen. NULL steht für einen fehlenden, nicht anwendbaren oder unbekannten Wert.

Beispiele

```
emp_sal1 := 50000;
emp_sal2 := 60000;
```

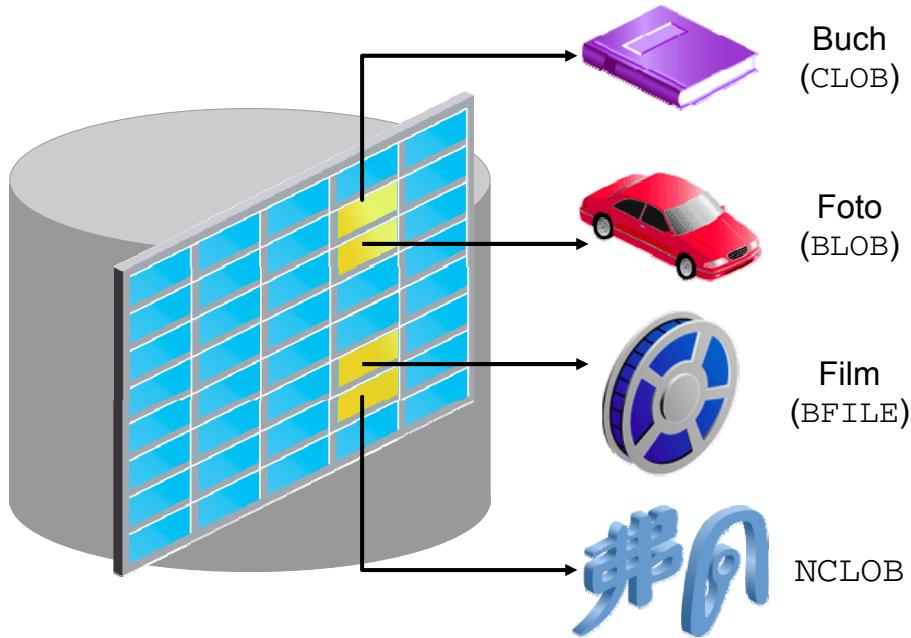
Der folgende Ausdruck ergibt TRUE:

```
emp_sal1 < emp_sal2
```

Boolesche Variable deklarieren und initialisieren:

```
DECLARE
    flag BOOLEAN := FALSE;
BEGIN
    flag := TRUE;
END;
/
```

Variablen mit LOB-Datentypen



ORACLE

Copyright © 2014, Oracle und/oder verbundene Unternehmen. All rights reserved. Alle Rechte vorbehalten.

Large Objects (LOBs) dienen zum Speichern großer Datenmengen. Datenbankspalten können der Kategorie LOB angehören. In dieser Kategorie der Datentypen (BLOB, CLOB und so weiter) können Sie Blöcke unstrukturierter Daten (zum Beispiel Text, Grafiken, Videoclips und Sounddateien) speichern, die je nach Größe des Datenbankblockes bis zu 128 Terabyte umfassen können. LOB-Datentypen ermöglichen den effizienten, direkten und schrittweisen Zugriff auf die Daten und können Attribute eines Objekttyps sein.

- Mit dem Datentyp **CLOB** (Character Large Object) speichern Sie große Blöcke von Zeichendaten in der Datenbank.
- Mit dem Datentyp **BLOB** (Binary Large Object) speichern Sie große unstrukturierte und strukturierte binäre Objekte in der Datenbank. Wenn Sie Daten dieser Art in die Datenbank einfügen oder daraus abrufen, interpretiert die Datenbank die Daten nicht. Sie müssen von den externen Anwendungen interpretiert werden, die die Daten verwenden.
- Mit dem Datentyp **BFILE** (Binärdatei) können Sie große Binärdateien speichern. Im Gegensatz zu anderen LOBs werden **BFILES** außerhalb der Datenbank gespeichert. **BFILES** können Betriebssystemdateien sein. In der Datenbank wird nur ein Zeiger zum **BFILE** gespeichert.
- Mit dem Datentyp **NCLOB** (National Language Character Large Object) können Sie große Blöcke aus **NCHAR**-Unicode-Daten in Ein-Byte- oder nicht proportionalen Mehr-Byte-Zeichensätzen in der Datenbank speichern.

Zusammengesetzte Datentypen – Records und Collections

PL/SQL-Record:

TRUE	23-DEC-98	ATLANTA	
------	-----------	---------	---

PL/SQL-Collections:

1	SMITH	1	5000
2	JONES	2	2345
3	NANCY	3	12
4	TIM	4	3456

PLS_INTEGER VARCHAR2

PLS_INTEGER NUMBER

ORACLE

Copyright © 2014, Oracle und/oder verbundene Unternehmen. All rights reserved. Alle Rechte vorbehalten.

Wie bereits erwähnt, speichern skalare Datentypen einen einzelnen Wert und besitzen keine internen Komponenten. Zusammengesetzte Datentypen (sogenannte PL/SQL-Records und PL/SQL-Collections) verfügen über interne Komponenten, die sich wie individuelle Variablen handhaben lassen.

- In einem PL/SQL-Record werden die internen Komponenten als Felder bezeichnet und können unterschiedliche Datentypen annehmen. Der Zugriff auf die einzelnen Felder erfolgt mit dieser Syntax: `record_name.field_name`. Eine Record-Variablen kann eine Tabellenzeile oder einige Spalten aus einer Tabellenzeile aufnehmen. Jedes Record-Feld entspricht einer Tabellenspalte.
- In einer PL/SQL-Collection werden die internen Komponenten Elemente genannt. Sie weisen stets denselben Datentyp auf. Der Zugriff auf die einzelnen Elemente erfolgt über eindeutige untergeordnete Skripte. Listen und Arrays sind klassische Beispiele von Collections. Es gibt drei Typen von PL/SQL-Collections: Assoziative Arrays, Nested Tables und VARRAY.

Hinweise

- PL/SQL-Records und assoziative Arrays werden in der Lektion "Mit zusammengesetzten Datentypen arbeiten" behandelt.
- Die Datentypen NESTED TABLE und VARRAY kommen im Kurs Advanced PL/SQL zur Sprache.

Agenda

- Variablen – Einführung
- Variable Datentypen und Attribut %TYPE untersuchen
- Bind-Variablen untersuchen

ORACLE

Copyright © 2014, Oracle und/oder verbundene Unternehmen. All rights reserved. Alle Rechte vorbehalten.

Bind-Variablen

Bind-Variablen werden:

- in der Umgebung erstellt
- auch als *Hostvariablen* bezeichnet
- mit dem Schlüsselwort VARIABLE erstellt*
- in SQL-Anweisungen und PL/SQL-Blöcken verwendet
- sogar nach Ausführung des PL/SQL-Blockes aufgerufen
- mit einem vorangestellten Doppelpunkt referenziert

Werte können mit dem Befehl PRINT ausgegeben werden.

* Bei Verwendung von SQL*Plus und SQL Developer obligatorisch

ORACLE

Copyright © 2014, Oracle und/oder verbundene Unternehmen. All rights reserved. Alle Rechte vorbehalten.

Bind-Variablen werden in einer Hostumgebung erstellt. Aus diesem Grund werden sie mitunter als *Hostvariablen* bezeichnet.

Verwendungszwecke von Bind-Variablen

Bind-Variablen werden in der Umgebung und nicht im deklarativen Bereich eines PL/SQL-Blockes erstellt. Daher ist der Zugriff auf Bind-Variablen auch möglich, nachdem der Block ausgeführt wurde. Erstellte Bind-Variablen können von mehreren Unterprogrammen verwendet und bearbeitet werden. Sie sind wie jede andere Variable in SQL-Anweisungen und PL/SQL-Blöcken verwendbar. Die Variablen können als Laufzeitwerte an oder von PL/SQL-Unterprogrammen übergeben werden.

Hinweis: Eine Bind-Variable keine globale Variable, sondern eine Umgebungsvariable.

Bind-Variablen erstellen

Um eine Bind-Variable in SQL Developer zu erstellen, verwenden Sie den Befehl VARIABLE. Deklarieren Sie zum Beispiel Variablen vom Typ NUMBER und VARCHAR2 wie folgt:

```
VARIABLE return_code NUMBER  
VARIABLE return_msg VARCHAR2(30)
```

Werte in Bind-Variablen anzeigen

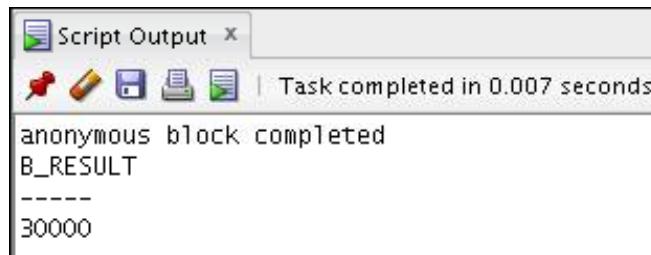
Sie können die Bind-Variable mithilfe von SQL Developer referenzieren und ihren Wert mit dem Befehl PRINT anzeigen.

Beispiel

Eine Bind-Variable kann in einem PL/SQL-Programm durch einen vorangestellten Doppelpunkt referenziert werden.

Der folgende PL/SQL-Block erstellt und verwendet beispielsweise die Bind-Variable `b_result`. Die Ausgabe des Befehls `PRINT` ist unterhalb des Codes abgebildet.

```
VARIABLE b_result NUMBER
BEGIN
    SELECT (SALARY*12) + NVL(COMMISSION_PCT,0) INTO :b_result
    FROM employees WHERE employee_id = 144;
END;
/
PRINT b_result
```



The screenshot shows the 'Script Output' window from Oracle SQL Developer. The title bar says 'Script Output'. Below it is a toolbar with icons for Run, Stop, Save, Print, and Paste. A status message 'Task completed in 0.007 seconds' is displayed. The main pane contains the output of the anonymous block. It starts with 'anonymous block completed', followed by 'B_RESULT' and a dashed line separator, then the value '30000'.

```
anonymous block completed
B_RESULT
-----
30000
```

Hinweis: Wenn Sie eine Bind-Variable vom Typ `NUMBER` erstellen, können Sie nicht die Gesamtstelligenzahl und die Anzahl der Nachkommastellen angeben. Allerdings lässt sich die Länge von Zeichenfolgen angeben. Oracle-Variablen vom Typ `NUMBER` werden ungeachtet der Dimension auf die gleiche Weise gespeichert. Der Oracle-Server verwendet zum Speichern der Werte 7, 70 und 0,0734 dieselbe Anzahl von Byte. Die Längenberechnung der Oracle-Zahlendarstellung anhand des Zahlenformats ist unpraktisch, weshalb der Code immer die erforderliche Anzahl von Byte zuweist. Bei Zeichenfolgen muss der Benutzer die Länge angeben, damit die erforderliche Anzahl von Byte zugewiesen werden kann.

Bind-Variablen referenzieren

Beispiel:

```
VARIABLE b_emp_salary NUMBER
BEGIN
    SELECT salary INTO :b_emp_salary
    FROM employees WHERE employee_id = 178;
END;
/
PRINT b_emp_salary
SELECT first_name, last_name
FROM employees
WHERE salary=:b_emp_salary;
```

Ausgabe →

Script Output	
	Task completed in 0.013 seconds
anonymous block completed	
B_EMP_SALARY	

7000	
FIRST_NAME	LAST_NAME
-----	-----
Oliver	Tuvault
Sarath	Sewall
Kimberely	Grant

ORACLE

Copyright © 2014, Oracle und/oder verbundene Unternehmen. All rights reserved. Alle Rechte vorbehalten.

Wie bereits erwähnt, können Sie eine Bind-Variable nach ihrer Erstellung in jeder anderen SQL-Anweisung oder in beliebigen PL/SQL-Programmen referenzieren.

Im Beispiel wird `b_emp_salary` als Bind-Variable im PL/SQL-Block erstellt. Anschließend wird sie in der folgenden SELECT-Anweisung verwendet.

Die Ausführung des auf der Folie gezeigten PL/SQL-Blockes ergibt die folgende Ausgabe:

- Der Befehl `PRINT` wird ausgeführt:

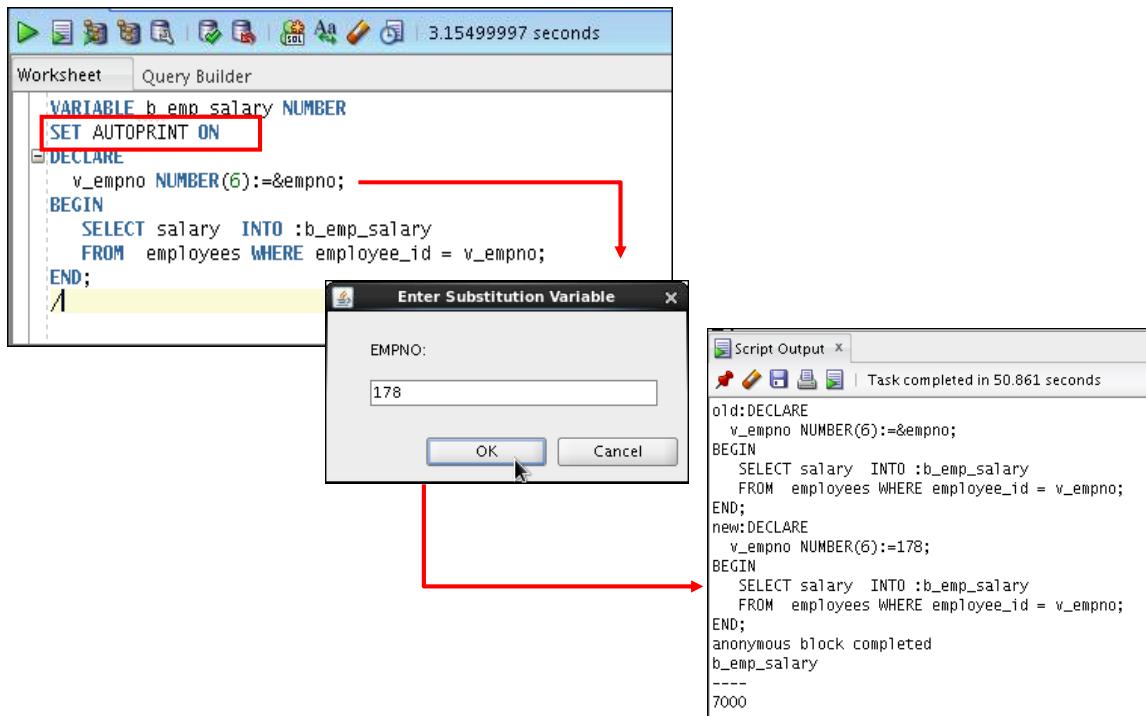
```
b_emp_salary
-----
7000
```

- Dann folgt die Ausgabe der SQL-Anweisung:

```
FIRST_NAME      LAST_NAME
-----          -----
Oliver          Tuvault
Sarah           Sewall
Kimberely       Grant
```

Hinweis: Um alle Bind-Variablen anzuzeigen, setzen Sie den Befehl `PRINT` ohne Variablen ab.

AUTOPRINT mit Bind-Variablen



ORACLE

Copyright © 2014, Oracle und/oder verbundene Unternehmen. All rights reserved. Alle Rechte vorbehalten.

Mit dem Befehl `SET AUTOPRINT ON` können Sie die in einem erfolgreichen PL/SQL-Block verwendeten Bind-Variablen automatisch anzeigen.

Beispiel

Im Codebeispiel gilt:

- Die Bind-Variable `b_emp_salary` wird erstellt, und `AUTOPRINT` wird aktiviert.
- Die Variable `v_empno` wird deklariert, und eine Substitutionsvariable empfängt die Benutzereingabe.
- Schließlich kommen die Bind-Variable und temporäre Variablen im ausführbaren Bereich des PL/SQL-Blockes zum Einsatz.

Wenn Sie eine gültige Personalnummer eingeben (in diesem Fall 178), wird die Ausgabe der Bind-Variable automatisch angezeigt. Die Bind-Variable enthält das Gehalt für die vom Benutzer angegebene Personalnummer.

Quiz

Das Attribut %TYPE:

- a. dient der Deklaration einer Variablen entsprechend der Definition einer Datenbankspalte
- b. dient der Deklaration einer Variablen entsprechend einer Collection von Spalten in einer Datenbanktabelle oder View
- c. dient der Deklaration einer Variablen entsprechend der Definition einer anderen deklarierten Variablen
- d. erhält als Präfix den Namen der Datenbanktabelle oder -spalte beziehungsweise den Namen der deklarierten Variablen

ORACLE

Copyright © 2014, Oracle und/oder verbundene Unternehmen. All rights reserved. Alle Rechte vorbehalten.

Richtige Antworten: a, c, d

Attribut %TYPE

PL/SQL-Variablen sind in der Regel so deklariert, dass sie in einer Datenbank gespeicherte Daten aufnehmen und bearbeiten. Wenn Sie PL/SQL-Variablen zum Speichern von Spaltenwerten deklarieren, müssen Sie sicherstellen, dass die Variablen den richtigen Datentyp und die richtige Gesamtstelligenzahl aufweisen. Ist dies nicht der Fall, tritt während der Ausführung ein PL/SQL-Fehler auf. Wenn Sie große Unterprogramme entwickeln müssen, kann dies zeitaufwändig sein und leicht zu Fehlern führen.

Anstatt den Datentyp und die Gesamtstelligenzahl von Variablen hartzudcodieren, können Sie Variablen mithilfe des Attributs %TYPE entsprechend einer zuvor deklarierten Variablen oder Datenbankspalte deklarieren. Das Attribut %TYPE wird meist verwendet, wenn der in der Variablen gespeicherte Wert von einer Tabelle in der Datenbank abgeleitet wird. Wenn Sie eine Variable mithilfe des Attributs %TYPE deklarieren, müssen Sie ihm den Namen der Datenbanktabelle und der Spalte voranstellen. Wenn Sie eine zuvor deklarierte Variable referenzieren, stellen Sie der zu deklarierenden Variablen den Variablenamen der zuvor deklarierten Variablen voran. %TYPE bietet den Vorteil, dass Sie die Variable nicht ändern müssen, wenn die Spalte geändert wird. Wenn die Variable in Berechnungen verwendet wird, müssen Sie außerdem keine Gesamtstelligenzahl festlegen.

Attribut %ROWTYPE

Mit dem Attribut %ROWTYPE wird ein Record deklariert, der eine gesamte Zeile einer Tabelle oder View speichern kann. In der Lektion "Mit zusammengesetzten Datentypen arbeiten" lernen Sie dieses Attribut näher kennen.

Zusammenfassung

Nach Ablauf dieser Lektion haben Sie folgende Ziele erreicht:

- Gültige und ungültige IDs erkennen
- Variablen im deklarativen Bereich eines PL/SQL-Blockes deklarieren
- Variablen initialisieren und im ausführbaren Bereich verwenden
- Zwischen skalaren und zusammengesetzten Datentypen unterscheiden
- Attribut %TYPE verwenden
- Bind-Variablen verwenden



Copyright © 2014, Oracle und/oder verbundene Unternehmen. All rights reserved. Alle Rechte vorbehalten.

Anonyme PL/SQL-Blöcke sind grundlegende, unbenannte Einheiten von PL/SQL-Programmen. Sie bestehen aus einer Gruppe von SQL- oder PL/SQL-Anweisungen, die eine logische Funktion ausführen. Der deklarative Teil ist der erste Teil eines PL/SQL-Blockes. In ihm werden Objekte wie Variablen, Konstanten, Cursor und Definitionen von Fehlersituationen (*Exceptions*) deklariert.

In dieser Lektion wurde erläutert, wie Sie Variablen im deklarativen Bereich deklarieren. Sie haben einige der Richtlinien zum Deklarieren von Variablen kennengelernt. Außerdem wissen Sie nun, wie Variablen beim Deklarieren initialisiert werden.

Der ausführbare Teil eines PL/SQL-Blockes ist obligatorisch und enthält SQL- und PL/SQL-Anweisungen zum Abfragen und Bearbeiten von Daten. Sie haben erfahren, wie Sie Variablen im ausführbaren Bereich initialisieren, verwenden und bearbeiten.

Übungen zu Lektion 3 – Übersicht

Diese Übung behandelt folgende Themen:

- Gültige IDs identifizieren
- Gültige Variablen Deklarationen bestimmen
- Variablen in einem anonymen Block deklarieren
- Variablen mit dem Attribut %TYPE deklarieren
- Bind-Variable deklarieren und ausgeben
- PL/SQL-Block ausführen

ORACLE

Copyright © 2014, Oracle und/oder verbundene Unternehmen. All rights reserved. Alle Rechte vorbehalten.

Die Aufgaben 1, 2 und 3 werden nicht am Rechner durchgeführt.

Ausführbare Anweisungen erstellen

4

ORACLE

Copyright © 2014, Oracle und/oder verbundene Unternehmen. All rights reserved. Alle Rechte vorbehalten.

Ziele

Nach Ablauf dieser Lektion haben Sie folgende Ziele erreicht:

- Lexikalische Einheiten in PL/SQL-Blöcken identifizieren
- Built-In-SQL-Funktionen in PL/SQL verwenden
- Beschreiben, wann implizite Konvertierungen vorkommen und explizite Konvertierungen verarbeitet werden müssen
- Verschachtelte Blöcke erstellen und Variablen mit Labels kennzeichnen
- Lesbaren Code mit entsprechender Einrückung erstellen
- Sequences in PL/SQL-Ausdrücken verwenden



Copyright © 2014, Oracle und/oder verbundene Unternehmen. All rights reserved. Alle Rechte vorbehalten.

Sie haben gelernt, wie Sie in PL/SQL-Blöcken Variablen deklarieren und ausführbare Anweisungen erstellen. In dieser Lektion wird beschrieben, wie ein PL/SQL-Block aus lexikalischen Einheiten gebildet wird. Sie lernen, verschachtelte Blöcke zu erstellen. Außerdem werden der Gültigkeitsbereich und die Sichtbarkeit von Variablen in verschachtelten Blöcken sowie das Kennzeichnen von Variablen mit Labels erläutert.

Agenda

- Ausführbare Anweisungen in einem PL/SQL-Block erstellen
- Verschachtelte Blöcke erstellen
- Operatoren verwenden und lesbaren Code entwickeln

ORACLE

Copyright © 2014, Oracle und/oder verbundene Unternehmen. All rights reserved. Alle Rechte vorbehalten.

Lexikalische Einheiten in PL/SQL-Blöcken

Lexikalische Einheiten:

- sind Bausteine von PL/SQL-Blöcken
- sind Folgen von Zeichen wie Buchstaben, Zahlen, Tabulatoren, Leerzeichen, Zeilenschaltungen und Symbolen
- lassen sich folgt klassifizieren:
 - IDs: v_fname, c_percent
 - Begrenzungszeichen: ; , +, -
 - Literale: John, 428, True
 - Kommentare: --, /* */



Copyright © 2014, Oracle und/oder verbundene Unternehmen. All rights reserved. Alle Rechte vorbehalten.

Lexikalische Einheiten beinhalten Buchstaben, Zahlen, Sonderzeichen, Tabulatoren, Leerzeichen, Zeilenschaltungen und Symbole.

- **IDs:** IDs sind die Namen, die PL/SQL-Objekten zugewiesen werden. Sie haben gelernt, gültige und ungültige IDs zu erkennen. Denken Sie daran, dass Schlüsselwörter nicht als IDs verwendet werden können.

IDs in Anführungszeichen:

- sorgen dafür, dass bei IDs die Groß-/Kleinschreibung beachtet werden muss
- beinhalten Zeichen wie Leerzeichen
- verwenden reservierte Wörter

Beispiele:

```
"begin date" DATE;
"end date"    DATE;
"exception thrown" BOOLEAN DEFAULT TRUE;
```

Diese Variablen müssen in allen nachfolgenden Verwendungen stets in doppelte Anführungszeichen gesetzt werden. Von der Verwendung von IDs in Anführungszeichen wird jedoch abgeraten.

- **Begrenzungszeichen:** Begrenzungszeichen sind Symbole mit einer speziellen Bedeutung. Sie haben bereits gelernt, dass das Semikolon (;) verwendet wird, um SQL- oder PL/SQL-Anweisungen zu beenden. Daher handelt es sich bei ; um ein Beispiel für ein Begrenzungszeichen.

Weitere Informationen finden Sie im Dokument *PL/SQL User's Guide and Reference*.

Begrenzungszeichen sind einfache oder zusammengesetzte Symbole mit einer Sonderbedeutung in PL/SQL.

Einfache Symbole

Symbol	Bedeutung
+	Operator für die Addition
-	Operator für die Subtraktion/Verneinung
*	Operator für die Multiplikation
/	Operator für die Division
=	Gleichheitsoperator
@	Indikator für Remote-Zugriff
;	Abschlusszeichen für eine Anweisung

Zusammengesetzte Symbole

Symbol	Bedeutung
<>	Ungleichheitsoperator
!=	Ungleichheitsoperator
	Verkettungsoperator
--	Indikator für einzeiligen Kommentar
/*	Anfangsbegrenzungszeichen Kommentar
*/	Endbegrenzungszeichen Kommentar
:=	Zuweisungsoperator

Hinweis: Dies ist nur ein Ausschnitt und keine vollständige Liste der Begrenzungszeichen.

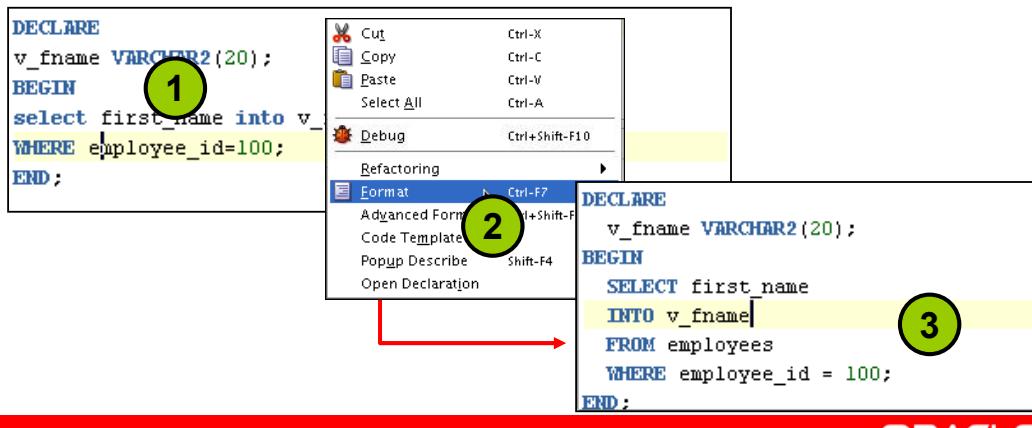
- **Literele:** Jeder Wert, der einer Variablen zugewiesen wird, ist ein Literal. Alle Zeichen, Zahlen, booleschen Werte oder Datumswerte, die keine IDs sind, werden Literale genannt. Literale lassen sich wie folgt klassifizieren:
 - **Zeichenliterale:** Zeichenfolgenliterale besitzen generell den Datentyp CHAR oder VARCHAR2 und werden daher als Zeichenliterale bezeichnet (Beispiele: John und 12c).
 - **Numerische Literale:** Numerische Literale stehen für eine Ganzzahl oder einen realen Wert (zum Beispiel 428 und 1.276).
 - **Boolesche Literale:** Werte, die booleschen Variablen zugewiesen werden, werden als boolesche Literale bezeichnet. TRUE, FALSE und NULL sind boolesche Literale oder Schlüsselwörter.
- **Kommentare:** Bei der Programmierung hat es sich als sinnvoll erwiesen, den Zweck eines Codes zu erklären. Wenn Sie die Erklärung jedoch in einen PL/SQL-Block aufnehmen, kann der Compiler diese Anweisungen nicht interpretieren. Sie müssen daher angeben können, dass diese Anweisungen nicht zu kompilieren sind. Zu diesem Zweck werden Kommentare verwendet. Kommentierte Anleitungen werden vom Compiler nicht interpretiert.
 - Eine einzelne Zeile wird mit zwei Bindestrichen --) als Kommentar gekennzeichnet.
 - Mehrere Zeilen werden mit den Anfangs- und Endbegrenzungszeichen für Kommentare /* und */ gekennzeichnet.

PL/SQL-Blöcke – Syntax und Richtlinien

- Literale verwenden
 - Zeichen- und Datumsliterale müssen in einfache Anführungszeichen gesetzt werden.
 - Ziffern können in einfacher oder wissenschaftlicher Darstellung geschrieben werden.

```
v_name := 'Henderson';
```

- Code formatieren: Anweisungen können sich über mehrere Zeilen erstrecken.



Copyright © 2014, Oracle und/oder verbundene Unternehmen. All rights reserved. Alle Rechte vorbehalten.

Literale verwenden

Ein Literal ist ein expliziter numerischer, Zeichenfolgen-, Datums- oder boolescher Wert, der nicht durch eine ID repräsentiert wird.

- Zeichenliterale beinhalten alle druckbaren Zeichen des PL/SQL-Zeichensatzes: Buchstaben, Zahlen, Leer- und Sonderzeichen.
- Numerische Literale können entweder durch einen einfachen Wert (etwa `-32.5`) oder durch wissenschaftliche Notation (zum Beispiel `2E5`, das heißt $2 \times 10^5 = 200.000$) dargestellt werden.

Code formatieren

In einem PL/SQL-Block kann sich eine SQL-Anweisung über mehrere Zeilen erstrecken (siehe 3. Beispiel auf der Folie).

Sie können eine unformatierte SQL-Anweisung mithilfe des SQL Worksheet-Kontextmenüs formatieren (siehe 1. Beispiel auf der Folie). Klicken Sie mit der rechten Maustaste auf das aktive SQL Worksheet, und wählen Sie im daraufhin angezeigten Kontextmenü die Option **Format** (siehe 2. Beispiel).

Hinweis: Sie können den Code auch mit dem Tasturbefehl STRG+F7 formatieren.

Code kommentieren

- Einzeligen Kommentaren zwei Bindestriche (--) voranstellen
- Blockkommentare in die Symbole /* und */ einschließen

Beispiel:

```
DECLARE
  ...
  v_annual_sal NUMBER (9,2);
BEGIN
  /* Compute the annual salary based on the
     monthly salary input from the user */
  v_annual_sal := monthly_sal * 12;
  --The following line displays the annual salary
  DBMS_OUTPUT.PUT_LINE(v_annual_sal);
END;
/
```

ORACLE

Copyright © 2014, Oracle und/oder verbundene Unternehmen. All rights reserved. Alle Rechte vorbehalten.

Sie sollten Code kommentieren, um die einzelnen Phasen zu dokumentieren und das Debugging zu erleichtern. In PL/SQL-Code gilt Folgendes:

- Einzeligen Kommentaren werden zwei Bindestriche (--) vorangestellt.
- Kommentare können auch in die Symbole /* und */ eingeschlossen werden.

Hinweis: Bei mehrzeiligen Kommentaren können Sie entweder jeder Kommentarzeile zwei Bindestriche voranstellen oder das Blockkommentarformat verwenden.

Kommentare dienen ausschließlich der Information und erzwingen keinerlei Bedingungen oder Verhalten hinsichtlich der Logik oder Daten. Richtig platzierte Kommentare verbessern die Lesbarkeit von Code und erleichtern die spätere Codeverwaltung.

SQL-Funktionen in PL/SQL

- In prozeduralen Anweisungen verfügbar:
 - Single-Row-Funktionen
- In prozeduralen Anweisungen nicht verfügbar:
 - DECODE
 - Gruppenfunktionen

ORACLE

Copyright © 2014, Oracle und/oder verbundene Unternehmen. All rights reserved. Alle Rechte vorbehalten.

SQL beinhaltet eine Reihe von vordefinierten Funktionen, die in SQL-Anweisungen verwendet werden können. Die meisten dieser Funktionen sind in PL/SQL-Ausdrücken gültig (beispielsweise Single-Row-Zahlen- und Zeichenfunktionen, Funktionen zur Datentypkonvertierung sowie Datums- und Zeitstempelfunktionen).

Folgende Funktionen stehen in prozeduralen Anweisungen nicht zur Verfügung:

- DECODE
- Gruppenfunktionen: AVG, MIN, MAX, COUNT, SUM, STDDEV und VARIANCE
Gruppenfunktionen gelten für Gruppen von Tabellenzeilen und stehen daher nur in SQL-Anweisungen in PL/SQL-Blöcken zur Verfügung. Die hier erwähnten Funktionen sind nur ein Ausschnitt aus der Liste der Funktionen.

SQL-Funktionen in PL/SQL – Beispiele

- Länge einer Zeichenfolge abrufen:

```
v_desc_size INTEGER(5);
v_prod_description VARCHAR2(70):='You can use this
product with your radios for higher frequency';

-- get the length of the string in prod_description
v_desc_size:= LENGTH(v_prod_description);
```

- Anzahl der Beschäftigungsmonate eines Mitarbeiters abrufen:

```
v_tenure:= MONTHS_BETWEEN (CURRENT_DATE, v_hiredate);
```



Copyright © 2014, Oracle und/oder verbundene Unternehmen. All rights reserved. Alle Rechte vorbehalten.

Mithilfe von SQL-Funktionen können Sie Daten bearbeiten. Diese Funktionen lassen sich in folgende Kategorien einteilen:

- Numerische Funktionen
- Zeichenfunktionen
- Konvertierungsfunktionen
- Datumsfunktionen
- Sonstige Funktionen

Sequences in PL/SQL-Ausdrücken

Ab 11g:

```
DECLARE
    v_new_id NUMBER;
BEGIN
    v_new_id := my_seq.NEXTVAL;
END;
/
```

Vor 11g:

```
DECLARE
    v_new_id NUMBER;
BEGIN
    SELECT my_seq.NEXTVAL INTO v_new_id FROM Dual;
END;
/
```

ORACLE

Copyright © 2014, Oracle und/oder verbundene Unternehmen. All rights reserved. Alle Rechte vorbehalten.

In Oracle Database 11g und höher können Sie die Pseudospalten NEXTVAL und CURRVAL in einem beliebigen PL/SQL-Kontext verwenden, in dem Ausdrücke vom Datentyp NUMBER zulässig sind. Die herkömmliche Abfrage von Sequences mit SELECT-Anweisungen ist zwar nach wie vor gültig, sollte jedoch nicht verwendet werden.

Vor Oracle Database 11g mussten Sie zwingend eine SQL-Anweisung erstellen, um in einer untergeordneten PL/SQL-Routine einen Sequence-Objektwert verwenden zu können. In der Regel wurden dabei die Pseudospalten NEXTVAL und CURVAL mithilfe einer SELECT-Anweisung referenziert, um eine Sequence-Nummer zu ermitteln. Diese Methode machte die Verwendung von Sequences umständlich.

In Oracle Database 11g und höher ist es nicht mehr erforderlich, eine SQL-Anweisung zum Abruf eines Sequence-Wertes zu erstellen. Das weiterentwickelte Sequence-Feature bietet folgende Vorteile:

- Verbesserte Sequence-Verwendbarkeit
- Weniger Eingaben durch den Entwickler
- Klarer strukturierter Code

Datentypen konvertieren

- Daten werden in vergleichbare Datentypen konvertiert.
- Es gibt zwei Typen:
 - Implizite Konvertierung
 - Explizite Konvertierung
- Funktionen:
 - TO_CHAR
 - TO_DATE
 - TO_NUMBER
 - TO_TIMESTAMP

ORACLE

Copyright © 2014, Oracle und/oder verbundene Unternehmen. All rights reserved. Alle Rechte vorbehalten.

Das Konvertieren von Datentypen ist in allen Programmiersprachen eine gängige Anforderung. PL/SQL kann diese Konvertierungen mit skalaren Datentypen durchführen. Es gibt zwei Arten von Datentypkonvertierungen:

Implizite Konvertierungen: PL/SQL versucht, Datentypen dynamisch zu konvertieren, wenn diese gemischt in einer Anweisung vorkommen. Betrachten Sie das folgende Beispiel:

```
DECLARE
    v_salary NUMBER(6) := 6000;
    v_sal_hike VARCHAR2(5) := '1000';
    v_total_salary v_salary%TYPE;
BEGIN
    v_total_salary := v_salary + v_sal_hike;
    DBMS_OUTPUT.PUT_LINE(v_total_salary);
END;
/
```

In diesem Beispiel weist die Variable `sal_hike` den Typ `VARCHAR2` auf. Beim Berechnen der Gesamtsumme der Gehälter konvertiert PL/SQL die Variable `sal_hike` zunächst in `NUMBER` und führt dann den Vorgang aus. Der Datentyp des Ergebnisses ist `NUMBER`.

Implizite Konvertierungen sind möglich zwischen:

- Zeichen und Zahlen
- Zeichen und Datumsangaben

Explizite Konvertierungen: Sie können Werte mithilfe von Built-In-Funktionen von einem Datentyp in einen anderen konvertieren. Beispiel: Um einen CHAR-Wert in einen DATE- oder NUMBER-Wert zu konvertieren, verwenden Sie TO_DATE beziehungsweise TO_NUMBER.

Datentypen konvertieren

1

```
-- implicit data type conversion  
v_date_of_joining DATE:= '02-Feb-2000';
```

2

```
-- error in data type conversion  
v_date_of_joining DATE:= 'February 02,2000';
```

3

```
-- explicit data type conversion  
v_date_of_joining DATE:= TO_DATE('February  
02,2000', 'Month DD, YYYY');
```

ORACLE

Copyright © 2014, Oracle und/oder verbundene Unternehmen. All rights reserved. Alle Rechte vorbehalten.

Beachten Sie die drei Beispiele impliziter und expliziter Konvertierungen vom Datentyp DATE auf der Folie:

1. Da das der Variablen `date_of_joining` zugewiesene Zeichenfolgenliteral im Standardformat vorliegt, wird in diesem Beispiel eine implizite Konvertierung ausgeführt und das angegebene Datum der Variablen `date_of_joining` zugewiesen.
2. PL/SQL gibt einen Fehler zurück, da das zuzuweisende Datum nicht das Standardformat aufweist.
3. Mit der Funktion `TO_DATE` können Sie das gegebene Datum explizit in ein bestimmtes Format konvertieren und der Variablen `date_of_joining` vom Datentyp DATE zuweisen.

Agenda

- Ausführbare Anweisungen in einem PL/SQL-Block erstellen
- Verschachtelte Blöcke erstellen
- Operatoren verwenden und lesbaren Code entwickeln

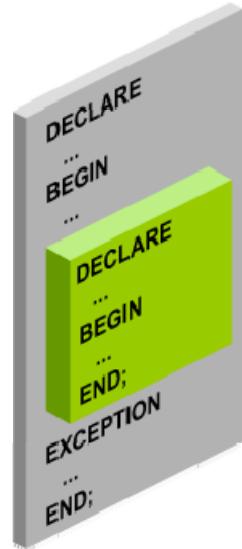
ORACLE

Copyright © 2014, Oracle und/oder verbundene Unternehmen. All rights reserved. Alle Rechte vorbehalten.

Verschachtelte Blöcke

PL/SQL-Blöcke lassen sich verschachteln.

- Der ausführbare Bereich (`BEGIN ... END`) kann verschachtelte Blöcke enthalten.
- Der Exception-Behandlungsbereich kann verschachtelte Blöcke enthalten.



ORACLE

Copyright © 2014, Oracle und/oder verbundene Unternehmen. All rights reserved. Alle Rechte vorbehalten.

Dank der prozeduralen Fähigkeiten von PL/SQL können Anweisungen verschachtelt werden. Blöcke lassen sich überall verschachteln, wo ausführbare Anweisungen zulässig sind. Der verschachtelte Block wird somit zu einer Anweisung. Wenn der ausführbare Bereich Code für viele logisch zusammengehörige Funktionalitäten enthält, die mehrere Geschäftsanforderungen unterstützen, können Sie ihn in kleinere Blöcke unterteilen. Der Exception-Bereich kann ebenfalls verschachtelte Blöcke enthalten.

Verschachtelte Blöcke – Beispiel

```
DECLARE
  v_outer_variable VARCHAR2(20) := 'GLOBAL VARIABLE';
BEGIN
  DECLARE
    v_inner_variable VARCHAR2(20) := 'LOCAL VARIABLE';
  BEGIN
    DBMS_OUTPUT.PUT_LINE(v_inner_variable);
    DBMS_OUTPUT.PUT_LINE(v_outer_variable);
  END;
  DBMS_OUTPUT.PUT_LINE(v_outer_variable);
END;
```

```
anonymous block completed
LOCAL VARIABLE
GLOBAL VARIABLE
GLOBAL VARIABLE
```

ORACLE

Copyright © 2014, Oracle und/oder verbundene Unternehmen. All rights reserved. Alle Rechte vorbehalten.

Das Beispiel auf der Folie zeigt einen äußeren (übergeordneten) Block und einen verschachtelten (untergeordneten) Block. Die Variable `v_outer_variable` wird im äußeren Block deklariert, die Variable `v_inner_variable` im inneren Block.

Die Variable `v_outer_variable` ist für den äußeren Block eine lokale Variable, während sie für den inneren Block eine globale Variable ist. Wenn Sie im inneren Block auf diese Variable zugreifen, sucht PL/SQL zuerst im inneren Block nach einer lokalen Variablen mit diesem Namen. Da sich im inneren Block keine Variable mit diesem Namen befindet, sucht PL/SQL im äußeren Block nach der Variablen. Aus diesem Grund wird die Variable `v_outer_variable` als globale Variable für alle einschließenden Blöcke betrachtet. Wie auf der Folie zu sehen ist, können Sie auf diese Variable im inneren Block zugreifen. In einem PL/SQL-Block deklarierte Variablen werden für diesen Block als lokal und für alle seine Unterblöcke als global betrachtet.

Die Variable `v_inner_variable` ist für den inneren Block eine lokale und keine globale Variable, da der innere Block nicht über verschachtelte Blöcke verfügt. Auf diese Variable kann nur innerhalb des inneren Blockes zugegriffen werden. Wenn PL/SQL die lokal deklarierte Variable nicht findet, wird sie im deklarativen Bereich der übergeordneten Blöcke gesucht. PL/SQL sucht nicht in den untergeordneten Blöcken.

Variablen – Gültigkeitsbereich und Sichtbarkeit

```
DECLARE
    v_father_name VARCHAR2(20):='Patrick';
    v_date_of_birth DATE:='20-Apr-1972';
BEGIN
    DECLARE
        v_child_name VARCHAR2(20):='Mike';
        v_date_of_birth DATE:='12-Dec-2002';
    BEGIN
        DBMS_OUTPUT.PUT_LINE('Father''s Name: '||v_father_name);
        DBMS_OUTPUT.PUT_LINE('Date of Birth: '||v_date_of_birth); ←
        DBMS_OUTPUT.PUT_LINE('Child''s Name: '||v_child_name);
    END;
    DBMS_OUTPUT.PUT_LINE('Date of Birth: '||v_date_of_birth);
END;
/
```

ORACLE

Copyright © 2014, Oracle und/oder verbundene Unternehmen. All rights reserved. Alle Rechte vorbehalten.

Der auf der Folie gezeigte Block ergibt folgende Ausgabe:

```
anonymous block completed
Father's Name: Patrick
Date of Birth: 12-DEC-02
Child's Name: Mike
Date of Birth: 20-APR-72
```

Beachten Sie das Geburtsdatum von Vater und Kind. Die Informationen in der Ausgabe sind nicht korrekt, da der Gültigkeitsbereich und die Sichtbarkeit der Variablen nicht richtig angewendet wurden.

- Der *Gültigkeitsbereich* einer Variablen bezeichnet den Teil des Programms, in dem die Variable deklariert wird und zugänglich ist.
- Die *Sichtbarkeit* einer Variablen bezeichnet den Teil des Programms, in dem ohne einen Qualifier auf die Variable zugegriffen werden kann.

Gültigkeitsbereich

- Die Variable `v_father_name` und das erste Vorkommen der Variablen `v_date_of_birth` werden im äußeren Block deklariert. Für diese Variablen gilt der Gültigkeitsbereich des Blockes, in dem sie deklariert sind. Daher ist der Gültigkeitsbereich auf den äußeren Block begrenzt.

Gültigkeitsbereich

- Die Variablen `v_child_name` und `v_date_of_birth` werden im inneren beziehungsweise verschachtelten Block deklariert. Sie können nur innerhalb des verschachtelten Blockes, nicht aber im äußeren Block auf die Variablen zugreifen. Liegt eine Variable außerhalb des Gültigkeitsbereichs, gibt PL/SQL den von der Variablen belegten Speicher frei. Diese Variablen sind daher nicht referenzierbar.

Sichtbarkeit

- Die im äußeren Block deklarierte Variable `v_date_of_birth` ist auch im inneren Block gültig. Sie ist jedoch im inneren Block nicht sichtbar, da der innere Block eine lokale Variable gleichen Namens enthält.
 1. Untersuchen Sie den Code im ausführbaren Bereich des PL/SQL-Blockes. Sie können den Namen des Vaters und des Kindes sowie das Geburtsdatum anzeigen. In diesem Fall kann nur das Geburtsdatum des Kindes angezeigt werden, da das Geburtsdatum des Vaters nicht sichtbar ist.
 2. Das Geburtsdatum des Vaters ist im äußeren Block sichtbar und kann somit angezeigt werden.

Hinweis: Ein Block darf keine gleichnamigen Variablen enthalten. Wie das Beispiel veranschaulicht, können Sie gleichnamige Variablen jedoch in zwei verschiedenen (verschachtelten) Blöcken deklarieren. Die beiden durch die IDs dargestellten Elemente sind voneinander unabhängig. Änderungen an einem Element wirken sich nicht auf das andere Element aus.

Qualifier mit verschachtelten Blöcken

```
BEGIN <>outer>>
DECLARE
    v_father_name VARCHAR2(20) := 'Patrick';
    v_date_of_birth DATE := '20-Apr-1972';
BEGIN
    DECLARE
        v_child_name VARCHAR2(20) := 'Mike';
        v_date_of_birth DATE := '12-Dec-2002';
    BEGIN
        DBMS_OUTPUT.PUT_LINE('Father''s Name: ' || v_father_name);
        DBMS_OUTPUT.PUT_LINE('Date of Birth: '
                             || outer.v_date_of_birth);
        DBMS_OUTPUT.PUT_LINE('Child''s Name: ' || v_child_name);
        DBMS_OUTPUT.PUT_LINE('Date of Birth: ' || v_date_of_birth);
    END;
END;
END outer;
```

ORACLE

Copyright © 2014, Oracle und/oder verbundene Unternehmen. All rights reserved. Alle Rechte vorbehalten.

Ein *Qualifier* ist ein Label, das einem Block zugewiesen wird. Mithilfe von Qualifiern können Sie auf die Variablen zugreifen, die innerhalb des Gültigkeitsbereichs liegen, aber nicht sichtbar sind.

Beispiel

Im Codebeispiel gilt Folgendes:

- Der äußere Block ist mit dem Label `outer` gekennzeichnet.
- Im inneren Block wird mit dem Qualifier `outer` auf die Variable `v_date_of_birth` zugegriffen, die im äußeren Block deklariert ist. Daher kann das Geburtsdatum sowohl des Vaters als auch des Kindes im inneren Block angezeigt werden.
- Die Ausgabe des Codes auf der Folie enthält die richtigen Informationen:

```
anonymous block completed
Father's Name: Patrick
Date of Birth: 20-APR-72
Child's Name: Mike
Date of Birth: 12-DEC-02
```

Hinweis: Die Kennzeichnung mit Labels ist nicht auf den äußeren Block beschränkt. Jeder Block kann mit Labels versehen werden.

Aufgabe: Gültigkeitsbereich von Variablen bestimmen

```
BEGIN <<outer>>
DECLARE
    v_sal      NUMBER(7,2) := 60000;
    v_comm     NUMBER(7,2) := v_sal * 0.20;
    v_message  VARCHAR2(255) := ' eligible for commission';
BEGIN
    DECLARE
        v_sal      NUMBER(7,2) := 50000;
        v_comm     NUMBER(7,2) := 0;
        v_total_comp NUMBER(7,2) := v_sal + v_comm;
    BEGIN
        1          v_message := 'CLERK not' || v_message;
        outer.v_comm := v_sal * 0.30;
    END;
    2          v_message := 'SALESMAN' || v_message;
    END;
END outer;
/
```

ORACLE

Copyright © 2014, Oracle und/oder verbundene Unternehmen. All rights reserved. Alle Rechte vorbehalten.

Werten Sie den PL/SQL-Block auf der Folie aus. Bestimmen Sie die folgenden Werte entsprechend den Regeln für Gültigkeitsbereiche:

1. Wert von `v_message` an 1. Position
2. Wert von `v_total_comp` an 2. Position
3. Wert von `v_comm` an 1. Position
4. Wert von `outer.v_comm` an 1. Position
5. Wert von `v_comm` an 2. Position
6. Wert von `v_message` an 2. Position

Antworten: Gültigkeitsbereich von Variablen bestimmen

Die Antworten auf die Fragen bezüglich des Gültigkeitsbereichs lauten wie folgt:

1. Wert von `v_message` an 1. Position: **CLERK not eligible for commission (nicht provisionsberechtigt)**
2. Wert von `v_total_comp` an 2. Position: **Fehler. v_total_comp ist hier nicht sichtbar, da es im inneren Block definiert ist.**
3. Wert von `v_comm` an 1. Position: **0**
4. Wert von `outer.v_comm` an 1. Position: **12000**
5. Wert von `v_comm` an 2. Position: **15000**
6. Wert von `v_message` an 2. Position: **SALESMANCLERK not eligible for commission (nicht provisionsberechtigt)**

Agenda

- Ausführbare Anweisungen in einem PL/SQL-Block erstellen
- Verschachtelte Blöcke erstellen
- Operatoren verwenden und lesbaren Code entwickeln

ORACLE

Copyright © 2014, Oracle und/oder verbundene Unternehmen. All rights reserved. Alle Rechte vorbehalten.

Operatoren in PL/SQL

- Logisch
- Arithmetische Operatoren
- Verkettungsoperatoren
- Klammern zur Steuerung der Reihenfolge

Wie in SQL

- Exponentialoperatoren (**)

ORACLE

Copyright © 2014, Oracle und/oder verbundene Unternehmen. All rights reserved. Alle Rechte vorbehalten.

Die Vorgänge in einem Ausdruck werden entsprechend ihrer Priorität in einer bestimmten Reihenfolge ausgeführt. In der folgenden Tabelle wird die Standardreihenfolge der Vorgänge von höchster nach niedrigster Priorität gezeigt:

Operator	Vorgang
**	Exponentialoperator
+ , -	Vorzeichen, Verneinung
* , /	Multiplikation, Division
+ , - ,	Addition, Subtraktion, Verkettung
=, <, >, <=, >=, <>, !=, ~=, ^=, IS NULL, LIKE, BETWEEN, IN	Vergleich
NOT	Logische Verneinung
AND	Verbindung
OR	Inklusion

Operatoren in PL/SQL – Beispiele

- Schleifenzähler erhöhen

```
loop_count := loop_count + 1;
```

- Wert eines booleschen Flags festlegen

```
good_sal := sal BETWEEN 50000 AND 150000;
```

- Prüfen, ob eine Personalnummer einen Wert enthält

```
valid := (empno IS NOT NULL);
```



Copyright © 2014, Oracle und/oder verbundene Unternehmen. All rights reserved. Alle Rechte vorbehalten.

Beim Arbeiten mit `NULL`-Werten können Sie einige häufige Fehler vermeiden, indem Sie die folgenden Regeln beachten:

- Vergleiche mit `NULL`-Werten ergeben stets `NULL`.
- Wenn der logische Operator `NOT` auf einen `NULL`-Wert angewendet wird, lautet das Ergebnis `NULL`.
- Wenn die Bedingung in `IF`-Anweisungen den Wert `NULL` ergibt, wird die zugehörige Anweisungsfolge nicht ausgeführt.

Richtlinien für die Programmierung

Codeverwaltung wie folgt vereinfachen:

- Code mit Kommentaren dokumentieren
- Konvention für Groß-/Kleinschreibung im Code festlegen
- Benennungskonventionen für IDs und andere Objekte festlegen
- Lesbarkeit durch Einrückungen verbessern

ORACLE

Copyright © 2014, Oracle und/oder verbundene Unternehmen. All rights reserved. Alle Rechte vorbehalten.

Folgen Sie beim Entwickeln von PL/SQL-Blöcken den auf der Folie angegebenen Richtlinien für die Programmierung, um übersichtlich strukturierten Code zu erstellen und den Verwaltungsaufwand zu reduzieren.

Codierungskonventionen

Die folgende Tabelle gibt Anregungen, wie Sie Schlüsselwörter und benannte Objekte mithilfe von Groß- und Kleinbuchstaben unterscheiden können.

Kategorie	Schreibweise	Beispiele
SQL-Anweisungen	Großbuchstaben	SELECT, INSERT
PL/SQL-Schlüsselwörter	Großbuchstaben	DECLARE, BEGIN, IF
Datentypen	Großbuchstaben	VARCHAR2, BOOLEAN
IDs und Parameter	Kleinbuchstaben	v_sal, emp_cursor, g_sal, p_empno
Datenbanktabellen	Kleinbuchstaben, Plural	employees, departments
Datenbankspalten	Kleinbuchstaben, Singular	employee_id, department_id

Code einrücken

Code durch Einrücken der einzelnen Codeebenen übersichtlich strukturieren

```
BEGIN
    IF x=0 THEN
        y:=1;
    END IF;
END;
/
```

```
DECLARE
    v_deptno          NUMBER(4);
    v_location_id    NUMBER(4);
BEGIN
    SELECT department_id,
           location_id
    INTO   v_deptno,
           v_location_id
    FROM   departments
    WHERE  department_name
           = 'Sales';
    ...
END;
/
```

ORACLE

Copyright © 2014, Oracle und/oder verbundene Unternehmen. All rights reserved. Alle Rechte vorbehalten.

Um den Code übersichtlich zu strukturieren und die Lesbarkeit zu verbessern, rücken Sie die einzelnen Codeebenen ein. Um die Struktur zu veranschaulichen, können Sie Zeilen durch Zeilenschaltungen trennen und mit Leerzeichen und Tabulatoren einrücken. Vergleichen Sie die folgenden IF-Anweisungen hinsichtlich ihrer Lesbarkeit:

```
IF x>y THEN max:=x;ELSE max:=y;END IF;
```

```
IF x > y THEN
    max := x;
ELSE
    max := y;
END IF;
```

Quiz

Die meisten Single-Row-Funktionen von SQL, beispielsweise Single-Row-Zahlen-, Zeichen-, Konvertierungs- und Datumsfunktionen, können in PL/SQL-Ausdrücken verwendet werden.

- a. Richtig
- b. Falsch



Copyright © 2014, Oracle und/oder verbundene Unternehmen. All rights reserved. Alle Rechte vorbehalten.

Richtige Antwort: a

SQL-Funktionen in PL/SQL

SQL beinhaltet eine Reihe von vordefinierten Funktionen, die in SQL-Anweisungen verwendet werden können. Die meisten dieser Funktionen sind in PL/SQL-Ausdrücken gültig (beispielsweise Single-Row-Zahlen- und Zeichenfunktionen, Funktionen zur Datentypkonvertierung sowie Datums- und Zeitstempelfunktionen).

Folgende Funktionen stehen in prozeduralen Anweisungen nicht zur Verfügung:

- DECODE
- Gruppenfunktionen: AVG, MIN, MAX, COUNT, SUM, STDDEV und VARIANCE
Gruppenfunktionen gelten für Gruppen von Tabellenzeilen und stehen daher nur in SQL-Anweisungen in PL/SQL-Blöcken zur Verfügung. Die hier erwähnten Funktionen sind nur ein Ausschnitt aus der Liste der Funktionen.

Zusammenfassung

Nach Ablauf dieser Lektion haben Sie folgende Ziele erreicht:

- Lexikalische Einheiten in PL/SQL-Blöcken identifizieren
- Built-In-SQL-Funktionen in PL/SQL verwenden
- Verschachtelte Blöcke erstellen, um logisch zusammengehörige Funktionalitäten zu untergliedern
- Entscheiden, wann explizite Konvertierungen ausgeführt werden
- Variablen in verschachtelten Blöcken kennzeichnen
- Sequences in PL/SQL-Ausdrücken verwenden



Copyright © 2014, Oracle und/oder verbundene Unternehmen. All rights reserved. Alle Rechte vorbehalten.

Da PL/SQL eine Erweiterung von SQL ist, gelten die allgemeinen Syntaxregeln von SQL auch für PL/SQL.

Ein Block kann eine beliebige Anzahl verschachtelter Blöcke enthalten, die in seinem ausführbaren Bereich definiert sind. Innerhalb eines Blockes definierte Blöcke werden als Unterblöcke bezeichnet. Sie können Blöcke nur im ausführbaren Bereich eines Blockes verschachteln. Da der Exception-Bereich ebenfalls zum ausführbaren Bereich zählt, kann auch er verschachtelte Blöcke enthalten. Achten Sie bei Verwendung verschachtelter Blöcke auf den richtigen Gültigkeitsbereich und die richtige Sichtbarkeit. Vermeiden Sie gleichnamige IDs in über- und untergeordneten Blöcken.

Die meisten in SQL verfügbaren Funktionen sind auch in PL/SQL-Ausdrücken zulässig. Konvertierungsfunktionen konvertieren einen Wert von einem Datentyp in einen anderen. Vergleichsoperatoren vergleichen zwei Ausdrücke miteinander. Das Ergebnis ist immer TRUE, FALSE oder NULL. In der Regel können Sie Vergleichsoperatoren in IF-Anweisungen und in der WHERE-Klausel von SQL-Anweisungen zur Datenmanipulation verwenden. Relationale Operatoren ermöglichen beliebige Vergleiche zwischen komplexen Ausdrücken.

Übungen zu Lektion 4 – Übersicht

Diese Übung behandelt folgende Themen:

- Regeln für Gültigkeitsbereiche und Verschachtelung
- PL/SQL-Blöcke erstellen und testen



Copyright © 2014, Oracle und/oder verbundene Unternehmen. All rights reserved. Alle Rechte vorbehalten.

Die erste und zweite Aufgabe werden nicht am Rechner durchgeführt.

SQL-Anweisungen in PL/SQL-Blöcken

ORACLE

Copyright © 2014, Oracle und/oder verbundene Unternehmen. All rights reserved. Alle Rechte vorbehalten.

Ziele

Nach Ablauf dieser Lektion haben Sie folgende Ziele erreicht:

- SQL-Anweisungen bestimmen, die direkt in den ausführbaren PL/SQL-Block aufgenommen werden können
- Daten in PL/SQL mit DML-Anweisungen bearbeiten
- Anweisungen zur Transaktionskontrolle in PL/SQL verwenden
- Von SQL-Anweisungen zurückgegebene Werte mithilfe der `INTO`-Klausel speichern
- Zwischen impliziten und expliziten Cursors unterscheiden
- SQL-Cursorattribute verwenden



Copyright © 2014, Oracle und/oder verbundene Unternehmen. All rights reserved. Alle Rechte vorbehalten.

In dieser Lektion wird die Einbettung der SQL-Standardanweisungen `SELECT`, `INSERT`, `UPDATE`, `DELETE` und `MERGE` in PL/SQL-Blöcke beschrieben. Sie lernen, wie Sie in PL/SQL Data Manipulation Language-(DML-)Anweisungen und Anweisungen zur Transaktionskontrolle aufnehmen. Des Weiteren wird auf die Notwendigkeit von Cursors eingegangen und der Unterschied zwischen den beiden Cursorarten erklärt. Außerdem werden in dieser Lektion die verschiedenen SQL-Cursorattribute erläutert, die Sie mit impliziten Cursors verwenden können.

Agenda

- Daten mit PL/SQL abrufen
- Daten mit PL/SQL bearbeiten
- SQL-Cursor – Einführung

ORACLE

Copyright © 2014, Oracle und/oder verbundene Unternehmen. All rights reserved. Alle Rechte vorbehalten.

SQL-Anweisungen in PL/SQL

- Datenbankzeilen mit dem SELECT-Befehl abrufen
- Änderungen in Datenbankzeilen mit DML-Befehlen vornehmen
- Transaktionen mit den Befehlen COMMIT, ROLLBACK und SAVEPOINT kontrollieren

ORACLE

Copyright © 2014, Oracle und/oder verbundene Unternehmen. All rights reserved. Alle Rechte vorbehalten.

In PL/SQL-Blöcken können Sie mit SQL-Anweisungen Daten aus der Datenbanktabelle abrufen und ändern. PL/SQL unterstützt Data Manipulation Language-(DML-)Befehle und Befehle zur Transaktionskontrolle. Mithilfe von DML-Befehlen können Sie die Daten in einer Datenbanktabelle ändern. Beachten Sie bei der Verwendung von DML-Anweisungen und Befehlen zur Transaktionskontrolle in PL/SQL-Blöcken jedoch die folgenden Aspekte:

- Das Schlüsselwort END gibt das Ende eines PL/SQL-Blockes und nicht das Ende einer Transaktion an. Genau wie ein Block mehrere Transaktionen umfassen kann, kann sich eine Transaktion über mehrere Blöcke erstrecken.
- Data Definition Language-(DDL-)Anweisungen wie CREATE TABLE, ALTER TABLE oder DROP TABLE werden von PL/SQL nicht direkt unterstützt. PL/SQL unterstützt Early Binding, das nicht möglich ist, wenn Anwendungen während der Laufzeit Werte übergeben müssen, um Datenbankobjekte zu erstellen. DDL-Anweisungen lassen sich nicht direkt ausführen. Bei diesen Anweisungen handelt es sich um dynamische SQL-Anweisungen. Dynamische SQL-Anweisungen werden zur Laufzeit als Zeichenfolgen erstellt und können Platzhalter für Parameter enthalten. Daher lassen sich DDL-Anweisungen in PL/SQL mit dynamischem SQL ausführen. Die Arbeit mit dynamischem SQL wird im Kurs *Oracle Database: Develop PL/SQL Program Units* ausführlich behandelt.
- Data Control Language-(DCL-)Anweisungen wie GRANT oder REVOKE werden von PL/SQL nicht direkt unterstützt. Sie können diese Anweisungen mit dynamischem SQL ausführen.

SELECT-Anweisungen in PL/SQL

Daten aus der Datenbank mit SELECT-Anweisungen abrufen

Syntax:

```
SELECT  select_list
INTO    {variable_name[ , variable_name]...
        | record_name}
FROM    table
[WHERE  condition];
```

ORACLE

Copyright © 2014, Oracle und/oder verbundene Unternehmen. All rights reserved. Alle Rechte vorbehalten.

Mit der SELECT-Anweisung rufen Sie Daten aus der Datenbank ab.

Richtlinien für den Datenabruf in PL/SQL

- Beenden Sie jede SQL-Anweisung mit einem Semikolon (;).
- Speichern Sie jeden abgerufenen Wert mithilfe der INTO-Klausel in einer Variablen.
- Die WHERE-Klausel ist optional. Sie können damit Eingabevervariablen, Konstanten, Literale und PL/SQL-Ausdrücke angeben. Wenn Sie jedoch die INTO-Klausel verwenden, dürfen Sie nur eine Zeile abrufen. In diesem Fall müssen Sie die WHERE-Klausel verwenden.

select_list Liste mit mindestens einer Spalte. Kann SQL-Ausdrücke, Zeilenfunktionen oder Gruppenfunktionen aufnehmen.

variable_name Skalare Variable, die den abgerufenen Wert aufnimmt

record_name PL/SQL-Record, der die abgerufenen Werte aufnimmt

table Gibt den Namen der Datenbanktabelle an

condition Besteht aus Spaltennamen, Ausdrücken, Konstanten und Vergleichsoperatoren einschließlich PL/SQL-Variablen und Konstanten

- Die Anzahl der Variablen in der `INTO`-Klausel muss der Anzahl der Datenbankspalten in der `SELECT`-Klausel entsprechen. Achten Sie darauf, dass die Positionen übereinstimmen und die Datentypen kompatibel sind.
- Verwenden Sie in SQL-Anweisungen Gruppenfunktionen wie `SUM`, da diese in Tabellen für Zeilengruppen gelten.

SELECT-Anweisungen in PL/SQL

- Die `INTO`-Klausel ist obligatorisch.
- Abfragen dürfen nur eine Zeile zurückgeben.

```
DECLARE
    v_fname VARCHAR2(25);
BEGIN
    SELECT first_name INTO v_fname
    FROM employees WHERE employee_id=200;
    DBMS_OUTPUT.PUT_LINE(' First Name is : ' || v_fname);
END;
/
```

```
anonymous block completed
First Name is : Jennifer
```

ORACLE

Copyright © 2014, Oracle und/oder verbundene Unternehmen. All rights reserved. Alle Rechte vorbehalten.

INTO-Klausel

Die `INTO`-Klausel ist obligatorisch. Sie steht zwischen der `SELECT`- und der `FROM`-Klausel. Sie gibt die Namen der Variablen an, die von SQL durch die `SELECT`-Klausel zurückgegebenen Werte aufnehmen. Sie müssen zu jedem gewählten Element eine Variable angeben, wobei die Reihenfolge der Variablen mit den gewählten Elementen übereinstimmen muss.

Füllen Sie PL/SQL-Variablen oder Hostvariablen mithilfe der `INTO`-Klausel.

Abfragen dürfen nur eine Zeile zurückgeben

Die `SELECT`-Anweisungen in einem PL/SQL-Block fallen in die ANSI-Klassifizierung der eingebetteten SQL-Anweisungen, für die folgende Regel gilt: Abfragen dürfen nur eine Zeile zurückgeben. Gibt eine Abfrage mehr als eine Zeile oder gar keine Zeile zurück, wird ein Fehler generiert.

PL/SQL löst bei diesen Fehlern Standard-Exceptions aus, die Sie im Exception-Bereich des Blockes mit den Exceptions `NO_DATA_FOUND` und `TOO_MANY_ROWS` bearbeiten können. Nehmen Sie eine `WHERE`-Bedingung in die SQL-Anweisung auf, sodass die Anweisung nur eine Zeile zurückgibt. Weitere Informationen zur Verarbeitung von Exceptions erhalten Sie in der Lektion "Exceptions behandeln".

Hinweis: Immer wenn `DBMS_OUTPUT.PUT_LINE` in den Codebeispielen verwendet wird, ist dem Block die Anweisung `SET SERVEROUTPUT ON` vorangestellt.

Mehrere Zeilen aus einer Tabelle abrufen und die Daten bearbeiten

Eine `SELECT`-Anweisung mit der `INTO`-Klausel kann jeweils nur eine Zeile abrufen. Wenn Sie mehrere Zeilen abrufen und die Daten bearbeiten möchten, können Sie explizite Cursor verwenden. Im weiteren Verlauf dieser Lektion erhalten Sie eine allgemeine Einführung in Cursor. Explizite Cursor werden in der Lektion "Explizite Cursor" behandelt.

Daten in PL/SQL abrufen – Beispiel

Werte `hire_date` und `salary` für den angegebenen Mitarbeiter abrufen

```
DECLARE
    v_emp_hiredate    employees.hire_date%TYPE;
    v_emp_salary       employees.salary%TYPE;
BEGIN
    SELECT    hire_date, salary
    INTO      v_emp_hiredate, v_emp_salary
    FROM     employees
    WHERE    employee_id = 100;
    DBMS_OUTPUT.PUT_LINE ('Hire date is :'|| v_emp_hiredate);
    DBMS_OUTPUT.PUT_LINE ('Salary is :'|| v_emp_salary);
END;
/
```

```
anonymous block completed
Hire date is :17-JUN-03
Salary is :24000
```

ORACLE

Copyright © 2014, Oracle und/oder verbundene Unternehmen. All rights reserved. Alle Rechte vorbehalten.

Im Beispiel auf der Folie sind die Variablen `v_emp_hiredate` und `v_emp_salary` im deklarativen Bereich des PL/SQL-Blockes deklariert. Im ausführbaren Bereich werden für den Mitarbeiter mit der Personalnummer (`employee_id`) 100 aus der Tabelle `employee` die Werte der Spalten `hire_date` und `salary` abgerufen. Anschließend werden die Werte in den Variablen `v_emp_hiredate` und `v_emp_salary` gespeichert. Die `INTO`-Klausel ruft in Verbindung mit der `SELECT`-Anweisung die Werte der Datenbankspalte ab und speichert sie in den PL/SQL-Variablen.

Hinweis: Die `SELECT`-Anweisung ruft erst `hire_date` und dann `salary` ab. Die Variablen in der `INTO`-Klausel müssen daher in derselben Reihenfolge angeordnet sein. Wenn Sie beispielsweise `v_emp_hiredate` und `v_emp_salary` in der Anweisung auf der Folie austauschen, führt die Anweisung zu einem Fehler.

Daten in PL/SQL abrufen

Summe der Gehälter aller Mitarbeiter in der angegebenen Abteilung

Beispiel:

```
DECLARE
    v_sum_sal    NUMBER(10,2);
    v_deptno     NUMBER NOT NULL := 60;
BEGIN
    SELECT  SUM(salary) -- group function
    INTO v_sum_sal  FROM employees
    WHERE      department_id = v_deptno;
    DBMS_OUTPUT.PUT_LINE ('The sum of salary is ' || v_sum_sal);
END;
```

```
anonymous block completed
The sum of salary is 28800
```

ORACLE

Copyright © 2014, Oracle und/oder verbundene Unternehmen. All rights reserved. Alle Rechte vorbehalten.

Im Beispiel auf der Folie sind die Variablen `v_sum_sal` und `v_deptno` im deklarativen Bereich des PL/SQL-Blockes deklariert. Im ausführbaren Bereich wird das Gesamtgehalt der Mitarbeiter der Abteilung mit der Abteilungsnummer (`department_id`) 60 mithilfe der SQL-Aggregatfunktion `SUM` berechnet. Das berechnete Gesamtgehalt wird der Variablen `v_sum_sal` zugewiesen.

Hinweis: Sie können in der PL/SQL-Syntax keine Gruppenfunktionen einsetzen. Diese müssen in SQL-Anweisungen innerhalb eines PL/SQL-Blockes verwendet werden (siehe Beispiel auf der Folie).

Gruppenfunktionen dürfen beispielsweise *nicht* mit der folgenden Syntax verwendet werden:

```
V_sum_sal := SUM(employees.salary);
```

Mehrdeutige Namen

```
DECLARE
    hire_date      employees.hire_date%TYPE;
    sysdate        hire_date%TYPE;
    employee_id    employees.employee_id%TYPE := 176;
BEGIN
    SELECT      hire_date, sysdate
    INTO        hire_date, sysdate
    FROM        employees
    WHERE       employee_id = employee_id;
END;
/
```

```
Error report:
ORA-01422: exact fetch returns more than requested number of rows
ORA-06512: at line 6
01422. 00000 - "exact fetch returns more than requested number of rows"
*Cause:    The number specified in exact fetch is less than the rows returned.
*Action:   Rewrite the query or change number of rows requested
```

ORACLE

Copyright © 2014, Oracle und/oder verbundene Unternehmen. All rights reserved. Alle Rechte vorbehalten.

In potenziell mehrdeutigen SQL-Anweisungen haben die Namen von Datenbankspalten Vorrang gegenüber den Namen lokaler Variablen.

Das Beispiel auf der Folie ist wie folgt definiert: Für den Mitarbeiter mit der Personalnummer (`employee_id`) 176 sollen das Einstellungsdatum und das Tagesdatum aus der Tabelle `employees` abgerufen werden. Dabei wird eine unbehandelte Laufzeit-Exception ausgelöst, da die Namen der PL/SQL-Variablen in der `WHERE`-Klausel mit den Namen der Datenbankspalten in der Tabelle `employees` identisch sind.

Die folgende `DELETE`-Anweisung löscht neben dem Mitarbeiter namens "King" auch alle Mitarbeiter in der Tabelle `employees`, deren Nachname nicht `NULL` ist, da der Oracle-Server annimmt, dass sich beide Vorkommen von `last_name` in der `WHERE`-Klausel auf die Datenbankspalte beziehen:

```
DECLARE
    last_name VARCHAR2(25) := 'King';
BEGIN
    DELETE FROM employees WHERE last_name = last_name;
    . . .
```

Benennungskonventionen

- Mehrdeutigkeiten in der WHERE-Klausel mithilfe einer Benennungskonvention vermeiden
- Keine Namen von Datenbankspalten als IDs verwenden
- Syntaxfehler können auftreten, da PL/SQL zuerst nach einer Spalte in der Datenbanktabelle sucht.
- Die Namen lokaler Variablen und formaler Parameter haben Vorrang gegenüber den Namen von Datenbank-*Tabellen*.
- Die Namen von *Spalten* in Datenbanktabellen haben Vorrang gegenüber den Namen lokaler Variablen.
- Die Namen von Variablen haben Vorrang gegenüber den Funktionsnamen.

ORACLE

Copyright © 2014, Oracle und/oder verbundene Unternehmen. All rights reserved. Alle Rechte vorbehalten.

Sie können Mehrdeutigkeiten in der WHERE-Klausel vermeiden, indem Sie Datenbankspalten und PL/SQL-Variablen unterschiedlich benennen.

- Datenbankspalten und IDs müssen verschiedene Namen haben.
- Syntaxfehler können auftreten, da PL/SQL zuerst nach einer Spalte in der Datenbanktabelle sucht.

Agenda

- Daten mit PL/SQL abrufen
- Daten mit PL/SQL bearbeiten
- SQL-Cursor – Einführung

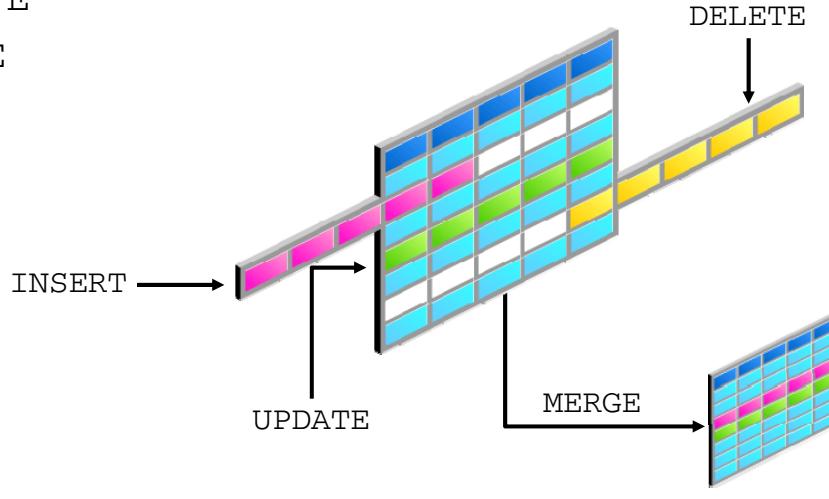


Copyright © 2014, Oracle und/oder verbundene Unternehmen. All rights reserved. Alle Rechte vorbehalten.

Daten mit PL/SQL bearbeiten

Datenbanktabellen mithilfe von DML-Befehlen ändern:

- INSERT
- UPDATE
- DELETE
- MERGE



ORACLE

Copyright © 2014, Oracle und/oder verbundene Unternehmen. All rights reserved. Alle Rechte vorbehalten.

Mit DML-Befehlen lassen sich Daten in der Datenbank bearbeiten. Sie können DML-Befehle wie `INSERT`, `UPDATE`, `DELETE` und `MERGE` ohne Beschränkung in PL/SQL absetzen. Durch Einbindung der `COMMIT`- oder `ROLLBACK`-Anweisungen im PL/SQL-Code werden Zeilensperren (und Tabellensperren) aufgehoben.

- `INSERT`-Anweisungen fügen neue Zeilen in die Tabelle ein.
- `UPDATE`-Anweisungen ändern vorhandene Tabellenzeilen.
- `DELETE`-Anweisungen löschen Zeilen aus der Tabelle.
- `MERGE`-Anweisungen wählen Zeilen in einer Tabelle, um sie in eine andere Tabelle einzufügen oder darin zu aktualisieren. Ob Zeilen in der Zieltabelle aktualisiert oder eingefügt werden, hängt von einer Bedingung in der `ON`-Klausel ab.

Hinweis: `MERGE` ist eine deterministische Anweisung. Dies bedeutet, dass Sie dieselbe Zeile in der Zieltabelle nicht mehrmals mit derselben `MERGE`-Anweisung aktualisieren können. Sie benötigen die Objektberechtigungen `INSERT` und `UPDATE` für die Zieltabelle und die Berechtigung `SELECT` für die Quelltabelle.

Daten einfügen – Beispiel

Der Tabelle EMPLOYEES neue Mitarbeiterinformationen hinzufügen

```
BEGIN
  INSERT INTO employees
  (employee_id, first_name, last_name, email,
   hire_date, job_id, salary)
  VALUES(employees_seq.NEXTVAL, 'Ruth', 'Cores',
    'RCORES', CURRENT_DATE, 'AD_ASST', 4000);
END;
/
```

ORACLE

Copyright © 2014, Oracle und/oder verbundene Unternehmen. All rights reserved. Alle Rechte vorbehalten.

Im Beispiel auf der Folie wird mithilfe einer `INSERT`-Anweisung in einem PL/SQL-Block ein Record in die Tabelle `employees` eingefügt. Wenn Sie in einem PL/SQL-Block den Befehl `INSERT` verwenden, können Sie:

- SQL-Funktionen verwenden, zum Beispiel `USER` und `CURRENT_DATE`
- Primärschlüsselwerte mit vorhandenen Datenbank-Sequences generieren
- Werte im PL/SQL-Block ableiten

Hinweis: Die Daten in der Tabelle `employees` müssen unverändert bleiben. Die Tabelle `employees` ist zwar nicht schreibgeschützt, Sie dürfen jedoch keine Daten einfügen, aktualisieren oder löschen, um eine konsistente Ausgabe zu gewährleisten. Deshalb wird das Befehls-Rollback wie im Code für die Folie 15_sa in `code_ex_05.sql` verwendet.

Daten aktualisieren – Beispiel

Gehalt aller Mitarbeiter erhöhen, die Lagerverwalter sind ("Stock Clerks", ST_CLERK)

```
DECLARE
    sal_increase    employees.salary%TYPE := 800;
BEGIN
    UPDATE      employees
    SET          salary = salary + sal_increase
    WHERE        job_id = 'ST_CLERK';
END;
/
```

```
anonymous block completed
FIRST_NAME      SALARY
-----
Julia           4000
Irene           3500
James           3200
Steven          3000
```

```
...
Curtis          3900
Randall         3400
Peter           3300
```

20 rows selected

ORACLE

Copyright © 2014, Oracle und/oder verbundene Unternehmen. All rights reserved. Alle Rechte vorbehalten.

In der SET-Klausel der UPDATE-Anweisung können mehrdeutige IDs auftreten. Während die ID auf der linken Seite des Zuweisungsoperators immer eine Datenbankspalte ist, kann die ID auf der rechten Seite entweder eine Datenbankspalte oder eine PL/SQL-Variable sein. Wie bereits erläutert, sucht der Oracle-Server bei identischen Spalten- und ID-Namen in der WHERE-Klausel zuerst in der Datenbank nach dem Namen.

Die betroffenen Zeilen bestimmen Sie mit der WHERE-Klausel. Wenn keine Zeilen geändert werden, tritt (anders als bei SELECT-Anweisungen in PL/SQL) kein Fehler auf.

Hinweis: PL/SQL-Variablenzuweisungen verwenden immer einen Doppelpunkt gefolgt von einem Gleichheitszeichen (`:=`) und SQL-Spaltenzuweisungen immer ein Gleichheitszeichen (`=`).

Daten löschen – Beispiel

Zeilen, die zu Abteilung 10 gehören, aus der Tabelle employees löschen

```
DECLARE
    deptno    employees.department_id%TYPE := 10;
BEGIN
    DELETE FROM employees
    WHERE department_id = deptno;
END;
/
```

ORACLE

Copyright © 2014, Oracle und/oder verbundene Unternehmen. All rights reserved. Alle Rechte vorbehalten.

Mit der `DELETE`-Anweisung entfernen Sie nicht benötigte Zeilen aus einer Tabelle. Falls die `WHERE`-Klausel nicht verwendet wird und keine Integritäts-Constraints vorliegen, können Sie alle Zeilen einer Tabelle löschen.

Zeilen zusammenführen

Zeilen in die Tabelle `copy_emp` einfügen oder entsprechend der Tabelle `employees` aktualisieren

```
BEGIN
MERGE INTO copy_emp c
    USING employees e
    ON (e.employee_id = c.empno)
    WHEN MATCHED THEN
        UPDATE SET
            c.first_name      = e.first_name,
            c.last_name       = e.last_name,
            c.email           = e.email,
            . . .
    WHEN NOT MATCHED THEN
        INSERT VALUES(e.employee_id, e.first_name, e.last_name,
                      . . ., e.department_id);
END;
/
```

ORACLE

Copyright © 2014, Oracle und/oder verbundene Unternehmen. All rights reserved. Alle Rechte vorbehalten.

Mit `MERGE`-Anweisungen können Sie Datenzeilen aus einer Tabelle in eine andere Tabelle einfügen oder darin aktualisieren. Jede Zeile wird gemäß einer Equi Join-Bedingung in die Zieltabelle eingefügt oder darin aktualisiert.

Im gezeigten Beispiel wird die Spalte `empno` in der Tabelle `copy_emp` mit der Spalte `employee_id` in der Tabelle `employees` abgeglichen. Wird eine Übereinstimmung gefunden, wird die Zeile in der Tabelle `copy_emp` an die Zeile in der Tabelle `employees` angepasst. Falls die Zeile in der Tabelle `copy_emp` nicht zu finden ist, wird sie eingefügt.

Das komplette Beispiel zur Verwendung der `MERGE`-Anweisung in einem PL/SQL-Block finden Sie auf der nächsten Seite.

```

BEGIN
MERGE INTO copy_emp c
    USING employees e
    ON (e.employee_id = c.empno)
WHEN MATCHED THEN
    UPDATE SET
        c.first_name      = e.first_name,
        c.last_name       = e.last_name,
        c.email           = e.email,
        c.phone_number    = e.phone_number,
        c.hire_date       = e.hire_date,
        c.job_id          = e.job_id,
        c.salary          = e.salary,
        c.commission_pct  = e.commission_pct,
        c.manager_id      = e.manager_id,
        c.department_id   = e.department_id
WHEN NOT MATCHED THEN
    INSERT VALUES(e.employee_id,  e.first_name,  e.last_name,
                  e.email,    e.phone_number, e.hire_date,  e.job_id,
                  e.salary,   e.commission_pct, e.manager_id,
                  e.department_id);
END;
/

```

Agenda

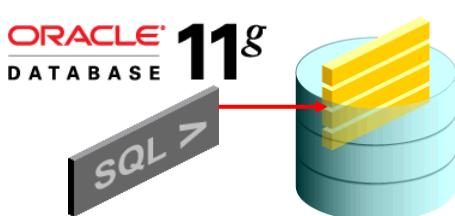
- Daten mit PL/SQL abrufen
- Daten mit PL/SQL bearbeiten
- SQL-Cursor – Einführung



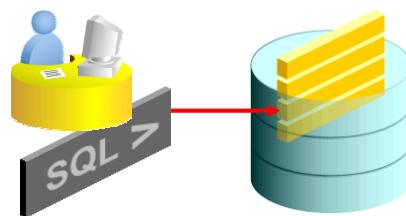
Copyright © 2014, Oracle und/oder verbundene Unternehmen. All rights reserved. Alle Rechte vorbehalten.

SQL-Cursor

- Ein Cursor ist ein Zeiger auf den vom Oracle-Server zugewiesenen privaten Speicherbereich. Cursor dienen zum Verarbeiten der Ergebnismenge von SELECT-Anweisungen.
- Es gibt zwei Cursortypen: implizite und explizite Cursor.
 - **Implizite Cursor:** Werden vom Oracle-Server erstellt und verwaltet, um SQL-Anweisungen zu verarbeiten.
 - **Explizite Cursor:** Werden explizit vom Programmierer deklariert.



Impliziter Cursor



Expliziter Cursor

Copyright © 2014, Oracle und/oder verbundene Unternehmen. All rights reserved. Alle Rechte vorbehalten.

Wie bereits erwähnt, können Sie SQL-Anweisungen, die eine einzelne Zeile zurückgeben, in einen PL/SQL-Block aufnehmen. Die von der SQL-Anweisung abgerufenen Daten werden mit der INTO-Klausel in Variablen platziert.

Wo verarbeitet der Oracle-Server SQL-Anweisungen?

Der Oracle-Server weist einen privaten Speicherbereich zu, der als *Kontextbereich* zur Verarbeitung von SQL-Anweisungen bezeichnet wird. In diesem Bereich werden SQL-Anweisungen geparsert und verarbeitet. Die für die Verarbeitung erforderlichen Informationen sowie die nach der Verarbeitung abgerufenen Informationen werden hier gespeichert. Sie haben keine Kontrolle über diesen Bereich, da er intern vom Oracle-Server verwaltet wird.

Ein Cursor ist ein Zeiger auf den Kontextbereich. In diesem Fall handelt es sich jedoch um einen impliziten Cursor, der automatisch vom Oracle-Server verwaltet wird. PL/SQL erstellt einen impliziten Cursor, wenn der ausführbare Block eine SQL-Anweisung ausgibt.

Cursortypen

Es gibt zwei Cursortypen:

- **Implizite Cursor:** Werden vom Oracle-Server erstellt und verwaltet. Sie können nicht darauf zugreifen. Der Oracle-Server erstellt diese Cursor, wenn er eine SQL-Anweisung ausführen muss.

- **Explizite Cursor:** Als Programmierer müssen Sie möglicherweise mehrere Zeilen aus einer Datenbanktabelle abrufen, einen Zeiger für jede abzurufende Zeile einbinden oder jede Zeile einzeln bearbeiten. In diesen Fällen können Sie Cursor explizit entsprechend Ihren Geschäftsanforderungen deklarieren. Ein von Programmierern deklarierter Cursor wird als *expliziter Cursor* bezeichnet. Sie deklarieren einen solchen Cursor im deklarativen Bereich eines PL/SQL-Blockes.

SQL-Cursorattribute für implizite Cursor

Ergebnis von SQL-Anweisungen mithilfe von SQL-Cursorattributen testen

SQL%FOUND	Boolesches Attribut, das mit TRUE ausgewertet wird, wenn sich die letzte SQL-Anweisung mindestens auf eine Zeile auswirkt
SQL%NOTFOUND	Boolesches Attribut, das mit TRUE ausgewertet wird, wenn sich die letzte SQL-Anweisung auf keine einzige Zeile auswirkt
SQL%ROWCOUNT	Eine Ganzzahl, die die Anzahl von Zeilen angibt, auf die sich die letzte SQL-Anweisung auswirkt



Copyright © 2014, Oracle und/oder verbundene Unternehmen. All rights reserved. Alle Rechte vorbehalten.

Mit SQL-Cursorattributen können Sie evaluieren, was bei der letzten Verwendung eines impliziten Cursors passiert ist. Verwenden Sie diese Attribute nur in PL/SQL-Anweisungen, nicht in SQL-Anweisungen.

Sie können die Attribute `SQL%ROWCOUNT`, `SQL%FOUND` und `SQL%NOTFOUND` im ausführbaren Bereich eines Blockes testen, um nach der Ausführung des entsprechenden DML-Befehls Informationen zu sammeln. PL/SQL gibt keinen Fehler zurück, wenn eine DML-Anweisung keine Zeilen in der zugrunde liegenden Tabelle bearbeitet. PL/SQL gibt jedoch eine Exception zurück, wenn eine SELECT-Anweisung keine Zeilen abruft.

Wie Sie sehen, erhalten die Attribute das Präfix SQL. Da die Cursorattribute mit impliziten Cursorn verwendet werden, die von PL/SQL automatisch erstellt werden und deren Namen Sie nicht kennen, verwenden Sie SQL anstelle des Cursornamens.

Das Attribut `SQL%NOTFOUND` bewirkt das Gegenteil von `SQL%FOUND`. Sie können dieses Attribut als EXIT-Bedingung in einer Schleife verwenden. In UPDATE- und DELETE-Anweisungen ist es hilfreich, wenn sich keine Zeilen ändern, da in diesen Fällen keine Exceptions zurückgegeben werden.

Explizite Cursor werden in der Lektion "Explizite Cursor" erläutert.

SQL-Cursorattribute für implizite Cursor

Zeilen mit der angegebenen Personalnummer aus der Tabelle employees löschen und Anzahl der gelöschten Zeilen anzeigen

Beispiel:

```
DECLARE
    v_rows_deleted VARCHAR2(30);
    v_empno employees.employee_id%TYPE := 176;
BEGIN
    DELETE FROM employees
    WHERE employee_id = v_empno;
    v_rows_deleted := (SQL%ROWCOUNT ||
                       ' row deleted.');
    DBMS_OUTPUT.PUT_LINE (v_rows_deleted);
END;
```



Copyright © 2014, Oracle und/oder verbundene Unternehmen. All rights reserved. Alle Rechte vorbehalten.

Im Beispiel auf der Folie wird eine Zeile mit der Personalnummer (employee_id) 176 aus der Tabelle employees gelöscht. Mit dem Attribut SQL%ROWCOUNT können Sie die Anzahl der gelöschten Zeilen anzeigen.

Quiz

Bei Verwendung der SELECT-Anweisung in PL/SQL ist die INTO-Klausel obligatorisch, und Abfragen können eine oder mehrere Zeilen zurückgeben.

- a. Richtig
- b. Falsch



Copyright © 2014, Oracle und/oder verbundene Unternehmen. All rights reserved. Alle Rechte vorbehalten.

Richtige Antwort: b

INTO-Klausel

Die INTO-Klausel ist obligatorisch. Sie steht zwischen der SELECT- und der FROM-Klausel. Sie gibt die Namen der Variablen an, die von SQL durch die SELECT-Klausel zurückgegebenen Werte aufnehmen. Sie müssen zu jedem gewählten Element eine Variable angeben, wobei die Reihenfolge der Variablen mit den gewählten Elementen übereinstimmen muss.

Füllen Sie PL/SQL-Variablen oder Hostvariablen mithilfe der INTO-Klausel.

Abfragen dürfen nur eine Zeile zurückgeben

Die SELECT-Anweisungen in einem PL/SQL-Block fallen in die ANSI-Klassifizierung der eingebetteten SQL-Anweisungen, für die folgende Regel gilt: Abfragen dürfen nur eine Zeile zurückgeben. Gibt eine Abfrage mehr als eine Zeile oder gar keine Zeile zurück, wird ein Fehler generiert.

PL/SQL löst bei diesen Fehlern Standard-Exceptions aus, die Sie im Exception-Bereich des Blockes mit den Exceptions NO_DATA_FOUND und TOO_MANY_ROWS bearbeiten können. Nehmen Sie eine WHERE-Bedingung in die SQL-Anweisung auf, sodass die Anweisung nur eine Zeile zurückgibt. Auf die Exception-Behandlung wird im Verlauf dieses Kurses noch eingegangen.

Zusammenfassung

Nach Ablauf dieser Lektion haben Sie folgende Ziele erreicht:

- DML-Anweisungen, Anweisungen zur Transaktionskontrolle und DDL-Anweisungen in PL/SQL einbetten
- INTO-Klausel verwenden, die für alle SELECT-Anweisungen in PL/SQL obligatorisch ist
- Zwischen impliziten und expliziten Cursorn unterscheiden
- Ergebnis von SQL-Anweisungen mithilfe von SQL-Cursorattributen bestimmen



Copyright © 2014, Oracle und/oder verbundene Unternehmen. All rights reserved. Alle Rechte vorbehalten.

DML-Befehle und Anweisungen zur Transaktionskontrolle können in PL/SQL-Programmen uneingeschränkt verwendet werden. DDL-Befehle sind jedoch nicht direkt verwendbar.

Eine SELECT-Anweisung in einem PL/SQL-Block kann nur eine Zeile zurückgeben. Die INTO-Klausel ist erforderlich, um die mit der SELECT-Anweisung abgerufenen Werte aufzunehmen.

Ein Cursor ist ein Zeiger auf den Speicherbereich. Es gibt zwei Typen von Cursorn. Implizite Cursor werden vom Oracle-Server erstellt und verwaltet, um SQL-Anweisungen auszuführen. Sie können diese Cursor gemeinsam mit SQL-Cursorattributen verwenden, um das Ergebnis der SQL-Anweisung zu bestimmen. Explizite Cursor werden von Programmierern deklariert.

Übungen zu Lektion 5 – Übersicht

Diese Übung behandelt folgende Themen:

- Daten aus einer Tabelle wählen
- Daten in eine Tabelle einfügen
- Daten in einer Tabelle aktualisieren
- Records aus einer Tabelle löschen



Copyright © 2014, Oracle und/oder verbundene Unternehmen. All rights reserved. Alle Rechte vorbehalten.

Kontrollstrukturen erstellen

ORACLE

Copyright © 2014, Oracle und/oder verbundene Unternehmen. All rights reserved. Alle Rechte vorbehalten.

Ziele

Nach Ablauf dieser Lektion haben Sie folgende Ziele erreicht:

- Verwendung und Typen von Kontrollstrukturen identifizieren
- IF-Anweisungen erstellen
- CASE-Anweisungen und CASE-Ausdrücke erstellen
- Schleifenanweisungen erstellen und identifizieren
- Richtlinien für bedingte Kontrollstrukturen anwenden

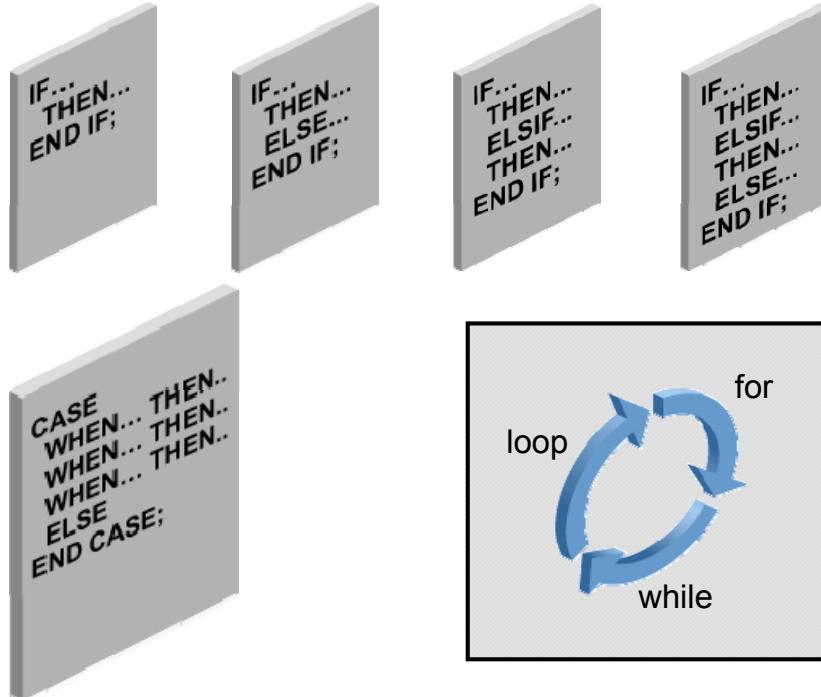


Copyright © 2014, Oracle und/oder verbundene Unternehmen. All rights reserved. Alle Rechte vorbehalten.

Sie haben gelernt, wie Sie PL/SQL-Blöcke mit deklarativen und ausführbaren Bereichen erstellen. Zudem wurde erläutert, wie Sie Ausdrücke und SQL-Anweisungen in den ausführbaren Block aufnehmen.

In dieser Lektion wird beschrieben, wie Sie Kontrollstrukturen wie IF-Anweisungen, CASE-Ausdrücke und LOOP-Strukturen in PL/SQL-Blöcken verwenden.

Ablauf der Ausführung steuern



ORACLE

Copyright © 2014, Oracle und/oder verbundene Unternehmen. All rights reserved. Alle Rechte vorbehalten.

Sie können den logischen Ablauf von Anweisungen innerhalb des PL/SQL-Blockes mithilfe verschiedener Kontrollstrukturen ändern. In dieser Lektion werden vier Typen von PL/SQL-Kontrollstrukturen erläutert: Bedingungskonstrukte mit IF-Anweisungen, CASE-Ausdrücke, LOOP-Kontrollstrukturen und CONTINUE-Anweisungen.

Agenda

- **IF-Anweisungen**
- **CASE-Anweisungen und CASE-Ausdrücke**
- **Schleifenanweisungen erstellen und identifizieren**

ORACLE

Copyright © 2014, Oracle und/oder verbundene Unternehmen. All rights reserved. Alle Rechte vorbehalten.

IF-Anweisungen

Syntax:

```
IF condition THEN
    statements;
[ELSIF condition THEN
    statements;]
[ELSE
    statements;]
END IF;
```

ORACLE

Copyright © 2014, Oracle und/oder verbundene Unternehmen. All rights reserved. Alle Rechte vorbehalten.

Die Struktur der PL/SQL-Anweisung `IF` ähnelt der Struktur von `IF`-Anweisungen in anderen prozeduralen Sprachen. In PL/SQL können Sie damit Aktionen selektiv auf Grundlage von Bedingungen ausführen.

Für die Syntax gilt:

<code>condition</code>	Boolesche Variable oder boolescher Ausdruck, die/der TRUE, FALSE oder NULL zurückgibt
<code>THEN</code>	Leitet eine Klausel ein, die den booleschen Ausdruck mit der nachfolgenden Anweisungsfolge verknüpft
<code>statements</code>	Steht für eine oder mehrere PL/SQL- oder SQL-Anweisungen. (Sie können zusätzliche <code>IF</code> -Anweisungen mit mehreren verschachtelten <code>IF</code> , <code>ELSE</code> und <code>ELSIF</code> -Anweisungen enthalten.) Die Anweisungen in der <code>THEN</code> -Klausel werden nur ausgeführt, wenn die zugehörige <code>IF</code> -Klausel die Bedingung mit TRUE ausgewertet wird.

Für die Syntax gilt:

- ELSIF Schlüsselwort, das einen booleschen Ausdruck einleitet. (Wenn die erste Bedingung FALSE oder NULL ergibt, leitet das Schlüsselwort ELSIF zusätzliche Bedingungen ein.)
- ELSE Führt die Standardklausel ein, die nur dann ausgeführt wird, wenn keines der zuvor mit IF und ELSIF eingeleiteten Prädikate TRUE lautet. Die Prüfungen erfolgen der Reihe nach, sodass ein früheres Prädikat mit dem Wert TRUE einem späteren Prädikat mit TRUE vorgezogen wird.
- END IF Markiert das Ende einer IF-Anweisung

Hinweis: ELSIF und ELSE sind in IF-Anweisungen optional. In IF-Anweisungen können Sie eine beliebige Anzahl von ELSIF-Schlüsselwörtern verwenden. Es ist aber nur ein ELSE-Schlüsselwort zulässig. END IF markiert das Ende einer IF-Anweisung und muss durch ein Semikolon abgeschlossen werden.

Einfache IF-Anweisungen

```
DECLARE
    v_myage  number:=31;
BEGIN
    IF v_myage  < 11
    THEN
        DBMS_OUTPUT.PUT_LINE(' I am a child ');
    END IF;
END;
/
```

```
anonymous block completed
```

ORACLE

Copyright © 2014, Oracle und/oder verbundene Unternehmen. All rights reserved. Alle Rechte vorbehalten.

Beispiel für einfache IF-Anweisung

Das Beispiel auf der Folie zeigt eine einfache IF-Anweisung mit der THEN-Klausel.

- Die Variable `v_myage` wird auf 31 initialisiert.
- Die Bedingung für die IF-Anweisung gibt FALSE zurück, da `v_myage` nicht kleiner als 11 ist.
- Aus diesem Grund geht die Kontrolle niemals auf die THEN-Klausel über.

Bedingungsausdrücke hinzufügen

Eine IF-Anweisung kann mehrere Bedingungsausdrücke enthalten, die mit logischen Operatoren wie AND, OR und NOT verknüpft sind.

Beispiel:

```
IF (myfirstname='Christopher' AND v_myage <11)
...
...
```

Die Bedingung verwendet den Operator AND und ist daher nur dann TRUE, wenn beide Bedingungen TRUE sind. Sie können beliebig viele bedingte Ausdrücke verwenden. Die Anweisungen müssen jedoch mit den entsprechenden logischen Operatoren verbunden sein.

IF THEN ELSE-Anweisungen

```
DECLARE
    v_myage  number:=31;
BEGIN
    IF
        v_myage < 11 THEN
            DBMS_OUTPUT.PUT_LINE(' I am a child ');
    ELSE
        DBMS_OUTPUT.PUT_LINE(' I am not a child ');
    END IF;
END;
/
```

```
anonymous block completed
I am not a child
```

ORACLE

Copyright © 2014, Oracle und/oder verbundene Unternehmen. All rights reserved. Alle Rechte vorbehalten.

Auf dieser Folie wurde dem Code die ELSE-Klausel hinzugefügt. Die Bedingung ist gleich geblieben und daher noch immer FALSE. Die Anweisungen in der THEN-Klausel werden nur ausgeführt, wenn die Bedingung TRUE ergibt. In diesem Fall gibt die Bedingung FALSE zurück, und es wird mit der ELSE-Anweisung fortgefahrt.

Die Ausgabe des Blockes ist unterhalb des Codes dargestellt.

IF ELSIF ELSE-Klauseln

```
DECLARE
    v_myage number:=31;
BEGIN
    IF v_myage < 11 THEN
        DBMS_OUTPUT.PUT_LINE(' I am a child ');
    ELSIF v_myage < 20 THEN
        DBMS_OUTPUT.PUT_LINE(' I am young ');
    ELSIF v_myage < 30 THEN
        DBMS_OUTPUT.PUT_LINE(' I am in my twenties');
    ELSIF v_myage < 40 THEN
        DBMS_OUTPUT.PUT_LINE(' I am in my thirties');
    ELSE
        DBMS_OUTPUT.PUT_LINE(' I am always young ');
    END IF;
END;
/
```

```
anonymous block completed
I am in my thirties
```

ORACLE

Copyright © 2014, Oracle und/oder verbundene Unternehmen. All rights reserved. Alle Rechte vorbehalten.

Die IF-Klausel kann mehrere ELSIF-Klauseln und eine ELSE-Klausel enthalten. Das Beispiel veranschaulicht die folgenden Eigenschaften dieser Klauseln:

- Die ELSIF-Klauseln können im Gegensatz zur ELSE-Klausel Bedingungen aufweisen.
- Auf die Bedingung für ELSIF muss die THEN-Klausel folgen, die dann ausgeführt wird, wenn die Bedingung für ELSIF als TRUE ausgewertet wird.
- Wenn Sie mehrere ELSIF-Klauseln haben und die erste Bedingung FALSE oder NULL ist, geht die Kontrolle auf die nächste ELSIF-Klausel über.
- Bedingungen werden von oben beginnend der Reihe nach ausgewertet.
- Wenn alle Bedingungen FALSE oder NULL sind, werden die Anweisungen in der ELSE-Klausel ausgeführt.
- Die abschließende ELSE-Klausel ist optional.

Im Beispiel ist die Ausgabe des Blockes unterhalb des Codes dargestellt.

NULL-Werte in IF-Anweisungen

```
DECLARE
    v_myage  number;
BEGIN
    IF v_myage  < 11 THEN
        DBMS_OUTPUT.PUT_LINE(' I am a child ');
    ELSE
        DBMS_OUTPUT.PUT_LINE(' I am not a child ');
    END IF;
END;
/
```

```
anonymous block completed
I am not a child
```

ORACLE

Copyright © 2014, Oracle und/oder verbundene Unternehmen. All rights reserved. Alle Rechte vorbehalten.

Im Beispiel auf der Folie wird die Variable `v_myage` deklariert, aber nicht initialisiert. Die Bedingung der `IF`-Anweisung gibt anstatt `TRUE` oder `FALSE` den Wert `NULL` zurück. In diesem Fall wird mit der `ELSE`-Anweisung fortgefahrene.

Richtlinien

- Aktionen können selektiv auf Grundlage von erfüllten Bedingungen ausgeführt werden.
- Beachten Sie beim Erstellen von Code die Schreibweise der Schlüsselwörter:
 - `ELSIF` in einem Wort
 - `END IF` in zwei Wörtern
- Wenn die boolesche Kontrollbedingung `TRUE` ergibt, wird die zugeordnete Anweisungsfolge ausgeführt. Ist die Bedingung `FALSE` oder `NULL`, wird die entsprechende Anweisungsfolge übersprungen. Sie können beliebig viele `ELSIF`-Klauseln verwenden.
- Rücken Sie die bedingt auszuführenden Anweisungen ein, um die Übersichtlichkeit zu verbessern.

Agenda

- **IF-Anweisungen**
- **CASE-Anweisungen und CASE-Ausdrücke**
- Schleifenanweisungen erstellen und identifizieren

ORACLE

Copyright © 2014, Oracle und/oder verbundene Unternehmen. All rights reserved. Alle Rechte vorbehalten.

CASE-Ausdrücke

- CASE-Ausdrücke wählen ein Ergebnis und geben es zurück.
- Zur Auswahl des Ergebnisses verwendet der CASE-Ausdruck mehrere Ausdrücke. Anhand des von diesen Ausdrücken zurückgegeben Wertes wird eine von mehreren Alternativen gewählt.

```
CASE selector
  WHEN expression1 THEN result1
    [WHEN expression2 THEN result2
      ...
      WHEN expressionN THEN resultN]
    [ELSE resultN+1]
  END;
```



Copyright © 2014, Oracle und/oder verbundene Unternehmen. All rights reserved. Alle Rechte vorbehalten.

Ein CASE-Ausdruck gibt ein Ergebnis zurück, das aus einer oder mehreren Alternativen ermittelt wird. Zur Auswahl des Ergebnisses verwendet der CASE-Ausdruck einen *Selector*. Ein Selector ist ein Ausdruck, dessen Wert verwendet wird, um aus mehreren Alternativen ein Ergebnis zurückzugeben. Dem Selector folgen eine oder mehrere WHEN-Klauseln, die der Reihe nach geprüft werden. Der Wert des Selectors bestimmt, welches Ergebnis zurückgegeben wird. Entspricht der Wert des Selectors dem Wert des Ausdrucks einer WHEN-Klausel, wird diese WHEN-Klausel ausgeführt und das entsprechende Ergebnis zurückgegeben.

PL/SQL stellt auch einen Searched CASE-Ausdruck bereit:

```
CASE
  WHEN search_condition1 THEN result1
    [WHEN search_condition2 THEN result2
      ...
      WHEN search_conditionN THEN resultN]
    [ELSE resultN+1]
  END;
```

Searched CASE-Ausdrücke haben keinen Selector. Die WHEN-Klauseln in den CASE-Ausdrücken enthalten keine Ausdrücke, die Werte eines beliebigen Typs zurückgeben, sondern Suchbedingungen, die einen booleschen Wert ergeben.

CASE-Ausdrücke – Beispiel

```
SET VERIFY OFF
DECLARE
    v_grade  CHAR(1) := UPPER('&grade');
    v_appraisal VARCHAR2(20);
BEGIN
    v_appraisal := CASE v_grade
        WHEN 'A' THEN 'Excellent'
        WHEN 'B' THEN 'Very Good'
        WHEN 'C' THEN 'Good'
        ELSE 'No such grade'
    END;
    DBMS_OUTPUT.PUT_LINE ('Grade: '|| v_grade || 
                          'Appraisal' || v_appraisal);
END;
/
```

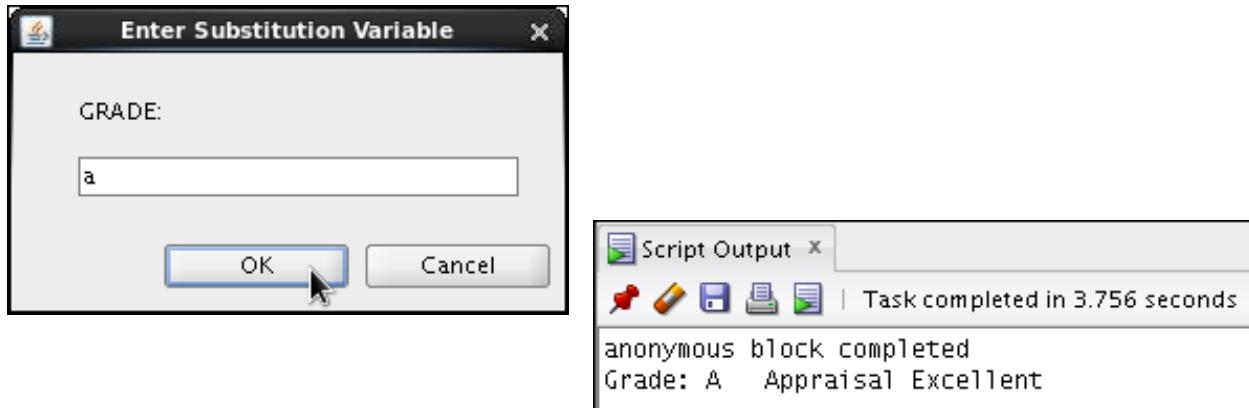
ORACLE

Copyright © 2014, Oracle und/oder verbundene Unternehmen. All rights reserved. Alle Rechte vorbehalten.

Im Beispiel auf der Folie verwendet der CASE-Ausdruck den Wert der Variablen `v_grade` als Ausdruck. Dieser Wert wird vom Benutzer mithilfe einer Substitutionsvariablen angenommen. Der CASE-Ausdruck gibt entsprechend dem vom Benutzer eingegebenen Wert den Wert der Variablen `v_appraisal` zurück, der auf dem Wert der Variablen `v_grade` basiert.

Ergebnis

Wenn Sie für `v_grade` den Wert `a` oder `A` eingeben (siehe Fenster), lautet die Ausgabe im Beispiel wie folgt:



Searched CASE-Ausdrücke

```
DECLARE
    v_grade  CHAR(1) := UPPER('&grade');
    v_appraisal VARCHAR2(20);
BEGIN
    v_appraisal := CASE
        WHEN v_grade = 'A' THEN 'Excellent'
        WHEN v_grade IN ('B', 'C') THEN 'Good'
        ELSE 'No such grade'
    END;
    DBMS_OUTPUT.PUT_LINE ('Grade: ' || v_grade || 
                          ' Appraisal ' || v_appraisal);
END;
/
```

ORACLE

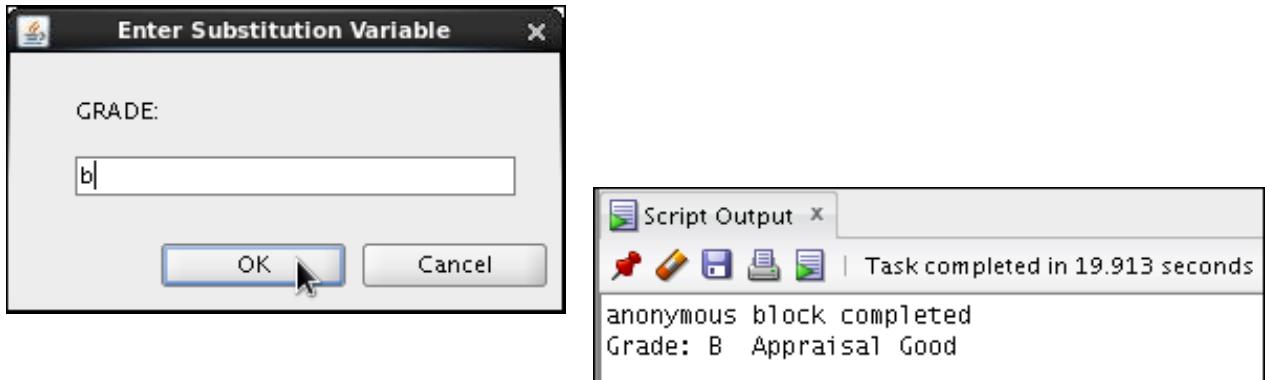
Copyright © 2014, Oracle und/oder verbundene Unternehmen. All rights reserved. Alle Rechte vorbehalten.

Das vorangegangene Beispiel enthielt einen einzelnen Testausdruck, nämlich die Variable `v_grade`. Die `WHEN`-Klausel vergleicht einen Wert mit diesem Testausdruck.

Searched CASE-Anweisungen enthalten keinen Testausdruck. Stattdessen enthält die `WHEN`-Klausel einen Ausdruck, der einen booleschen Wert ergibt. Auf dieser Folie sehen Sie das gleiche Beispiel, diesmal mit Searched CASE-Anweisungen.

Ergebnis

Wenn Sie für `v_grade` den Wert `b` oder `B` eingeben, lautet die Ausgabe im Beispiel wie folgt:



CASE-Anweisungen

```
DECLARE
    v_deptid NUMBER;
    v_deptname VARCHAR2(20);
    v_emps NUMBER;
    v_mngid NUMBER:= 108;
BEGIN
    CASE  v_mngid
        WHEN  108 THEN
            SELECT department_id, department_name
            INTO v_deptid, v_deptname FROM departments
            WHERE manager_id=108;
            SELECT count(*) INTO v_emps FROM employees
            WHERE department_id=v_deptid;
        WHEN  200 THEN
            ...
    END CASE;
    DBMS_OUTPUT.PUT_LINE ('You are working in the '|| v_deptname|||
    ' department. There are '||v_emps ||' employees in this
    department');
END;
/
```

ORACLE

Copyright © 2014, Oracle und/oder verbundene Unternehmen. All rights reserved. Alle Rechte vorbehalten.

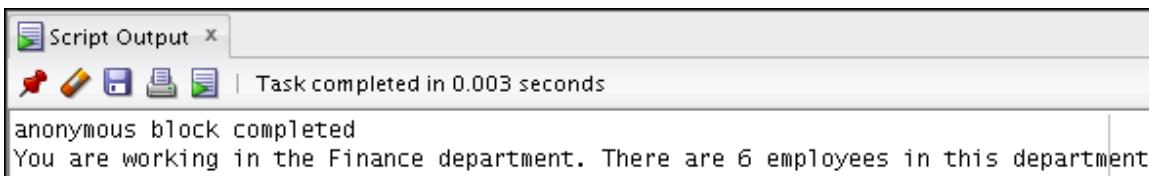
Rufen Sie sich ins Gedächtnis zurück, wie IF-Anweisungen verwendet werden. Sie können in THEN- und ELSE-Klauseln *n* PL/SQL-Anweisungen aufnehmen. In ähnlicher Weise lassen sich Anweisungen in die CASE-Anweisung einbinden, was die Lesbarkeit im Vergleich zu mehreren IF- und ELSIF-Anweisungen erhöht.

Unterschiede zwischen CASE-Ausdrücken und CASE-Anweisungen

CASE-Ausdrücke werten die Bedingung aus und geben einen Wert zurück, während CASE-Anweisungen die Bedingung auswerten und eine Aktion ausführen. Eine CASE-Anweisung kann ein kompletter PL/SQL-Block sein.

- CASE-Anweisungen enden mit END CASE;.
- CASE-Ausdrücke enden mit END;.

Das Beispiel auf der Folie generiert folgende Ausgabe:



The screenshot shows the 'Script Output' window from Oracle SQL Developer. It displays the following text:
anonymous block completed
You are working in the Finance department. There are 6 employees in this department

Hinweis: Während IF-Anweisungen ohne Auswirkung bleiben können (die Bedingungen können alle FALSE sein, und die ELSE Klausel ist nicht obligatorisch), müssen CASE-Anweisungen PL/SQL-Anweisungen ausführen.

NULL-Werte behandeln

Beim Arbeiten mit NULL-Werten können Sie einige häufige Fehler vermeiden, indem Sie die folgenden Regeln beachten:

- Einfache Vergleiche mit NULL-Werten ergeben stets NULL.
- Wenn Sie den logischen Operator NOT auf einen NULL-Wert anwenden, lautet das Ergebnis NULL.
- Wenn die Bedingung in IF-Anweisungen NULL ergibt, wird die zugehörige Anweisungsfolge nicht ausgeführt.



Copyright © 2014, Oracle und/oder verbundene Unternehmen. All rights reserved. Alle Rechte vorbehalten.

Beispiel:

```
x := 5;
y := NULL;
...
IF x != y THEN -- yields NULL, not TRUE
    -- sequence_of_statements that are not executed
END IF;
```

Möglicherweise erwarten Sie, dass die Anweisungsfolge ausgeführt wird, da `x` ungleich `y` zu sein scheint. NULL-Werte sind jedoch unbestimmt. Es ist nicht bekannt, ob `x` gleich `y` ist. Aus diesem Grund wird die IF-Bedingung mit NULL ausgewertet und die Anweisungsfolge umgangen.

```
a := NULL;
b := NULL;
...
IF a = b THEN -- yields NULL, not TRUE
    -- sequence_of_statements that are not executed
END IF;
```

Im zweiten Beispiel erwarten Sie möglicherweise, dass die Anweisungsfolge ausgeführt wird, da `a` gleich `b` zu sein scheint. Doch auch hier ist die Gleichheit unbekannt, sodass die IF-Bedingung mit NULL ausgewertet und die Anweisungsfolge umgangen wird.

Logiktabellen

Einfache boolesche Bedingung mit einem Vergleichsoperator erstellen

AND	TRUE	FALSE	NULL	OR	TRUE	FALSE	NULL	NOT	
TRUE	TRUE	FALSE	NULL	TRUE	TRUE	TRUE	TRUE	TRUE	FALSE
FALSE	FALSE	FALSE	FALSE	FALSE	TRUE	FALSE	NULL	FALSE	TRUE
NULL	NULL	FALSE	NULL	NULL	TRUE	NULL	NULL	NULL	NULL

ORACLE

Copyright © 2014, Oracle und/oder verbundene Unternehmen. All rights reserved. Alle Rechte vorbehalten.

Sie können eine einfache boolesche Bedingung erstellen, indem Sie Zahlen-, Zeichenfolgen- und Datumsausdrücke mit Vergleichsoperatoren kombinieren.

Zur Erstellung komplexer boolescher Bedingungen kombinieren Sie einfache boolesche Bedingungen mit den logischen Operatoren AND, OR und NOT. Mit den logischen Operatoren werden die booleschen Variablenwerte geprüft, die TRUE, FALSE oder NULL zurückgeben.

Für die Logiktabellen auf der Folie gilt:

- FALSE hat in einer AND-Bedingung Vorrang. In einer OR-Bedingung hat TRUE Vorrang.
- AND gibt nur dann TRUE zurück, wenn beide Operanden TRUE sind.
- OR gibt nur dann FALSE zurück, wenn beide Operanden FALSE sind.
- NULL AND TRUE ist immer NULL, da nicht bekannt ist, ob der zweite Operand TRUE ist.

Hinweis: Die Negation von NULL (NOT NULL) ergibt einen NULL-Wert, weil NULL-Werte unbestimmt sind.

Boolescher Ausdruck oder logischer Ausdruck?

Wie lautet der Wert von flag in den einzelnen Fällen?

```
flag := reorder_flag AND available_flag;
```

REORDER_FLAG	AVAILABLE_FLAG	FLAG
TRUE	TRUE	? (1)
TRUE	FALSE	? (2)
NULL	TRUE	? (3)
NULL	FALSE	? (4)

ORACLE

Copyright © 2014, Oracle und/oder verbundene Unternehmen. All rights reserved. Alle Rechte vorbehalten.

Mit der Logiktabelle für AND können Sie die möglichen Ergebnisse der booleschen Bedingung auf der Folie auswerten.

Antworten

1. TRUE
2. FALSE
3. NULL
4. FALSE

Agenda

- IF-Anweisungen
- CASE-Anweisungen und CASE-Ausdrücke
- Schleifenanweisungen erstellen und identifizieren

ORACLE

Copyright © 2014, Oracle und/oder verbundene Unternehmen. All rights reserved. Alle Rechte vorbehalten.

Iterative Kontrollstrukturen – LOOP-Anweisungen

- Schleifen wiederholen eine oder mehrere Anweisungen.
- Es gibt drei Schleifentypen:
 - Basisschleifen
 - FOR-Schleifen
 - WHILE-Schleifen



ORACLE

Copyright © 2014, Oracle und/oder verbundene Unternehmen. All rights reserved. Alle Rechte vorbehalten.

PL/SQL bietet verschiedene Möglichkeiten, Schleifen so zu strukturieren, dass eine oder mehrere Anweisungen wiederholt werden. Mit Schleifen werden Anweisungen wiederholt ausgeführt, bis eine `EXIT`-Bedingung erreicht ist. Schleifen müssen eine `EXIT`-Bedingung enthalten, da sie sonst unendlich sind.

Schleifenkonstrukte sind der dritte Typ von Kontrollstrukturen. PL/SQL bietet die folgenden Schleifentypen:

- Basisschleifen für wiederholte Aktionen ohne allgemeine Bedingungen
- FOR-Schleifen für iterative Aktionen auf Basis eines Zählers
- WHILE-Schleifen für iterative Aktionen auf Basis einer Bedingung

Hinweis: Sie können Schleifen mit einer `EXIT`-Anweisung abschließen. Basisschleifen müssen eine `EXIT`-Anweisung enthalten. Die Cursor FOR-Schleife (ein weiterer FOR-Schleifentyp) wird in der Lektion "Explizite Cursor" behandelt.

Basisschleifen

Syntax:

```
LOOP
    statement1;
    . . .
    EXIT [WHEN condition];
END LOOP;
```



Copyright © 2014, Oracle und/oder verbundene Unternehmen. All rights reserved. Alle Rechte vorbehalten.

Die einfachste Form einer `LOOP`-Anweisung ist die Basisschleife, die eine Anweisungsfolge zwischen die Schlüsselwörter `LOOP` und `END LOOP` setzt. Jedes Mal, wenn der Ausführungsablauf die Anweisung `END LOOP` erreicht, wird die Kontrolle an die entsprechende darüber liegende `LOOP`-Anweisung zurückgegeben. Mit einer Basisschleife können Sie die zugehörigen Anweisungen mindestens ein Mal ausführen, selbst dann, wenn die `EXIT`-Bedingung beim Aufrufen der Schleife bereits erfüllt ist. Ohne eine `EXIT`-Anweisung wäre die Schleife unendlich.

EXIT-Anweisungen

Mit der `EXIT`-Anweisung wird eine Schleife abgeschlossen. Die Kontrolle geht auf die nächste Anweisung nach der Anweisung `END LOOP` über. Sie können die `EXIT`-Anweisung entweder als Aktion innerhalb einer `IF`-Anweisung oder als eigenständige Anweisung innerhalb der Schleife ausgeben. Die `EXIT`-Anweisung muss sich innerhalb einer Schleife befinden. In letzterem Fall können Sie eine `WHEN`-Klausel anhängen, um den bedingten Abschluss der Schleife zu ermöglichen. Bei Erreichen der `EXIT`-Anweisung wird die Bedingung in der `WHEN`-Klausel ausgewertet. Wenn die Bedingung `NULL` ergibt, wird die Schleife umgangen, und die nächste Anweisung nach der Schleife wird ausgeführt.

Basisschleifen können mehrere `EXIT`-Anweisungen enthalten, es wird jedoch empfohlen, sich auf einen `EXIT`-Punkt zu beschränken.

Basisschleifen – Beispiel

```
DECLARE
    v_countryid      locations.country_id%TYPE := 'CA';
    v_loc_id         locations.location_id%TYPE;
    v_counter        NUMBER(2) := 1;
    v_new_city       locations.city%TYPE := 'Montreal';
BEGIN
    SELECT MAX(location_id) INTO v_loc_id FROM locations
    WHERE country_id = v_countryid;
    LOOP
        INSERT INTO locations(location_id, city, country_id)
        VALUES((v_loc_id + v_counter), v_new_city, v_countryid);
        v_counter := v_counter + 1;
        EXIT WHEN v_counter > 3;
    END LOOP;
END;
/
```

ORACLE

Copyright © 2014, Oracle und/oder verbundene Unternehmen. All rights reserved. Alle Rechte vorbehalten.

Das auf der Folie gezeigte Beispiel einer Basisschleife ist wie folgt definiert: "Für den Ländercode CA und die Stadt Montreal sollen drei neue Standortnummern eingefügt werden."

Hinweise

- Die Anweisungen einer Basisschleife werden ausgeführt, bis die Bedingung unter `EXIT WHEN` erfüllt ist.
- Ist die Bedingung so in der Schleife angeordnet, dass sie erst geprüft wird, nachdem die Schleifenanweisungen ausgeführt wurden, wird die Schleife mindestens einmal ausgeführt.
- Wenn die `EXIT`-Bedingung jedoch vor allen anderen ausführbaren Anweisungen der Schleife platziert wird und `TRUE` ist, wird die Schleife beendet, und die Anweisungen werden nicht ausgeführt.

Ergebnisse

Führen Sie zur Anzeige der Ausgabe das Codebeispiel zu Folie 22_sa in `code_ex_06.sql` aus.

WHILE-Schleifen

Syntax:

```
WHILE condition LOOP
    statement1;
    statement2;
    . . .
END LOOP;
```

Mit WHILE-Schleifen können Sie Anweisungen solange wiederholen, wie eine Bedingung TRUE ist.

ORACLE

Copyright © 2014, Oracle und/oder verbundene Unternehmen. All rights reserved. Alle Rechte vorbehalten.

Mit WHILE-Schleifen können Sie eine Anweisungsfolge wiederholen, bis die Kontrollbedingung nicht mehr TRUE ist. Die Bedingung wird am Anfang jeder Iteration ausgewertet. Die Schleife wird abgeschlossen, wenn die Bedingung FALSE oder NULL ergibt. Ist die Bedingung zu Beginn der Schleife FALSE oder NULL, werden keine weiteren Iterationen ausgeführt. Es kann daher vorkommen, dass keine der Anweisungen in der Schleife ausgeführt wird.

Für die Syntax gilt:

condition Boolesche Variable oder boolescher Ausdruck (TRUE, FALSE oder NULL)
statement Steht für eine oder mehrere PL/SQL- oder SQL-Anweisungen

Wenn sich die in den Bedingungen enthaltenen Variablen innerhalb der Schleife nicht verändern, bleibt die Bedingung TRUE, und die Schleife wird nicht abgeschlossen.

Hinweis: Wenn die Bedingung NULL ergibt, wird die Schleife umgangen, und die nächste Anweisung wird ausgeführt.

WHILE-Schleifen – Beispiel

```
DECLARE
    v_countryid    locations.country_id%TYPE := 'CA';
    v_loc_id        locations.location_id%TYPE;
    v_new_city      locations.city%TYPE := 'Montreal';
    v_counter       NUMBER := 1;
BEGIN
    SELECT MAX(location_id) INTO v_loc_id FROM locations
    WHERE country_id = v_countryid;
    WHILE v_counter <= 3 LOOP
        INSERT INTO locations(location_id, city, country_id)
        VALUES((v_loc_id + v_counter), v_new_city, v_countryid);
        v_counter := v_counter + 1;
    END LOOP;
END;
/
```

ORACLE

Copyright © 2014, Oracle und/oder verbundene Unternehmen. All rights reserved. Alle Rechte vorbehalten.

Im Beispiel auf der Folie werden drei neue Standortnummern für den Ländercode CA und die Stadt Montreal hinzugefügt.

- Mit jeder Iteration der WHILE-Schleife wird ein Zähler (`v_counter`) um 1 erhöht.
- Wenn die Anzahl der Iterationen kleiner oder gleich der Zahl 3 ist, wird der Code innerhalb der Schleife ausgeführt, und es wird eine Zeile in die Tabelle `locations` eingefügt.
- Wenn `v_counter` die Anzahl neuer Standorte für diese Stadt und dieses Land überschreitet, wird die Bedingung der Schleife mit `FALSE` ausgewertet, und die Schleife wird abgeschlossen.

Ergebnisse

Führen Sie zur Anzeige der Ausgabe das Codebeispiel zu Folie 24_sa in `code_ex_06.sql` aus.

FOR-Schleifen

- Mit FOR-Schleifen die Prüfung der Anzahl von Iterationen abkürzen
- Zähler nicht deklarieren, da er implizit deklariert wird

```
FOR counter IN [REVERSE]
    lower_bound..upper_bound LOOP
    statement1;
    statement2;
    . . .
END LOOP;
```

ORACLE

Copyright © 2014, Oracle und/oder verbundene Unternehmen. All rights reserved. Alle Rechte vorbehalten.

FOR-Schleifen haben die gleiche allgemeine Struktur wie Basisschleifen. Zudem enthalten sie vor dem Schlüsselwort LOOP eine Kontrollanweisung, mit der die Anzahl der von PL/SQL ausgeführten Iterationen festgelegt wird.

Für die Syntax gilt:

<i>counter</i>	Implizit deklarierte Ganzzahl, deren Wert bei jeder Iteration der Schleife automatisch um 1 erhöht oder (bei Verwendung des Schlüsselwortes REVERSE) herabgesetzt wird, bis die Unter- oder Obergrenze erreicht ist
REVERSE	Bewirkt, dass der Zähler, ausgehend von der Obergrenze, bei jeder Iteration bis zum Erreichen der Untergrenze herabgesetzt wird Hinweis: Die Untergrenze wird dennoch als Erstes referenziert.
<i>lower_bound</i>	Gibt die Untergrenze für den Bereich der Zählerwerte an
<i>upper_bound</i>	Gibt die Obergrenze für den Bereich der Zählerwerte an

Deklarieren Sie den Zähler nicht. Er ist implizit als Ganzzahl deklariert.

Hinweis: Die Anweisungsfolge wird bei jeder Inkrementierung des Zählers zwischen den beiden Grenzwerten ausgeführt. Die Ober- und Untergrenze des Schleifenbereichs können Literale, Variablen oder Ausdrücke sein. Sie müssen jedoch Ganzzahlen ergeben. Die Grenzwerte werden auf Ganzzahlen gerundet, das heißt, 11/3 und 8/5 sind gültige Ober- oder Untergrenzen. Die Unter- und Obergrenze zählen zum Schleifenbereich. Wenn die Untergrenze des Schleifenbereichs eine größere Ganzzahl als die Obergrenze ergibt, wird die Anweisungsfolge nicht ausgeführt.

Die folgende Anweisung wird beispielsweise nur einmal ausgeführt:

```
FOR i IN 3..3
LOOP
    statement1;
END LOOP;
```

FOR-Schleifen – Beispiel

```
DECLARE
    v_countryid    locations.country_id%TYPE := 'CA';
    v_loc_id        locations.location_id%TYPE;
    v_new_city      locations.city%TYPE := 'Montreal';
BEGIN
    SELECT MAX(location_id) INTO v_loc_id
        FROM locations
       WHERE country_id = v_countryid;
    FOR i IN 1..3 LOOP
        INSERT INTO locations(location_id, city, country_id)
        VALUES((v_loc_id + i), v_new_city, v_countryid );
    END LOOP;
END;
/
```

ORACLE

Copyright © 2014, Oracle und/oder verbundene Unternehmen. All rights reserved. Alle Rechte vorbehalten.

Sie haben bereits gelernt, wie Sie mit einer Basisschleife und einer WHILE-Schleife drei neue Standorte für den Ländercode CA und die Stadt Montreal einfügen. Das Beispiel auf dieser Folie zeigt, wie Sie dies mit einer FOR-Schleife erreichen.

Ergebnisse

Führen Sie zur Anzeige der Ausgabe das Codebeispiel zu Folie 27_sa in code_ex_06.sql aus.

FOR-Schleifen – Regeln

- Zähler nur innerhalb der Schleife referenzieren, da er außerhalb nicht definiert ist
- Zähler nicht als Ziel einer Zuweisung referenzieren
- Keine der Schleifengrenzen darf NULL sein.

ORACLE

Copyright © 2014, Oracle und/oder verbundene Unternehmen. All rights reserved. Alle Rechte vorbehalten.

Die Folie enthält Richtlinien für das Erstellen von FOR-Schleifen.

Hinweis: Als Unter- und Obergrenze einer LOOP-Anweisung müssen keine numerischen Literale verwendet werden. Sie können auch Ausdrücke verwenden, die in numerische Werte konvertiert werden.

Beispiel:

```
DECLARE
    v_lower  NUMBER := 1;
    v_upper  NUMBER := 100;
BEGIN
    FOR i IN v_lower..v_upper LOOP
        ...
    END LOOP;
END;
/
```

Einsatz von Schleifen – Empfehlungen

- Basisschleife verwenden, wenn die Anweisungen innerhalb der Schleife mindestens einmal ausgeführt werden müssen
- WHILE-Schleife verwenden, wenn die Bedingung am Anfang jeder Iteration ausgewertet werden muss
- FOR-Schleife verwenden, wenn die Anzahl der Iterationen bekannt ist



Copyright © 2014, Oracle und/oder verbundene Unternehmen. All rights reserved. Alle Rechte vorbehalten.

Mit einer Basisschleife können Sie die zugehörige Anweisung mindestens einmal ausführen, selbst dann, wenn die Bedingung beim Aufrufen der Schleife bereits erfüllt ist. Ohne eine EXIT-Anweisung wäre die Schleife unendlich.

Mit WHILE-Schleifen können Sie eine Anweisungsfolge wiederholen, bis die Kontrollbedingung nicht mehr TRUE ist. Die Bedingung wird am Anfang jeder Iteration ausgewertet. Die Schleife wird abgeschlossen, wenn die Bedingung FALSE ist. Ist die Bedingung zu Beginn einer Schleife FALSE, werden keine weiteren Iterationen ausgeführt.

FOR-Schleifen haben vor dem Schlüsselwort LOOP eine Kontrollanweisung, mit der die Anzahl der von PL/SQL ausgeführten Iterationen bestimmt wird. Verwenden Sie FOR-Schleifen, wenn die Anzahl der Iterationen vordefiniert ist.

Verschachtelte Schleifen und Labels

- Mehrfache Verschachtelung von Schleifen möglich
- Blöcke und Schleifen zur Unterscheidung mit Labels versehen
- Äußere Schleife mit EXIT-Anweisung beenden, die das Label referenziert

ORACLE

Copyright © 2014, Oracle und/oder verbundene Unternehmen. All rights reserved. Alle Rechte vorbehalten.

Sie können FOR-Schleifen, WHILE-Schleifen und Basisschleifen ineinander verschachteln. Die übergeordnete Schleife wird bei Abschluss einer verschachtelten Schleife nicht abgeschlossen, es sei denn, es wurde eine Exception ausgelöst. Sie können Schleifen jedoch mit einem Label kennzeichnen und die äußere Schleife mit einer EXIT-Anweisung beenden.

Für Labelnamen gelten dieselben Regeln wie für andere IDs. Setzen Sie Labels vor Anweisungen, entweder in derselben oder einer separaten Zeile. Leerzeichen sind beim Parsen von PL/SQL nur in Literalen von Bedeutung. Versehen Sie Basisschleifen mit einem Label, indem Sie das Label vor das Wort LOOP zwischen Begrenzungszeichen (<<label>>) setzen. In FOR- und WHILE-Schleifen müssen Sie das Label vor FOR oder WHILE setzen.

Wenn die Schleife mit Labels versehen ist, können Sie den Labelnamen zur übersichtlicheren Strukturierung nach den END LOOP-Anweisungen einfügen.

Verschachtelte Schleifen und Labels – Beispiel

```
...
BEGIN
  <<Outer_loop>>
  LOOP
    v_counter := v_counter+1;
    EXIT WHEN v_counter>10;
    <<Inner_loop>>
    LOOP
      ...
      EXIT Outer_loop WHEN total_done = 'YES';
      -- Leave both loops
      EXIT WHEN inner_done = 'YES';
      -- Leave inner loop only
      ...
    END LOOP Inner_loop;
    ...
  END LOOP Outer_loop;
END;
/
```

ORACLE

Copyright © 2014, Oracle und/oder verbundene Unternehmen. All rights reserved. Alle Rechte vorbehalten.

Das Beispiel auf der Folie enthält zwei Schleifen. Die äußere Schleife wird mit dem Label <<Outer_Loop>> und die innere Schleife mit dem Label <<Inner_Loop>> definiert. Die IDs werden vor dem Wort LOOP zwischen Labelbegrenzungszeichen (<<label>>) eingefügt. Die innere Schleife ist in der äußeren Schleife verschachtelt. Die Labelnamen werden zur übersichtlichen Strukturierung nach den END LOOP-Anweisungen eingefügt.

CONTINUE-Anweisungen in PL/SQL

- Definition
 - Fügen die Funktionalität zum Ausführen der nächsten Schleifeniteration hinzu
 - Ermöglichen Programmierern die Übergabe der Kontrolle an die nächste Iteration einer Schleife
 - Verwenden für EXIT-Anweisungen eine parallele Struktur und Semantik
- Vorteile
 - Erleichtern die Programmierung
 - Ermöglichen leichte Performanceverbesserungen im Vergleich zu früheren Programmierworkarounds zur Simulierung von CONTINUE-Anweisungen



ORACLE®

Copyright © 2014, Oracle und/oder verbundene Unternehmen. All rights reserved. Alle Rechte vorbehalten.

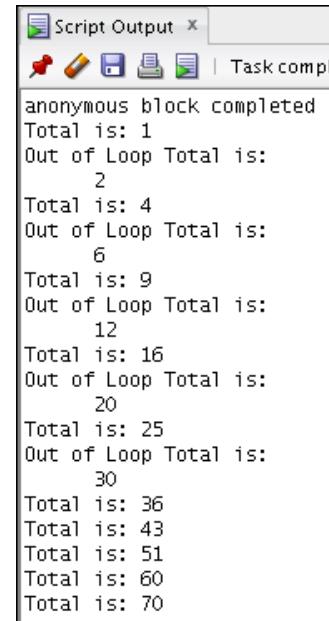
Mit CONTINUE-Anweisungen können Sie die Kontrolle innerhalb einer Schleife an eine neue Iteration zurückgeben oder die Schleife verlassen. Diese Funktion ist auch in vielen anderen Programmiersprachen zu finden. Ab Oracle Database 11g bietet auch PL/SQL diese Funktion. Vor Oracle Database 11g bestand die Möglichkeit, die Programmfunction CONTINUE mit booleschen Variablen und Bedingungsanweisungen zu simulieren. Dieser Workaround ist jedoch in einigen Fällen weniger effizient.

Mit der Anweisung CONTINUE können Sie Schleifeniterationen auf einfachere Weise steuern. Unter Umständen ist sie effizienter als frühere Codierungworkarounds.

CONTINUE-Anweisungen werden häufig verwendet, um Daten in einem Schleifenbody vor der Hauptverarbeitung zu filtern.

CONTINUE-Anweisungen in PL/SQL – 1. Beispiel

```
DECLARE
    v_total SIMPLE_INTEGER := 0;
BEGIN
    FOR i IN 1..10 LOOP
        1) v_total := v_total + i;
        dbms_output.put_line
            ('Total is: '|| v_total);
        CONTINUE WHEN i > 5;
        v_total := v_total + i;
        2) dbms_output.put_line
            ('Out of Loop Total is:
             '|| v_total);
    END LOOP;
END;
/
```



```
anonymous block completed
Total is: 1
Out of Loop Total is:
2
Total is: 4
Out of Loop Total is:
6
Total is: 9
Out of Loop Total is:
12
Total is: 16
Out of Loop Total is:
20
Total is: 25
Out of Loop Total is:
30
Total is: 36
Total is: 43
Total is: 51
Total is: 60
Total is: 70
```

ORACLE

Copyright © 2014, Oracle und/oder verbundene Unternehmen. All rights reserved. Alle Rechte vorbehalten.

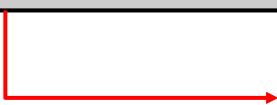
Im Beispiel verwenden zwei Zuweisungen die Variable v_total:

1. Die erste Zuweisung wird für jede der zehn Iterationen der Schleife ausgeführt.
2. Die zweite Ausführung der Zuweisung erfolgt für die ersten fünf Iterationen der Schleife. Die CONTINUE-Anweisung gibt die Kontrolle innerhalb einer Schleife an eine neue Iteration zurück, sodass die zweite Zuweisung von TOTAL für die letzten fünf Iterationen der Schleife nicht ausgeführt wird.

Das Endergebnis der Variablen TOTAL lautet 70.

CONTINUE-Anweisungen in PL/SQL – 2. Beispiel

```
DECLARE
  v_total NUMBER := 0;
BEGIN
  <<BeforeTopLoop>>
  FOR i IN 1..10 LOOP
    v_total := v_total + 1;
    dbms_output.put_line
      ('Total is: ' || v_total);
    FOR j IN 1..10 LOOP
      CONTINUE BeforeTopLoop WHEN i + j > 5;
      v_total := v_total + 1;
    END LOOP;
  END LOOP BeforeTopLoop;
END;
```



```
Script Output X
Task completed
anonymous block completed
Total is: 1
Total is: 6
Total is: 10
Total is: 13
Total is: 15
Total is: 16
Total is: 17
Total is: 18
Total is: 19
Total is: 20
```

ORACLE

Copyright © 2014, Oracle und/oder verbundene Unternehmen. All rights reserved. Alle Rechte vorbehalten.

Mit CONTINUE-Anweisungen können Sie zur nächsten Iteration einer äußeren Schleife springen. Versehen Sie hierfür die äußere Schleife mit einem Label, um das Ziel der CONTINUE-Anweisung zu identifizieren.

Die CONTINUE-Anweisung in der innersten Schleife schließt diese Schleife ab, wenn die WHEN-Bedingung mit TRUE ausgewertet wird (ebenso wie das Schlüsselwort EXIT). Nachdem die innerste Schleife mit der CONTINUE-Anweisung abgeschlossen wurde, wird in diesem Beispiel die Kontrolle an die nächste Iteration der äußersten Schleife mit dem Label BeforeTopLoop übergeben.

Wenn dieses Schleifenpaar beendet ist, weist die Variable TOTAL den Wert 20 auf.

Sie können CONTINUE-Anweisungen auch in einem inneren Codeblock verwenden, der keine Schleife enthält. Allerdings setzt dies voraus, dass der Block in einer entsprechenden äußeren Schleife verschachtelt ist.

Einschränkungen

- CONTINUE-Anweisungen dürfen nicht außerhalb einer Schleife auftreten. Dies generiert einen Compilerfehler.
- Mit CONTINUE-Anweisungen dürfen Sie nicht die Grenzen von Prozeduren, Funktionen oder Methoden überschreiten. Dies generiert einen Compilerfehler.

Quiz

Es gibt drei Schleifentypen: Basisschleifen, FOR-Schleifen und WHILE-Schleifen

- a. Richtig
- b. Falsch



Copyright © 2014, Oracle und/oder verbundene Unternehmen. All rights reserved. Alle Rechte vorbehalten.

Richtige Antwort: a

Schleifentypen

PL/SQL bietet die folgenden Schleifentypen:

- Basisschleifen für wiederholte Aktionen ohne allgemeine Bedingungen
- FOR-Schleifen für iterative Aktionen auf Basis eines Zählers
- WHILE-Schleifen für iterative Aktionen auf Basis einer Bedingung

Zusammenfassung

In dieser Lektion haben Sie gelernt, wie Sie den logischen Anweisungsablauf mit den folgenden Kontrollstrukturen ändern:

- Bedingungsanweisungen (`IF`-Anweisungen)
- CASE-Ausdrücke und CASE-Anweisungen
- Schleifen:
 - Basisschleifen
 - FOR-Schleifen
 - WHILE-Schleifen
- EXIT-Anweisungen
- CONTINUE-Anweisungen



Copyright © 2014, Oracle und/oder verbundene Unternehmen. All rights reserved. Alle Rechte vorbehalten.

Eine Sprache ist nur dann eine Programmiersprache, wenn sie Kontrollstrukturen zum Implementieren der Geschäftslogik bietet. Mit diesen Kontrollstrukturen können Sie auch den Programmfluss steuern. Die Programmiersprache PL/SQL integriert Programmierkonstrukte in SQL.

Bedingte Kontrollkonstrukte prüfen die Gültigkeit von Bedingungen und führen entsprechende Aktionen aus. Mit dem Konstrukt `IF` können Sie Anweisungen bedingt ausführen.

Kontrollierte Iterationskonstrukte führen Anweisungsfolgen wiederholt aus, bis eine angegebene Bedingung `TRUE` ist. Für iterative Vorgänge stehen verschiedene Schleifenkonstrukte zur Verfügung.

Übungen zu Lektion 6 – Übersicht

Diese Übung behandelt folgende Themen:

- Bedingte Aktionen mit `IF`-Anweisungen ausführen
- Iterative Schritte mit `LOOP`-Strukturen ausführen



Copyright © 2014, Oracle und/oder verbundene Unternehmen. All rights reserved. Alle Rechte vorbehalten.

In diesen Übungen erstellen Sie PL/SQL-Blöcke, die Schleifen und bedingte Kontrollstrukturen beinhalten. Sie erstellen dabei anhand des Gelernten verschiedene `IF`-Anweisungen und `LOOP`-Konstrukte.



Mit zusammengesetzten Datentypen arbeiten

ORACLE

Copyright © 2014, Oracle und/oder verbundene Unternehmen. All rights reserved. Alle Rechte vorbehalten.

Ziele

Nach Ablauf dieser Lektion haben Sie folgende Ziele erreicht:

- PL/SQL-Collections und -Records beschreiben
- Benutzerdefinierte PL/SQL-Records erstellen
- PL/SQL-Records mit dem Attribut %ROWTYPE definieren
- Assoziative Arrays erstellen
 - INDEX BY-Tabellen
 - INDEX BY-Record-Tabellen



Copyright © 2014, Oracle und/oder verbundene Unternehmen. All rights reserved. Alle Rechte vorbehalten.

Zusammengesetzte Datentypen wurden bereits vorgestellt. In dieser Lektion erfahren Sie mehr über zusammengesetzte Datentypen und ihre Verwendung.

Agenda

- Zusammengesetzte Datentypen untersuchen
- PL/SQL-Records verwenden
 - Daten mit PL/SQL-Records bearbeiten
 - Attribut %ROWTYPE – Vorteile
- PL/SQL-Collections verwenden
 - Assoziative Arrays untersuchen
 - Nested Tables – Einführung
 - VARRAYS – Einführung

ORACLE

Copyright © 2014, Oracle und/oder verbundene Unternehmen. All rights reserved. Alle Rechte vorbehalten.

Zusammengesetzte Datentypen

- Können (im Gegensatz zu skalaren Typen) mehrere Werte aufnehmen
- Zwei Arten:
 - PL/SQL-Records
 - PL/SQL-Collections
 - Assoziative Arrays (INDEX BY-Tabellen)
 - Nested Tables
 - VARRAYS

ORACLE

Copyright © 2014, Oracle und/oder verbundene Unternehmen. All rights reserved. Alle Rechte vorbehalten.

Sie haben bereits erfahren, dass Variablen mit skalarem Datentyp nur einen einzelnen Wert speichern, während Variablen mit zusammengesetztem Datentyp mehrere Werte (skalar oder zusammengesetzt) speichern können. Es gibt zwei Arten von zusammengesetzten Datentypen:

- **PL/SQL-Records:** Mit Records können Sie zusammengehörige Daten unterschiedlichen Typs als logische Einheit behandeln. PL/SQL-Records können Variablen unterschiedlicher Datentypen enthalten. Beispiel: Sie möchten einen Record definieren, der Mitarbeiterdetails speichern soll. Zu diesem Zweck speichern Sie beispielsweise eine Personalnummer als NUMBER und einen Vor- und Nachnamen als VARCHAR2. Mit dem Record zum Speichern von Mitarbeiterdetails erstellen Sie eine logische kollektive Einheit, die den Datenzugriff und die Datenbearbeitung vereinfacht.
- **PL/SQL-Collections:** In Collections zusammengefasste Daten werden als eine Einheit behandelt. Es gibt drei Arten von Collections:
 - Assoziative Arrays
 - Nested Tables
 - VARRAYS

Zweck zusammengesetzter Datentypen

Alle zusammengehörigen Daten liegen in einer Einheit vor. Sie können einfach auf die Daten zugreifen und sie bearbeiten. Zusammengesetzte Daten lassen sich leichter verwalten, verknüpfen und übertragen. Vergleichen Sie dies mit einer Tasche, in der Sie alle Komponenten Ihres Laptops befördern. Das ist einfacher, als jede einzelne Komponente in einer separaten Tasche aufzubewahren.

PL/SQL-Records oder -Collections?

- In PL/SQL-Records jeweils ein Vorkommen von Werten unterschiedlichen Datentyps speichern
- Mit PL/SQL-Collections Werte speichern, die demselben Datentyp angehören

PL/SQL-Record:

TRUE	23-DEC-98	ATLANTA	
------	-----------	---------	---

PL/SQL-Collection:

1	SMITH
2	JONES
3	BENNETT
4	KRAMER

PLS_INTEGER

VARCHAR2



ORACLE

Copyright © 2014, Oracle und/oder verbundene Unternehmen. All rights reserved. Alle Rechte vorbehalten.

Nach welchen Kriterien entscheiden Sie sich zwischen den beiden zusammengesetzten Datentypen, PL/SQL-Records und PL/SQL-Collections?

- Verwenden Sie PL/SQL-Records, wenn Sie Werte unterschiedlichen Datentyps speichern möchten, die logisch zusammengehören. Beispiel: Sie können einen Record zur Aufnahme von Mitarbeiterdetails erstellen und angeben, dass alle gespeicherten Werte zusammengehörig sind, da sie Informationen zu einem bestimmten Mitarbeiter bereitstellen.
- Verwenden Sie PL/SQL-Collections, um Werte zu speichern, die demselben Datentyp angehören. Dies kann auch ein zusammengesetzter Datentyp (etwa ein Record) sein. Sie können beispielsweise eine Collection definieren, in der die Vornamen aller Mitarbeiter erfasst sind. Sie können in der Collection n Namen speichern, wobei der 1. Name jedoch keinerlei Beziehung zum 2. Namen hat. Die einzige Beziehung besteht darin, dass beides Mitarbeiternamen sind. Diese Collections sind mit Arrays in Programmiersprachen wie C, C++ und Java vergleichbar.

Agenda

- Zusammengesetzte Datentypen untersuchen
- PL/SQL-Records verwenden
 - Daten mit PL/SQL-Records bearbeiten
 - Attribut %ROWTYPE – Vorteile
- PL/SQL-Collections verwenden
 - Assoziative Arrays untersuchen
 - Nested Tables – Einführung
 - VARRAYS – Einführung

ORACLE®

Copyright © 2014, Oracle und/oder verbundene Unternehmen. All rights reserved. Alle Rechte vorbehalten.

PL/SQL-Records

- Müssen eine oder mehrere Komponenten (*Felder*) vom Tabellendatentyp skalar, RECORD oder INDEX BY enthalten
- Ähneln in der Struktur den meisten Sprachen der dritten Generation (einschließlich C und C++)
- Sind benutzerdefiniert und können ein Ausschnitt einer Tabellenzeile sein
- Behandeln eine Collection von Feldern als logische Einheit
- Eignen sich zum Abrufen einer Datenzeile aus einer Tabelle für die Verarbeitung

ORACLE

Copyright © 2014, Oracle und/oder verbundene Unternehmen. All rights reserved. Alle Rechte vorbehalten.

Ein Record ist eine Gruppe zusammengehöriger Datenelemente, die mit eigenem Namen und Datentyp in Feldern gespeichert sind.

- Jeder definierte Record kann beliebig viele Felder enthalten.
- Records können Ausgangswerte zugeordnet werden. Sie können als NOT NULL definiert werden.
- Felder ohne Ausgangswerte werden mit NULL initialisiert.
- Zum Initialisieren der Felder können Sie das Schlüsselwort DEFAULT und den Operator := verwenden.
- Sie können RECORD-Typen definieren und im deklarativen Teil von Blöcken, Unterprogrammen oder Packages benutzerdefinierte Records deklarieren.
- Außerdem können Sie verschachtelte Records deklarieren und referenzieren. Ein Record kann die Komponente eines anderen Records sein.

PL/SQL-Records erstellen

Syntax:

1

```
TYPE type_name IS RECORD  
  (field_declaration[, field_declaration]...);
```

2

```
identifier      type_name;
```

field_declaration:

```
field_name {field_type | variable%TYPE  
           | table.column%TYPE | table%ROWTYPE}  
           [ [NOT NULL] {:= | DEFAULT} expr]
```

ORACLE

Copyright © 2014, Oracle und/oder verbundene Unternehmen. All rights reserved. Alle Rechte vorbehalten.

PL/SQL-Records sind benutzerdefinierte zusammengesetzte Datentypen. Sie verwenden diese wie folgt:

1. Definieren Sie den Record im deklarativen Bereich eines PL/SQL-Blockes. Die Syntax ist auf der Folie dargestellt.
2. Deklarieren und initialisieren (optional) Sie die internen Komponenten des Record-Typs.

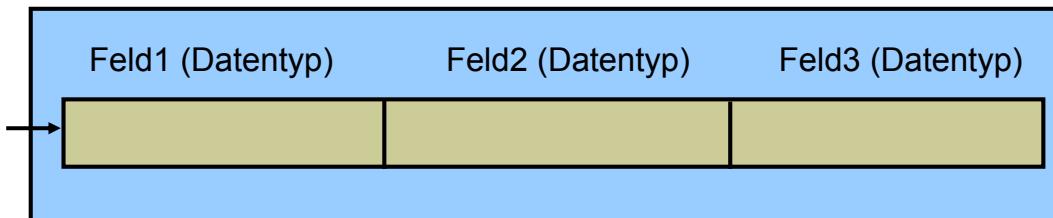
Für die Syntax gilt:

type_name	Name des RECORD-Typs (ID zur Deklarierung von Records)
field_name	Name eines Feldes innerhalb des Records
field_type	Datentyp des Feldes (Stellt alle PL/SQL-Datentypen außer REF CURSOR dar. Verwendung der Attribute %TYPE und %ROWTYPE möglich)
expr	Ausgangswert

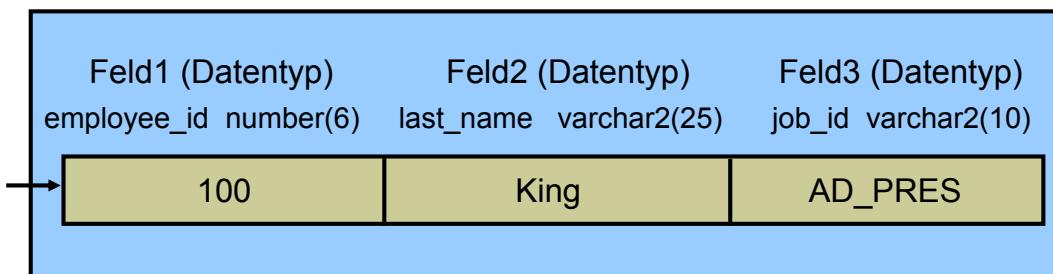
Das NOT NULL Constraint verhindert, dass den angegebenen Feldern NULL-Werte zugewiesen werden. Vergewissern Sie sich, dass NOT NULL-Felder initialisiert werden.

Struktur von PL/SQL-Records

Felddeklarationen:



Beispiel:



ORACLE

Copyright © 2014, Oracle und/oder verbundene Unternehmen. All rights reserved. Alle Rechte vorbehalten.

Auf Felder in einem Record greifen Sie mit dem Namen des Records zu. Um ein einzelnes Feld zu referenzieren oder zu initialisieren, verwenden Sie die Punktschreibweise:

```
record_name.field_name
```

Das Feld `job_id` im Record `emp_record` wird wie folgt referenziert:

```
emp_record.job_id
```

Anschließend können Sie dem Feld einen Wert zuweisen:

```
emp_record.job_id := 'ST_CLERK';
```

In einem Block oder Unterprogramm enthaltene benutzerdefinierte Records werden instanziert, wenn Sie den Block oder das Unterprogramm starten. Sie werden gelöscht, wenn Sie den Block oder das Unterprogramm beenden.

Attribut %ROWTYPE

- Variablen entsprechend einer Collection von Spalten in Datenbanktabellen oder Datenbank-Views deklarieren
- Datenbanktabellen oder Datenbank-Views als Präfix für %ROWTYPE verwenden
- Felder im Record übernehmen Namen und Datentyp von den Spalten der Tabelle oder View.

Syntax:

```
DECLARE  
    identifier reference%ROWTYPE;
```

ORACLE

Copyright © 2014, Oracle und/oder verbundene Unternehmen. All rights reserved. Alle Rechte vorbehalten.

Sie haben bereits gelernt, dass Sie mit %TYPE Variablen für Spalten deklarieren können. Die Variable hat denselben Datentyp und dieselbe Größe wie die Tabellenspalte. %TYPE bietet den Vorteil, dass Sie die Variable nicht ändern müssen, wenn die Spalte geändert wird. Und wenn die Variable eine Zahl ist, die in Berechnungen verwendet wird, müssen Sie keine Gesamtstelligenzahl festlegen.

Mit dem Attribut %ROWTYPE wird ein Record deklariert, in dem Sie eine ganze Zeile einer Tabelle oder View speichern können. Die Felder im Record übernehmen Namen und Datentyp aus den Spalten der Tabelle oder View. Der Record kann auch eine ganze Datenzeile speichern, die aus einem Cursor oder einer Cursorvariablen gelesen wurde.

Die Folie zeigt die Syntax, mit der Sie einen Record deklarieren. Für die Syntax gilt:

identifier Für den ganzen Record gewählter Name

reference Name der Tabelle, der View, des Cursors oder der Cursorvariablen, auf der/ dem der Record basiert. (Damit diese Referenz gültig ist, muss die Tabelle oder View vorhanden sein.)

Im folgenden Beispiel wird ein Record mit der Datentypspezifikation %ROWTYPE deklariert:

```
DECLARE  
    emp_record employees%ROWTYPE;  
    ...
```

Der Record `emp_record` besteht aus den folgenden Feldern, die jeweils eine Spalte der Tabelle `employees` darstellen.

Hinweis: Dies ist kein Code, sondern lediglich die Struktur der zusammengesetzten Variablen.

```
(employee_id      NUMBER(6),
 first_name       VARCHAR2(20),
 last_name        VARCHAR2(20),
 email            VARCHAR2(20),
 phone_number     VARCHAR2(20),
 hire_date        DATE,
 job_id           VARCHAR2(10),
 salary            NUMBER(8,2),
 commission_pct   NUMBER(2,2),
 manager_id       NUMBER(6),
 department_id    NUMBER(4))
```

Um ein einzelnes Feld zu referenzieren, verwenden Sie die Punktschreibweise:

```
record_name.field_name
```

Das Feld `commission_pct` im Record `emp_record` wird wie folgt referenziert:

```
emp_record.commission_pct
```

Anschließend können Sie dem Feld einen Wert zuweisen:

```
emp_record.commission_pct := .35;
```

Records Werte zuweisen

Sie können Records mit der Anweisung `SELECT` oder `FETCH` eine Liste gängiger Werte zuweisen. Achten Sie darauf, dass die Spaltennamen in der gleichen Reihenfolge wie die Felder im Record angezeigt werden. Sie können Records auch anderen Records zuweisen, wenn die Datentypen in beiden Records gleich sind. Records vom Typ `employees%ROWTYPE` und benutzerdefinierte Records mit analogen Feldern der Tabelle `employees` besitzen denselben Datentyp. Sie können daher dem `%ROWTYPE`-Record einen benutzerdefinierten Record zuweisen, wenn die enthaltenen Felder denen eines `%ROWTYPE`-Records ähnlich sind.

PL/SQL-Records erstellen – Beispiel

```
DECLARE
    TYPE t_rec IS RECORD
        (v_sal number(8),
         v_minsal number(8) default 1000,
         v_hire_date employees.hire_date%type,
         v_rec1 employees%rowtype);
    v_myrec t_rec;
BEGIN
    v_myrec.v_sal := v_myrec.v_minsal + 500;
    v_myrec.v_hire_date := sysdate;
    SELECT * INTO v_myrec.v_rec1
        FROM employees WHERE employee_id = 100;
    DBMS_OUTPUT.PUT_LINE(v_myrec.v_rec1.last_name || ' ' ||
        to_char(v_myrec.v_hire_date) || ' ' || to_char(v_myrec.v_sal));
END;
```

```
anonymous block completed
King 16-OCT-12 1500
```

ORACLE

Copyright © 2014, Oracle und/oder verbundene Unternehmen. All rights reserved. Alle Rechte vorbehalten.

Die Felddeklarationen zum Definieren von Records ähneln Variablen Deklarationen. Jedes Feld besitzt einen eindeutigen Namen und einen spezifischen Datentyp. Anders als bei skalaren Variablen gibt es für PL/SQL-Records keine vordefinierten Datentypen. Daher müssen Sie zuerst den Record-Typ erstellen und danach mithilfe dieses Typs eine ID deklarieren.

Im Beispiel auf der Folie wird die Erstellung eines PL/SQL-Records in den erforderlichen zwei Schritten ausgeführt:

1. Ein Record-Typ (`t_rec`) wird definiert.
2. Ein Record (`v_myrec`) vom Typ `t_rec` wird deklariert.

Hinweise

- Der Record enthält vier Felder: `v_sal`, `v_minsal`, `v_hire_date` und `v_rec1`.
- `v_rec1` wird mit dem Attribut `%ROWTYPE` definiert, das `%TYPE` ähnelt. Mit `%TYPE` übernimmt ein Feld den Datentyp einer angegebenen Spalte. Mit `%ROWTYPE` übernimmt ein Feld die Spaltennamen und Datentypen aller Spalten in der referenzierten Tabelle.
- PL/SQL-Record-Felder werden mit der Notation `<record>. <field>` referenziert. Felder, die mit dem Attribut `%ROWTYPE` definiert wurden, werden mit der Notation `<record>. <field>. <column>` referenziert.
- Um die Zuweisung von NULL-Werten zu verhindern, können Sie der Felddeklaration das NOT NULL Constraint hinzufügen. Denken Sie daran, dass Felder, die als NOT NULL deklariert sind, initialisiert werden müssen.

Attribut %ROWTYPE – Vorteile

- Anzahl und Datentyp der zugrunde liegenden Datenbankspalten müssen nicht bekannt sein und können sich zur Laufzeit ändern.
- Das Attribut %ROWTYPE ist nützlich zum Abrufen von Zeilen mit:
 - SELECT *-Anweisungen
 - INSERT- und UPDATE-Anweisungen auf Zeilenebene



Copyright © 2014, Oracle und/oder verbundene Unternehmen. All rights reserved. Alle Rechte vorbehalten.

Die Vorteile des Attributs %ROWTYPE sind auf der Folie aufgelistet. Verwenden Sie das Attribut %ROWTYPE, wenn Sie die Struktur der zugrunde liegenden Datenbanktabelle nicht kennen.

Der Hauptvorteil von %ROWTYPE liegt in der einfacheren Verwaltung. Mit %ROWTYPE stellen Sie sicher, dass sich die Datentypen der mit dem Attribut deklarierten Variablen dynamisch an Änderungen der zugrunde liegenden Tabelle anpassen. Falls die Spalten einer Tabelle durch eine DDL-Anweisung geändert werden, wird die PL/SQL-Programmeinheit invalidiert. Nach der Rekompilierung des Programms entspricht dieses automatisch dem neuen Tabellenformat.

Das Attribut %ROWTYPE ist speziell dann nützlich, wenn eine ganze Zeile aus einer Tabelle abgerufen werden soll. Ohne dieses Attribut müssten Sie für jede der mit der SELECT-Anweisung abgerufenen Spalten eine Variable deklarieren.

Attribut %ROWTYPE – Weiteres Beispiel

```
DECLARE
    v_employee_number number:= 124;
    v_emp_rec    employees%ROWTYPE;
BEGIN
    SELECT * INTO v_emp_rec FROM employees
    WHERE employee_id = v_employee_number;
    INSERT INTO retired_emps(empno, ename, job, mgr,
                           hiredate, leavedate, sal, comm, deptno)
    VALUES (v_emp_rec.employee_id, v_emp_rec.last_name,
            v_emp_rec.job_id, v_emp_rec.manager_id,
            v_emp_rec.hire_date, SYSDATE,
            v_emp_rec.salary, v_emp_rec.commission_pct,
            v_emp_rec.department_id);
END;
/
SELECT * FROM retired_emps;
```

Query Result | All Rows Fetched: 1 in 0.005 seconds

EMPNO	ENAME	JOB	MGR	HIREDATE	LEAVEDATE	SAL	COMM	DEPTNO
1	124 Mourgos	ST_MAN	100	16-NOV-07	16-OCT-12	5800	(null)	50

ORACLE

Copyright © 2014, Oracle und/oder verbundene Unternehmen. All rights reserved. Alle Rechte vorbehalten.

Die Folie zeigt ein weiteres Beispiel für das Attribut %ROWTYPE. Ein Mitarbeiter geht in den Ruhestand. Informationen zu diesem Mitarbeiter werden in eine Tabelle aufgenommen, die Informationen zu pensionierten Mitarbeitern erfasst. Der Benutzer gibt die Personalnummer an. Der Record des vom Benutzer angegebenen Mitarbeiters wird aus der Tabelle `employees` abgerufen und in der Variablen `emp_rec` gespeichert, die mit dem Attribut %ROWTYPE deklariert wird.

Die CREATE-Anweisung, mit der Sie die Tabelle `retired_emps` erstellen, lautet:

```
CREATE TABLE retired_emps
  (EMPNO      NUMBER(4), ENAME      VARCHAR2(10),
   JOB        VARCHAR2(9), MGR        NUMBER(4),
   HIREDATE    DATE, LEAVEDATE   DATE,
   SAL         NUMBER(7,2), COMM        NUMBER(7,2),
   DEPTNO     NUMBER(2))
```

Hinweise

- Der Record, der in die Tabelle `retired_emps` eingefügt wird, ist auf der Folie dargestellt.
- Um die auf der Folie dargestellte Ausgabe anzuzeigen, positionieren Sie den Cursor in SQL Developer auf die SELECT-Anweisung am Ende des Codebeispiels, und drücken Sie F9.
- Das vollständige Codebeispiel finden Sie unter der Folie 14_s-n in `code_ex_07.sql`.

Records mit %ROWTYPE einfügen

```
...
DECLARE
    v_employee_number number:= 124;
    v_emp_rec retired_emps%ROWTYPE;
BEGIN
    SELECT employee_id, last_name, job_id, manager_id,
    hire_date, hire_date, salary, commission_pct,
    department_id INTO v_emp_rec FROM employees
    WHERE employee_id = v_employee_number;
    INSERT INTO retired_emps VALUES v_emp_rec;
END;
/
SELECT * FROM retired_emps;
```

The screenshot shows the Oracle SQL Developer interface. A query window titled 'Query Result' displays the output of the executed PL/SQL block. The output shows one row inserted into the 'retired_emps' table. The columns are labeled: EMPNO, ENAME, JOB, MGR, HIREDATE, LEAVEDATE, SAL, COMM, DEPTNO. The values for the inserted row are: 1, 124, Mourgos, ST_MAN, 100, 16-NOV-07, 16-NOV-07, 5800, (null), 50.

EMPNO	ENAME	JOB	MGR	HIREDATE	LEAVEDATE	SAL	COMM	DEPTNO
1	124	Mourgos	ST_MAN	100	16-NOV-07	16-NOV-07	5800	(null)

ORACLE

Copyright © 2014, Oracle und/oder verbundene Unternehmen. All rights reserved. Alle Rechte vorbehalten.

Vergleichen Sie die `INSERT`-Anweisungen auf dieser und der vorherigen Folie miteinander. Der Record `emp_rec` ist vom Typ `retired_emps`. Die Anzahl der Felder im Record muss mit der Anzahl der Feldnamen in der `INTO`-Klausel übereinstimmen. Sie können diesen Record verwenden, um Werte in eine Tabelle einzufügen. Auf diese Weise wird der Code besser lesbar. Sehen Sie sich die `SELECT`-Anweisung auf der Folie an. Sie wählen zweimal `hire_date` und fügen den Wert von `hire_date` in der Tabelle `retired_emps` in das Feld `leavedate` ein. Das Ruhestandsdatum kann nicht das Einstellungsdatum sein. Der eingefügte Record wird auf der Folie angezeigt. (Auf der nächsten Folie sehen Sie, wie Sie die Zeile aktualisieren.)

Hinweis: Um die auf der Folie dargestellte Ausgabe anzuzeigen, positionieren Sie den Cursor in SQL Developer auf die `SELECT`-Anweisung am Ende des Codebeispiels, und drücken Sie F9.

Zeilen in einer Tabelle mit einem Record aktualisieren

```
DECLARE
    v_employee_number number:= 124;
    v_emp_rec    retired_emps%ROWTYPE;
BEGIN
    SELECT * INTO v_emp_rec FROM retired_emps WHERE
        empno = v_employee_number;
    v_emp_rec.leavedate:= CURRENT_DATE;
    UPDATE retired_emps SET ROW = v_emp_rec WHERE
        empno=v_employee_number;
END;
/
SELECT * FROM retired_emps;
```

EMPNO	ENAME	JOB	MGR	HIREDATE	LEAVEDATE	SAL	COMM	DEPTNO	
1	124	Mourgos	ST_MAN	100	16-NOV-07	05-NOV-12	5800	(null)	50

ORACLE

Copyright © 2014, Oracle und/oder verbundene Unternehmen. All rights reserved. Alle Rechte vorbehalten.

Sie haben erfahren, wie Sie mit einem Record eine Zeile einfügen. Auf dieser Folie wird gezeigt, wie Sie eine Zeile aktualisieren.

- Das Schlüsselwort `ROW` steht für die ganze Zeile.
- Der Code auf der Folie aktualisiert das `leavedate` des Mitarbeiters.
- Der Record wird wie auf der Folie gezeigt aktualisiert.

Hinweis: Um die auf der Folie dargestellte Ausgabe anzuzeigen, positionieren Sie den Cursor in SQL Developer auf die `SELECT`-Anweisung am Ende des Codebeispiels, und drücken Sie F9.

Agenda

- Zusammengesetzte Datentypen untersuchen
- PL/SQL-Records verwenden
 - Daten mit PL/SQL-Records bearbeiten
 - Attribut %ROWTYPE – Vorteile
- PL/SQL-Collections verwenden
 - Assoziative Arrays untersuchen
 - Nested Tables – Einführung
 - VARRAYS – Einführung

ORACLE

Copyright © 2014, Oracle und/oder verbundene Unternehmen. All rights reserved. Alle Rechte vorbehalten.

Wie bereits erläutert, werden PL/SQL-Collections verwendet, um Werte vom selben Datentyp zu speichern. Dabei kann es sich auch um einen zusammengesetzten Datentyp (etwa einen Record) handeln.

Die Daten in Collections werden als eine Einheit behandelt. Es gibt drei Arten von Collections:

- Assoziative Arrays
- Nested Tables
- VARRAYS

Hinweis: Der Schwerpunkt unter diesen drei Collection-Typen liegt in dieser Lektion auf assoziativen Arrays. Nested Tables und VARRAYS werden hier nur zu Vergleichszwecken vorgestellt. Ausführliche Informationen über diese beiden Collections erhalten Sie im Kurs *Oracle Database: Advanced PL/SQL*.

Assoziative Arrays (**INDEX BY**-Tabellen)

Ein assoziatives Array ist eine PL/SQL-Collection mit zwei Spalten:

- Primärschlüssel vom Datentyp Ganzzahl oder Zeichenfolge
- Spalte vom Datentyp skalar oder Record

Schlüssel	Werte
1	JONES
2	HARDEY
3	MADURO
4	KRAMER

ORACLE

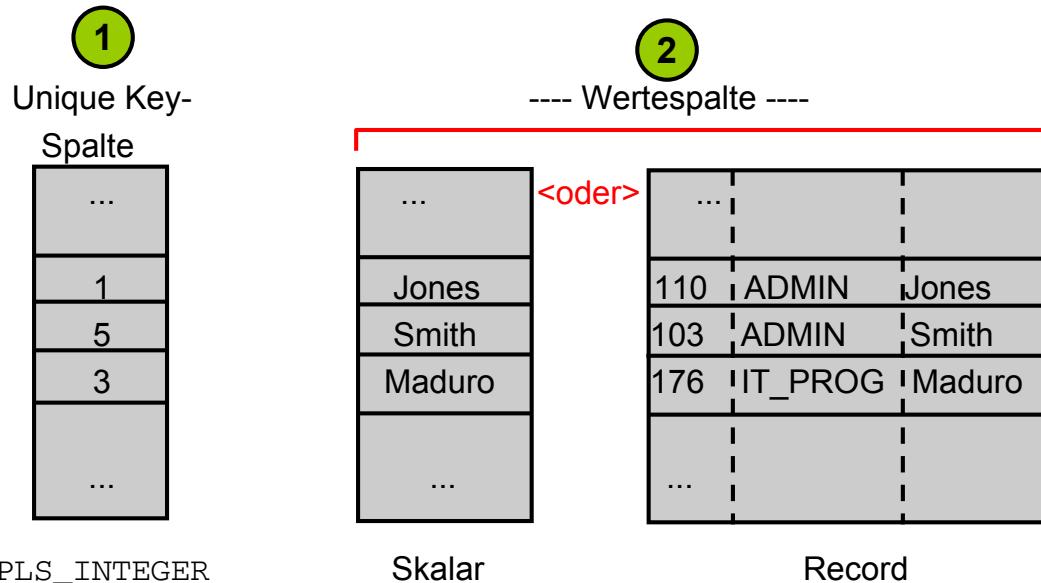
Copyright © 2014, Oracle und/oder verbundene Unternehmen. All rights reserved. Alle Rechte vorbehalten.

Ein assoziatives Array ist ein Typ einer PL/SQL-Collection. Es handelt sich um einen zusammengesetzten Datentyp, der benutzerdefiniert ist. Assoziative Arrays sind Gruppen von Schlüssel/Werte-Paaren. Sie können Daten mit einem Primärschlüsselwert als Index speichern, wobei die Schlüsselwerte nicht zwingend sequenziell sind. Assoziative Arrays werden auch als *INDEX BY*-Tabellen bezeichnet.

Sie besitzen nur zwei Spalten, die beide nicht benannt werden können:

- Die erste Spalte für Ganzzahlen- oder Zeichenfolgenwerte dient als Primärschlüssel.
- In der zweiten Spalte vom skalaren Datentyp oder Record-Datentyp werden Werte gespeichert.

Assoziative Arrays – Struktur



ORACLE

Copyright © 2014, Oracle und/oder verbundene Unternehmen. All rights reserved. Alle Rechte vorbehalten.

Wie bereits erwähnt, verfügen assoziative Arrays über zwei Spalten. In der zweiten Spalte werden pro Zeile entweder ein Wert oder mehrere Werte gespeichert.

Unique Key-Spalte: Die Spalte kann folgende Datentypen annehmen:

- Numerisch, entweder BINARY_INTEGER oder PLS_INTEGER. Diese beiden numerischen Datentypen belegen weniger Speicher als NUMBER. Außerdem können arithmetische Vorgänge schneller ausgeführt werden als mit NUMBER-Arithmetik.
- VARCHAR2 oder ein Subtyp

Wertespalte: Die Wertespalte kann entweder einen skalaren Datentyp oder einen Record-Datentyp aufweisen. Eine Spalte mit skalarem Datentyp kann nur einen Wert pro Zeile aufnehmen, während eine Spalte des Record-Datentyps mehrere Werte pro Zeile speichern kann.

Weitere Eigenschaften

- Ein assoziatives Array wird bei der Deklaration nicht gefüllt. Es enthält weder Schlüssel noch Werte, und Sie können ein assoziatives Array bei dessen Deklaration nicht initialisieren.
- Um das assoziative Array zu füllen, müssen Sie eine explizite ausführbare Anweisung angeben.
- Ein assoziatives Array ist wie eine Datenbanktabelle hinsichtlich der Größe uneingeschränkt. Die Anzahl der Zeilen kann sich also dynamisch erhöhen, sodass das assoziative Array durch Hinzufügen neuer Zeilen wächst. Beachten Sie, dass die Schlüssel nicht sequenziell sein müssen und sowohl positiv als auch negativ sein können.

Assoziative Arrays erstellen – Schrittfolge

Syntax:

```
1   TYPE type_name IS TABLE OF
    { column_type [NOT NULL] | variable%TYPE [NOT NULL]
    | table.column%TYPE [NOT NULL]
    | table%ROWTYPE }
    INDEX BY { PLS_INTEGER | BINARY_INTEGER
    | VARCHAR2(<size>) } ;
2   identifier type_name;
```

Beispiel:

```
...
→ TYPE [ename_table_type] IS TABLE OF
    employees.last_name%TYPE
    INDEX BY PLS_INTEGER;
...
→ ename_table [ename_table_type];
```

ORACLE

Copyright © 2014, Oracle und/oder verbundene Unternehmen. All rights reserved. Alle Rechte vorbehalten.

Assoziative Arrays werden in zwei Schritten erstellt:

1. Deklarieren Sie mit der Option INDEX BY einen TABLE-Datentyp.
2. Deklarieren Sie eine Variable dieses Datentyps.

Syntax

type_name Name des TABLE-Typs (Dieser Name wird in der nachfolgenden Deklaration der Array-ID verwendet.)
column_type Beliebiger skalarer oder zusammengesetzter Datentyp wie VARCHAR2, DATE, NUMBER oder %TYPE (Mit dem Attribut %TYPE können Sie den Spaltendatentyp angeben.)
identifier Name der ID, die ein ganzes assoziatives Array darstellt

Hinweis: Das NOT NULL Constraint verhindert, dass dem assoziativen Array NULL-Werte zugewiesen werden.

Beispiel:

Im Beispiel wird ein assoziatives Array mit dem Variablenamen `ename_table` deklariert, das die Nachnamen von Mitarbeitern speichert.

Assoziative Arrays erstellen und abrufen

```
...
DECLARE
    TYPE ename_table_type IS TABLE OF
        employees.last_name%TYPE
        INDEX BY PLS_INTEGER;
    TYPE hiredate_table_type IS TABLE OF DATE
        INDEX BY PLS_INTEGER;
    ename_table      ename_table_type;
    hiredate_table   hiredate_table_type;
BEGIN
    ename_table(1)      := 'CAMERON';
    hiredate_table(8)   := SYSDATE + 7;
    IF ename_table.EXISTS(1) THEN
        INSERT INTO ...
    ...
END;
/
...
```

Script Output	
	Task completed in 0.047 seconds
anonymous block completed	
ENAME	HIREDT
CAMERON	23-OCT-12

ORACLE

Copyright © 2014, Oracle und/oder verbundene Unternehmen. All rights reserved. Alle Rechte vorbehalten.

Im Beispiel auf der Folie werden zwei assoziative Arrays mit den IDs `ename_table` und `hiredate_table` erstellt.

Über den Schlüssel der einzelnen assoziativen Arrays wird mit folgender Syntax auf ein Element im Array zugegriffen:

```
identifier(index)
```

In beiden Arrays gehört der Wert von `index` dem Typ `PLS_INTEGER` an.

- Referenzieren Sie die erste Zeile im assoziativen Array `ename_table` mit:
`ename_table(1)`
- Referenzieren Sie die achte Zeile im assoziativen Array `hiredate_table` mit:
`hiredate_table(8)`

Hinweise

- Die Größe von `PLS_INTEGER` liegt zwischen -2.147.483.647 und 2.147.483.647, das heißt, der Primärschlüsselwert kann negativ sein. Die Indizierung muss nicht mit 1 beginnen.
- Die Methode `exists(i)` gibt `TRUE` zurück, wenn eine Zeile mit dem Index *i* zurückgegeben wird. Mit der Methode `exists` verhindern Sie, dass aufgrund eines nicht vorhandenen Tabellenelements eine Fehlermeldung ausgelöst wird.
- Das vollständige Codebeispiel finden Sie unter der Folie 21_sa in `code_ex_07.sql`.

INDEX BY-Tabellen – Methoden

Die folgenden Methoden vereinfachen die Verwendung von assoziativen Arrays:

- EXISTS
- COUNT
- FIRST
- LAST
- PRIOR
- NEXT
- DELETE



Copyright © 2014, Oracle und/oder verbundene Unternehmen. All rights reserved. Alle Rechte vorbehalten.

Methoden für INDEX BY-Tabellen sind Built-In-Prozeduren oder -Funktionen, die für ein assoziatives Array ausgeführt und mit der Punktschreibweise aufgerufen werden.

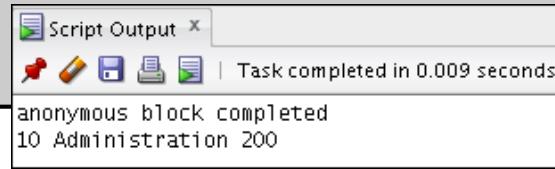
Syntax: `table_name.method_name[(parameters)]`

Methode	Beschreibung
EXISTS (<i>n</i>)	Gibt TRUE zurück, wenn das <i>n</i> -te Element in einem assoziativen Array existiert
COUNT	Gibt die Anzahl der Elemente zurück, die ein assoziatives Array derzeit enthält
FIRST	<ul style="list-style-type: none">• Gibt die erste (kleinste) Indexnummer in einem assoziativen Array zurück• Gibt NULL zurück, wenn das assoziative Array leer ist
LAST	<ul style="list-style-type: none">• Gibt die letzte (größte) Indexnummer in einem assoziativen Array zurück• Gibt NULL zurück, wenn das assoziative Array leer ist
PRIOR (<i>n</i>)	Gibt die Indexnummer zurück, die Index <i>n</i> in einem assoziativen Array vorangeht
NEXT (<i>n</i>)	Gibt die Indexnummer zurück, die auf Index <i>n</i> in einem assoziativen Array folgt
DELETE	<ul style="list-style-type: none">• Mit DELETE werden alle Elemente aus einem assoziativen Array entfernt.• Mit DELETE (<i>n</i>) wird das <i>n</i>-te Element aus einem assoziativen Array entfernt.• Mit DELETE (<i>m</i>, <i>n</i>) werden alle Elemente aus einem assoziativen Array entfernt, die im Bereich <i>m</i> ... <i>n</i> liegen.

INDEX BY-Record-Tabellen

Assoziatives Array definieren, das eine ganze Zeile aus einer Tabelle aufnimmt

```
DECLARE
  TYPE dept_table_type
  IS
    TABLE OF departments%ROWTYPE INDEX BY VARCHAR2(20);
    dept_table dept_table_type;
    -- Each element of dept_table is a record
  BEGIN
    SELECT * INTO dept_table(1) FROM departments
    WHERE department_id = 10;
    DBMS_OUTPUT.PUT_LINE(dept_table(1).department_id || ' ' ||
    dept_table(1).department_name || ' ' ||
    dept_table(1).manager_id);
  END;
/
```



ORACLE

Copyright © 2014, Oracle und/oder verbundene Unternehmen. All rights reserved. Alle Rechte vorbehalten.

Wie bereits erwähnt, kann ein mit einem skalaren Datentyp deklariertes assoziatives Array jeweils nur die Details einer Spalte in einer Datenbanktabelle speichern. In vielen Fällen müssen Sie jedoch alle durch eine Abfrage abgerufenen Spalten speichern. Hierfür steht die Option der INDEX BY Record-Tabelle zur Verfügung. Sie ermöglicht, dass eine Arraydefinition die Informationen zu allen Feldern einer Datenbanktabelle aufnimmt.

Record-Tabellen erstellen und referenzieren

Wie das Beispiel für ein assoziatives Array auf der Folie zeigt, können Sie:

- mit dem Attribut %ROWTYPE einen Record deklarieren, der eine Zeile in einer Datenbanktabelle repräsentiert
- Felder im Array dept_table referenzieren, da jedes Element des Arrays ein Record ist

Das Attribut %ROWTYPE und der zusammengesetzte Datentyp PL/SQL-Record weisen folgende Unterschiede auf:

- PL/SQL-Record-Typen können benutzerdefiniert sein. %ROWTYPE definiert den Record implizit.
- PL/SQL-Records ermöglichen die Angabe von Feldern und Datentypen bei der Deklaration. %ROWTYPE ermöglicht keine Angabe von Feldern. Das Attribut %ROWTYPE stellt eine Tabellenzeile mit allen Feldern dar, die sich aus der Tabellendefinition ergeben.
- Benutzerdefinierte Records sind statisch, %ROWTYPE-Records dynamisch. Sie basieren auf einer Tabellenstruktur. Wenn sich die Tabellenstruktur ändert, ändert sich auch die Record-Struktur.

INDEX BY-Record-Tabellen – 2. Beispiel

```
DECLARE
    TYPE emp_table_type IS TABLE OF
        employees%ROWTYPE INDEX BY PLS_INTEGER;
    my_emp_table emp_table_type;
    max_count      NUMBER(3) := 104;
BEGIN
    FOR i IN 100..max_count
    LOOP
        SELECT * INTO my_emp_table(i) FROM employees
        WHERE employee_id = i;
    END LOOP;
    FOR i IN my_emp_table.FIRST..my_emp_table.LAST
    LOOP
        DBMS_OUTPUT.PUT_LINE(my_emp_table(i).last_name);
    END LOOP;
END;
/
```

ORACLE

Copyright © 2014, Oracle und/oder verbundene Unternehmen. All rights reserved. Alle Rechte vorbehalten.

Im Beispiel auf der Folie wird ein assoziatives Array mit der Option der INDEX BY-Record-Tabelle deklariert. Sie soll vorübergehend die Details von Mitarbeitern speichern, deren Personalnummer zwischen 100 und 104 liegt. Der Variablenname für das Array lautet `emp_table_type`.

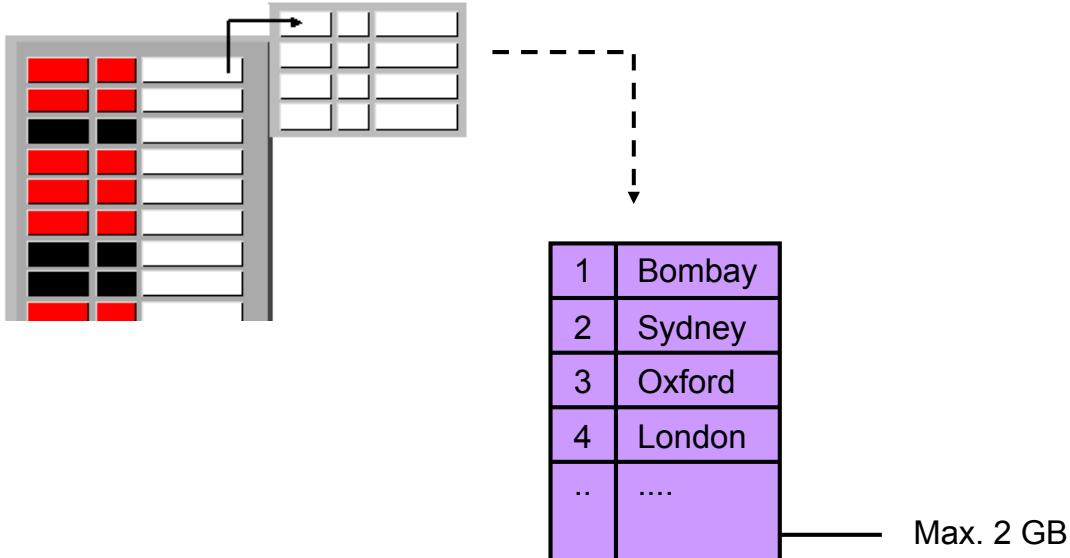
Die Informationen der Mitarbeiter werden mithilfe einer Schleife aus der Tabelle EMPLOYEES abgerufen und im Array gespeichert. Mit einer weiteren Schleife werden die Nachnamen aus dem Array ausgegeben. Beachten Sie, wie die Methoden `first` und `last` im Beispiel verwendet werden.

Hinweis: Die Folie veranschaulicht eine Möglichkeit der Verwendung eines assoziativen Arrays mit der Methode der INDEX BY-Record-Tabelle. Die in der Lektion "Explizite Cursor" behandelten Cursor stellen jedoch eine effizientere Methode hierfür dar.

Das Codebeispiel führt zu folgendem Ergebnis:

```
Script Output x
anonymous block completed
King
Kochhar
De Haan
Hunold
Ernst
```

Nested Tables



ORACLE

Copyright © 2014, Oracle und/oder verbundene Unternehmen. All rights reserved. Alle Rechte vorbehalten.

Nested Tables bieten eine ähnliche Funktionalität wie assoziative Arrays, werden jedoch anders implementiert.

- Im Gegensatz zu assoziativen Arrays sind Nested Tables ein gültiger Datentyp in Tabellen auf Schemaebene. Daher können Nested Tables im Unterschied zu assoziativen Arrays in der Datenbank gespeichert werden.
- Die Größe von Nested Tables kann sich dynamisch erhöhen, liegt allerdings maximal bei 2 GB.
- Der Schlüssel darf kein negativer Wert sein (was im assoziativen Array möglich ist). Die erste Spalte wird zwar als Schlüssel referenziert, aber eigentlich enthalten Nested Tables keine Schlüssel, sondern eine Spalte mit Zahlen.
- Sie können in Nested Tables an beliebigen Stellen Elemente löschen, wodurch Lücken in der Reihenfolge der Schlüssel entstehen. Die Zeilen von Nested Tables haben keine bestimmte Reihenfolge.
- Wenn Sie Werte aus Nested Tables abrufen, werden den Zeilen aufeinanderfolgende untergeordnete Skripte zugewiesen, die mit 1 beginnen.

Syntax:

```
TYPE type_name IS TABLE OF
  {column_type | variable%TYPE
  | table.column%TYPE} [NOT NULL]
  | table.%ROWTYPE
```

Beispiel:

```
TYPE location_type IS TABLE OF locations.city%TYPE;
  offices location_type;
```

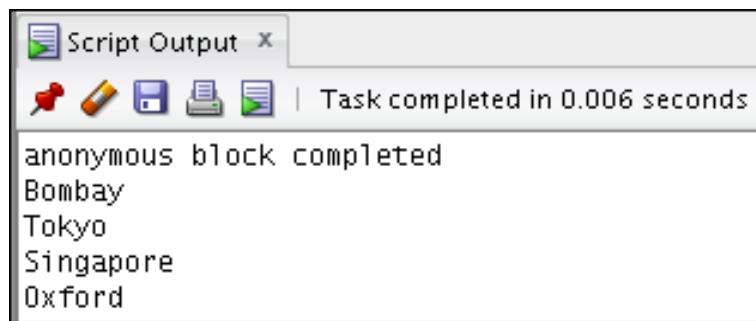
Nested Tables, die Sie nicht initialisieren, werden automatisch auf NULL initialisiert. Sie können die Nested Table `offices` mithilfe eines Konstruktors initialisieren:

```
offices := location_type('Bombay', 'Tokyo', 'Singapore', 'Oxford');
```

Das vollständige Codebeispiel und die Ausgabe sehen wie folgt aus:

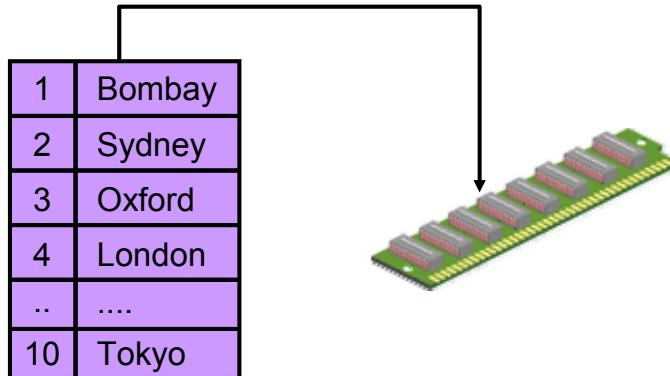
```
SET SERVEROUTPUT ON;

DECLARE
  TYPE location_type IS TABLE OF locations.city%TYPE;
  offices location_type;
  table_count NUMBER;
BEGIN
  offices := location_type('Bombay', 'Tokyo', 'Singapore',
                           'Oxford');
  FOR i in 1.. offices.count() LOOP
    DBMS_OUTPUT.PUT_LINE(offices(i));
  END LOOP;
END;
/
```



The screenshot shows the 'Script Output' window from Oracle SQL Developer. The window title is 'Script Output'. At the top, there are several icons: a red screwdriver, a yellow pencil, a blue folder, a green printer, and a blue file. To the right of these icons, the text 'Task completed in 0.006 seconds' is displayed. Below this, the output of the anonymous block is shown in a monospaced font:
anonymous block completed
Bombay
Tokyo
Singapore
Oxford

VARRAYS



ORACLE

Copyright © 2014, Oracle und/oder verbundene Unternehmen. All rights reserved. Alle Rechte vorbehalten.

Ein Array mit variabler Größe (VARRAY) ähnelt einem assoziativen Array, ist aber größtenbeschränkt.

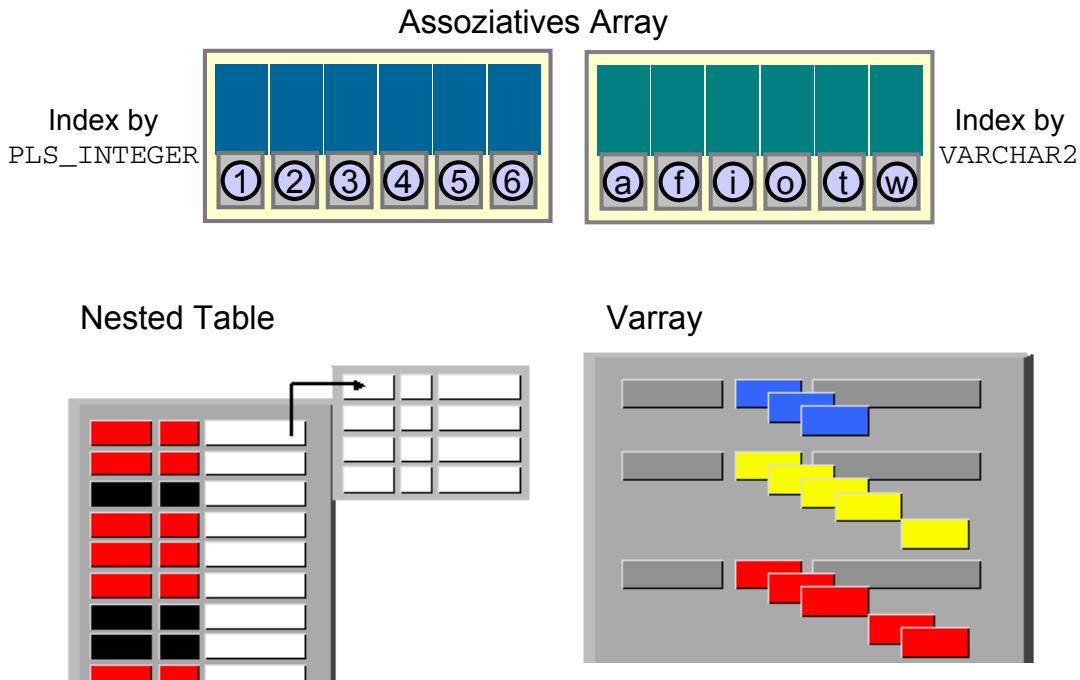
- Ein VARRAY ist in Tabellen auf Schemaebene gültig.
- Elemente vom Typ VARRAY werden als VARRAYS bezeichnet.
- VARRAYS haben eine feste Obergrenze, die bei der Deklaration angegeben werden muss. Dies entspricht Arrays in der Programmiersprache C. Die maximale Größe eines VARRAY beträgt wie bei Nested Tables 2 GB.
- Nested Tables und VARRAYS unterscheiden sich im physischen Speichermodus. Die Elemente eines VARRAY werden inline mit den Tabellendaten gespeichert, sofern das VARRAY nicht größer als 4 KB ist. Nested Tables werden dagegen immer out-of-line gespeichert.
- Mithilfe von SQL können Sie einen VARRAY-Typ in der Datenbank erstellen.

Beispiel:

```
TYPE location_type IS VARRAY(3) OF locations.city%TYPE;  
offices location_type;
```

Die Größe dieses VARRAY ist auf 3 eingeschränkt. Sie können ein VARRAY mithilfe von Konstruktoren initialisieren. Wenn Sie versuchen, das VARRAY mit mehr als drei Elementen zu initialisieren, wird die Fehlermeldung "Subscript outside of limit" angezeigt.

Collection-Typen – Zusammenfassung



ORACLE

Copyright © 2014, Oracle und/oder verbundene Unternehmen. All rights reserved. Alle Rechte vorbehalten.

Assoziative Arrays

Assoziative Arrays sind Gruppen von Schlüssel/Werte-Paaren, bei denen jeder Schlüssel eindeutig ist und für die Suche nach einem entsprechenden Wert im Array verwendet wird. Der Schlüssel kann auf Ganzzahlen oder Zeichen basieren. Der Arraywert kann den skalaren Datentyp (einzelner Wert) oder den Record-Datentyp (mehrere Werte) aufweisen.

Da assoziative Arrays zum Speichern von temporären Daten vorgesehen sind, ist ihre Verwendung in SQL-Anweisungen wie `INSERT` und `SELECT INTO` nicht möglich.

Nested Tables

Eine Nested Table enthält eine Gruppe von Werten. Das heißt, sie entspricht einer Tabelle in einer Tabelle. Nested Tables haben kein festes Format, das heißt, die Größe der Tabelle kann sich dynamisch erhöhen. Nested Tables sind sowohl in PL/SQL als auch in der Datenbank verfügbar. In PL/SQL entsprechen Nested Tables eindimensionalen Arrays, deren Größe sich dynamisch erhöhen kann.

Varrays

Arrays variabler Größe beziehungsweise Varrays sind ebenfalls Collections mit homogenen Elementen, die eine feste Anzahl von Elementen enthalten. (Sie können die Anzahl der Elemente jedoch zur Laufzeit ändern.) Varrays verwenden sequenzielle Nummern als untergeordnete Skripte. Sie können äquivalente SQL-Typen definieren, um Varrays in Datenbanktabellen speichern zu können.

Quiz

Geben Sie an, in welchen Fällen Sie das Attribut %ROWTYPE verwenden können.

- a. Wenn Sie die Struktur der zugrunde liegenden Datenbanktabelle nicht kennen
- b. Wenn Sie eine ganze Zeile aus einer Tabelle abrufen möchten
- c. Wenn Sie eine Variable entsprechend einer zuvor deklarierten Variablen oder Datenbankspalte deklarieren möchten



Copyright © 2014, Oracle und/oder verbundene Unternehmen. All rights reserved. Alle Rechte vorbehalten.

Richtige Antworten: a, b

Vorteile des Attributs %ROWTYPE

Verwenden Sie das Attribut %ROWTYPE, wenn Sie die Struktur der zugrunde liegenden Datenbanktabelle nicht kennen.

Der Hauptvorteil von %ROWTYPE liegt in der einfacheren Verwaltung. Mit %ROWTYPE stellen Sie sicher, dass sich die Datentypen der mit dem Attribut deklarierten Variablen dynamisch an Änderungen der zugrunde liegenden Tabelle anpassen. Falls die Spalten einer Tabelle durch eine DDL-Anweisung geändert werden, wird die PL/SQL-Programmeinheit invalidiert. Nach der Rekompilierung des Programms entspricht dieses automatisch dem neuen Tabellenformat.

Das Attribut %ROWTYPE ist speziell dann nützlich, wenn eine ganze Zeile aus einer Tabelle abgerufen werden soll. Ohne dieses Attribut müssten Sie für jede der mit der SELECT-Anweisung abgerufenen Spalten eine Variable deklarieren.

Zusammenfassung

In dieser Lektion haben Sie Folgendes gelernt:

- PL/SQL-Variablen mit zusammengesetzten Datentypen definieren und referenzieren
 - PL/SQL-Records
 - Assoziative Arrays
 - INDEX BY-Tabellen
 - INDEX BY-Record-Tabellen
- PL/SQL-Records mit dem Attribut %ROWTYPE definieren
- Die drei PL/SQL-Collection-Typen vergleichen und gegenüberstellen:
 - Assoziative Arrays
 - Nested Tables
 - VARRAYS



Copyright © 2014, Oracle und/oder verbundene Unternehmen. All rights reserved. Alle Rechte vorbehalten.

Ein PL/SQL-Record ist eine Collection individueller Felder, die eine Zeile in einer Tabelle darstellen. Mithilfe von Records können Sie die Daten in einer Struktur gruppieren und diese Struktur dann als eine Entität oder logische Einheit bearbeiten. Auf diese Weise können Sie die Codierung reduzieren und den Code leicht verwaltbar und lesbar gestalten.

Wie PL/SQL-Records sind auch PL/SQL-Collections ein zusammengesetzter Datentyp. Typen von PL/SQL-Collections:

- Assoziative Arrays (auch als INDEX BY-Tabellen bezeichnet). Objekte vom Typ TABLE, die Datenbanktabellen ähneln, sich von diesen aber leicht unterscheiden. Die sogenannten INDEX BY-Tabellen ermöglichen einen arrayähnlichen Zugriff auf Zeilen mithilfe eines Primarschlüssels. Die Größe von assoziativen Arrays ist unbeschränkt.
- Nested Tables. Der Schlüssel für Nested Tables darf kein negativer Wert sein (was in INDEX BY-Tabellen möglich ist). Die Schlüssel sind außerdem der Reihe nach angeordnet.
- Arrays mit variabler Größe (VARRAYS). Ein VARRAY ähnelt einem assoziativen Array, ist jedoch in der Größe beschränkt.

Übungen zu Lektion 7 – Übersicht

Diese Übung behandelt folgende Themen:

- Assoziative Arrays deklarieren
- Daten mithilfe von assoziativen Arrays verarbeiten
- PL/SQL-Record deklarieren
- Daten mit einem PL/SQL-Record verarbeiten



Copyright © 2014, Oracle und/oder verbundene Unternehmen. All rights reserved. Alle Rechte vorbehalten.

In diesen Übungen definieren, erstellen und verwenden Sie assoziative Arrays und PL/SQL-Records.

Explizite Cursor

8

ORACLE

Copyright © 2014, Oracle und/oder verbundene Unternehmen. All rights reserved. Alle Rechte vorbehalten.

Ziele

Nach Ablauf dieser Lektion haben Sie folgende Ziele erreicht:

- Zwischen impliziten und expliziten Cursorn unterscheiden
- Gründe für die Verwendung expliziter Cursor erörtern
- Explizite Cursor deklarieren und kontrollieren
- Daten mit einfachen Schleifen und Cursor FOR-Schleifen lesen
- Cursor mit Parametern deklarieren und verwenden
- Zeilen mit der Klausel FOR UPDATE sperren
- Aktuelle Zeile mit der Klausel WHERE CURRENT OF referenzieren



Copyright © 2014, Oracle und/oder verbundene Unternehmen. All rights reserved. Alle Rechte vorbehalten.

Sie haben implizite Cursor kennengelernt, die von PL/SQL automatisch erstellt werden, wenn Sie die SQL-Anweisung SELECT oder eine DML-Anweisung ausführen. In dieser Lektion werden explizite Cursor behandelt. Sie lernen den Unterschied zwischen impliziten und expliziten Cursorn kennen. Darüber hinaus wird erläutert, wie Sie einfache Cursor und Cursor mit Parametern deklarieren und kontrollieren.

Agenda

- Explizite Cursor – Definition
- Explizite Cursor verwenden
- Cursor mit Parametern
- Zeilen sperren und die aktuelle Zeile referenzieren

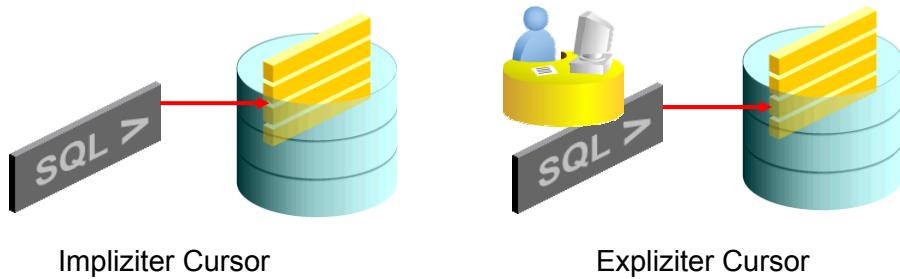
ORACLE

Copyright © 2014, Oracle und/oder verbundene Unternehmen. All rights reserved. Alle Rechte vorbehalten.

Cursor

Jeder vom Oracle-Server ausgeführten SQL-Anweisung ist ein eigener Cursor zugeordnet:

- Implizite Cursor: Von PL/SQL für alle DML- und PL/SQL-Anweisungen vom Typ `SELECT` deklariert und verwaltet
- Explizite Cursor: Vom Programmierer deklariert und verwaltet



ORACLE

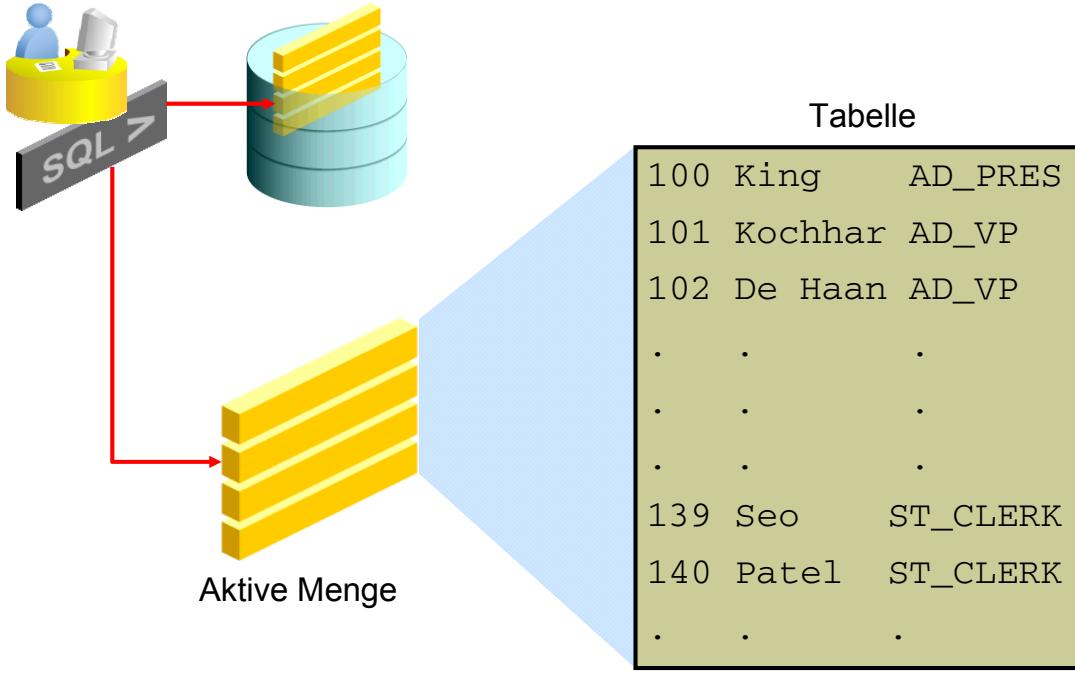
Copyright © 2014, Oracle und/oder verbundene Unternehmen. All rights reserved. Alle Rechte vorbehalten.

Um SQL-Anweisungen auszuführen und Verarbeitungsinformationen zu speichern, verwendet der Oracle-Server Arbeitsbereiche (sogenannte *private SQL-Bereiche*). Mit expliziten Cursorn können Sie private SQL-Bereiche benennen und auf die darin gespeicherten Informationen zugreifen.

Cursortyp	Beschreibung
Implizit	Implizite Cursor werden von PL/SQL implizit für alle DML-Anweisungen sowie für <code>SELECT</code> -Anweisungen in PL/SQL deklariert.
Explizit	Für Abfragen, die mehrere Zeilen zurückgeben, werden explizite Cursor vom Programmierer deklariert, verwaltet und über spezielle Anweisungen im ausführbaren Teil des Blockes bearbeitet.

Zur Verarbeitung einer SQL-Anweisung, der kein explizit deklarierter Cursor zugeordnet ist, öffnet der Oracle-Server einen Cursor implizit. Mit PL/SQL können Sie den zuletzt verwendeten impliziten Cursor als SQL-Cursor referenzieren.

Vorgänge mit expliziten Cursorn



ORACLE

Copyright © 2014, Oracle und/oder verbundene Unternehmen. All rights reserved. Alle Rechte vorbehalten.

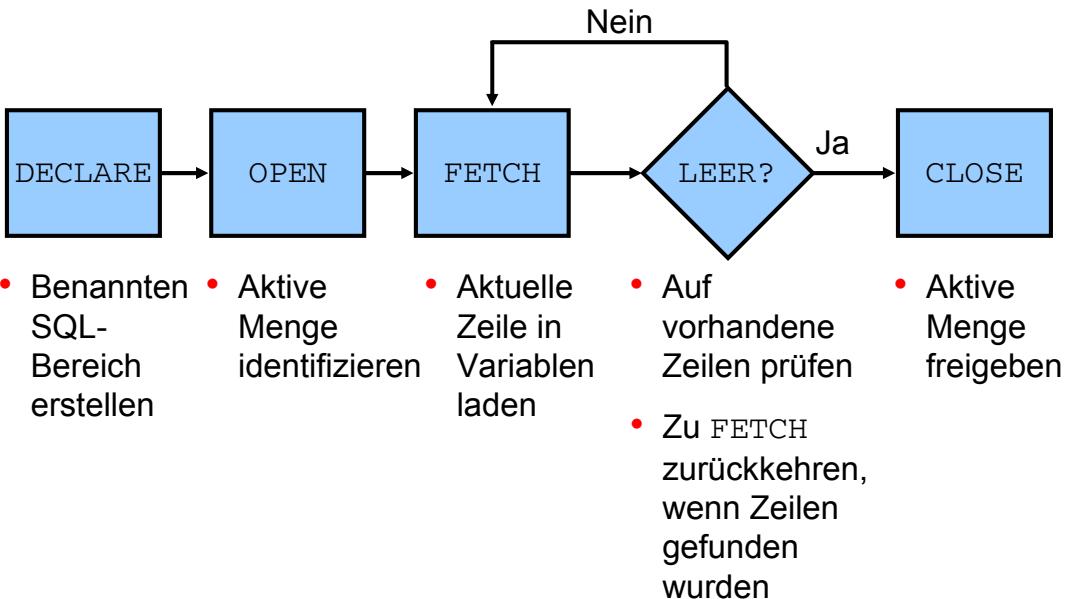
Sie deklarieren explizite Cursor in PL/SQL, wenn eine SELECT-Anweisung mehrere Zeilen zurückgibt. Sie können jede von der SELECT-Anweisung zurückgegebene Zeile verarbeiten.

Die von einer Multiple Row-Abfrage zurückgegebene Zeilenmenge wird als *aktive Menge* bezeichnet. Ihre Größe hängt von der Anzahl der Zeilen ab, die den Suchkriterien entsprechen. Das Diagramm auf der Folie veranschaulicht, wie ein expliziter Cursor auf die aktuelle Zeile der aktiven Menge "zeigt". Das Programm kann auf diese Weise die Zeilen der Reihe nach verarbeiten.

Funktionen expliziter Cursor:

- Ermöglichen die zeilenweise Verarbeitung über die erste von der Abfrage zurückgegebene Zeile hinaus
- Verfolgen, welche Zeile derzeit verarbeitet wird
- Ermöglichen Programmierern die manuelle Kontrolle über explizite Cursor im PL/SQL-Block

Explizite Cursor kontrollieren



ORACLE

Copyright © 2014, Oracle und/oder verbundene Unternehmen. All rights reserved. Alle Rechte vorbehalten.

Nachdem Sie nun ein grundsätzliches Verständnis über Cursor erworben haben, wenden wir uns der Schrittfolge zu ihrer Verwendung zu.

1. Deklarieren Sie einen Cursor im deklarativen Bereich eines PL/SQL-Blockes, indem Sie ihn benennen und die Struktur der zuzuordnenden Abfrage definieren.
2. Öffnen Sie den Cursor.
Die OPEN-Anweisung führt die Abfrage aus, bindet alle referenzierten Variablen und positioniert den Cursor in der ersten Zeile. Die von der Abfrage identifizierten Zeilen werden als *aktive Menge* bezeichnet. Sie stehen jetzt für Zeilenlesevorgänge (Fetching) zur Verfügung.
3. Lesen Sie Daten aus dem Cursor.
Im Ablaufdiagramm auf der Folie wird nach jeder gelesenen Zeile geprüft, ob der Cursor weitere Zeilen enthält. Wenn alle Zeilen verarbeitet wurden, müssen Sie den Cursor schließen.
Die FETCH-Anweisung ruft die aktuelle Zeile ab und versetzt den Cursor in die nächste Zeile, bis keine weitere Zeile mehr vorhanden ist oder eine bestimmte Bedingung erfüllt wurde.
4. Schließen Sie den Cursor.
Die CLOSE-Anweisung gibt die aktive Zeilenmenge frei. Sie können den Cursor jetzt erneut öffnen, um eine neue aktive Menge zu erstellen.

Agenda

- Explizite Cursor – Definition
- Explizite Cursor verwenden
- Cursor mit Parametern
- Zeilen sperren und die aktuelle Zeile referenzieren

ORACLE

Copyright © 2014, Oracle und/oder verbundene Unternehmen. All rights reserved. Alle Rechte vorbehalten.

Cursor deklarieren

Syntax:

```
CURSOR cursor_name IS  
    select_statement;
```

Beispiele:

```
DECLARE  
    CURSOR c_emp_cursor IS  
        SELECT employee_id, last_name FROM employees  
        WHERE department_id = 30;
```

```
DECLARE  
    v_locid NUMBER := 1700;  
    CURSOR c_dept_cursor IS  
        SELECT * FROM departments  
        WHERE location_id = v_locid;  
    ...
```

ORACLE

Copyright © 2014, Oracle und/oder verbundene Unternehmen. All rights reserved. Alle Rechte vorbehalten.

Die Syntax zum Deklarieren von Cursorn ist auf der Folie dargestellt. Für die Syntax gilt:

<i>cursor_name</i>	PL/SQL-ID
<i>select_statement</i>	SELECT-Anweisung ohne INTO-Klausel

Die aktive Menge eines Cursors wird durch die SELECT-Anweisung in der Cursordeklaration bestimmt. In PL/SQL muss eine SELECT-Anweisung eine INTO-Klausel enthalten. Die SELECT-Anweisung in der Cursordeklaration darf jedoch keine INTO-Klausel enthalten. Das liegt daran, dass Sie im deklarativen Bereich nur einen Cursor definieren, aber keine Zeilen in den Cursor lesen.

Hinweise

- Nehmen Sie die INTO-Klausel nicht in die Cursordeklaration auf, da sie später in der FETCH-Anweisung enthalten ist.
- Falls die Zeilen in einer bestimmten Reihenfolge verarbeitet werden sollen, verwenden Sie in der Abfrage eine ORDER BY-Klausel.
- Der Cursor kann eine beliebige gültige SELECT-Anweisung sein, einschließlich Joins, Unterabfragen und so weiter.

Um die Spalten `employee_id` und `last_name` für die Mitarbeiter abzurufen, die in der Abteilung mit der `department_id` 30 arbeiten, deklarieren Sie den Cursor `c_emp_cursor`.

Für den Abruf aller Details zur Abteilung mit der `location_id` 1700 deklarieren Sie den Cursor `c_dept_cursor`. Beachten Sie, dass beim Deklarieren des Cursors eine Variable verwendet wird. Diese Variablen werden als Bind-Variablen bezeichnet. Sie müssen sichtbar sein, wenn Sie den Cursor deklarieren. Die Variablen werden nur einmal beim Öffnen des Cursors geprüft. Sie haben gelernt, dass Sie in PL/SQL explizite Cursor verwenden, um mehrere Zeilen abzurufen und zu bearbeiten. Dieses Beispiel zeigt jedoch, dass explizite Cursor auch verwendet werden können, wenn die `SELECT`-Anweisung nur eine Zeile zurückgibt.

Cursor öffnen

```
DECLARE
    CURSOR c_emp_cursor IS
        SELECT employee_id, last_name FROM employees
        WHERE department_id =30;
    ...
BEGIN
    OPEN c_emp_cursor;
```



Copyright © 2014, Oracle und/oder verbundene Unternehmen. All rights reserved. Alle Rechte vorbehalten.

Die `OPEN`-Anweisung führt die dem Cursor zugeordnete Abfrage aus, identifiziert die aktive Menge und positioniert den Cursorzeiger in der ersten Zeile. Nehmen Sie die `OPEN`-Anweisung in den ausführbaren Bereich des PL/SQL-Blockes auf.

`OPEN` ist eine ausführbare Anweisung, die die folgenden Vorgänge ausführt:

1. Sie weist einem Kontextbereich dynamisch Speicher zu.
2. Sie parst die `SELECT`-Anweisung.
3. Sie bindet die Eingabeveriablen. (Die Anweisung legt die Werte für die Eingabeveriablen fest, indem sie ihre Speicheradressen abruft.)
4. Sie identifiziert die aktive Menge (die den Suchkriterien entsprechende Zeilenmenge). Wenn Sie die `OPEN`-Anweisung ausführen, werden keine Zeilen in der aktiven Menge in Variablen gelesen. Stattdessen ruft die `FETCH`-Anweisung die Zeilen aus dem Cursor in die Variablen ab.
5. Sie positioniert den Zeiger in der ersten Zeile der aktiven Menge.

Hinweis: Wenn die Abfrage bei geöffnetem Cursor keine Zeilen zurückgibt, löst PL/SQL keine Exception aus.

Daten aus Cursorn lesen

```
DECLARE
    CURSOR c_emp_cursor IS
        SELECT employee_id, last_name FROM employees
        WHERE department_id =30;
        v_empno employees.employee_id%TYPE;
        v_lname employees.last_name%TYPE;
BEGIN
    OPEN c_emp_cursor;
    FETCH c_emp_cursor INTO v_empno, v_lname;
    DBMS_OUTPUT.PUT_LINE( v_empno || ' ' || v_lname);
END;
/
```

```
anonymous block completed
114 Raphaely
```

ORACLE

Copyright © 2014, Oracle und/oder verbundene Unternehmen. All rights reserved. Alle Rechte vorbehalten.

Mit einer `FETCH`-Anweisung rufen Sie die Zeilen nacheinander aus dem Cursor ab. Nach jedem Zeilenlesevorgang rückt der Cursor zur nächsten Zeile der aktiven Menge vor. Mit dem Attribut `%NOTFOUND` können Sie bestimmen, ob die aktive Menge vollständig gelesen wurde.

Sehen Sie sich das Beispiel auf der Folie an. Um die gelesenen Werte aus dem Cursor aufzunehmen, deklarieren Sie die beiden Variablen `v_empno` und `v_lname`. Schauen Sie sich die `FETCH`-Anweisung an.

Sie haben die Werte erfolgreich aus dem Cursor in die Variablen gelesen. Allerdings arbeiten in Abteilung 30 sechs Mitarbeiter, und es wurde nur eine Zeile gelesen. Um alle Zeilen abzurufen, müssen Sie Schleifen verwenden. Auf der nächsten Folie sehen Sie, wie Sie mit einer Schleife alle Zeilen lesen.

Die `FETCH`-Anweisung führt die folgenden Vorgänge aus:

1. Sie liest die Daten für die aktuelle Zeile in die PL/SQL-Ausgabevariablen.
2. Sie rückt den Zeiger in der aktiven Menge zur nächsten Zeile vor.

Sie können in die `INTO`-Klausel der `FETCH`-Anweisung genauso viele Variablen aufnehmen, wie Spalten in der `SELECT`-Anweisung angegeben sind. Stellen Sie dabei sicher, dass die Datentypen kompatibel sind. Ordnen Sie die einzelnen Variablen entsprechend den Positionen der Spalten an. Sie können auch einen Record für den Cursor definieren und in der Klausel `FETCH INTO` referenzieren. Prüfen Sie schließlich, ob der Cursor Zeilen enthält. Wenn die `FETCH`-Anweisung keine Werte erfasst, sind in der aktiven Menge keine weiteren zu verarbeitenden Zeilen vorhanden, und es wird kein Fehler aufgezeichnet.

Daten aus Cursorn lesen

```
DECLARE
    CURSOR c_emp_cursor IS
        SELECT employee_id, last_name FROM employees
        WHERE department_id =30;
    v_empno employees.employee_id%TYPE;
    v_lname employees.last_name%TYPE;
BEGIN
    OPEN c_emp_cursor;
    LOOP
        FETCH c_emp_cursor INTO v_empno, v_lname;
        EXIT WHEN c_emp_cursor%NOTFOUND;
        DBMS_OUTPUT.PUT_LINE( v_empno || ' ' || v_lname);
    END LOOP;
END;
/
```

ORACLE

Copyright © 2014, Oracle und/oder verbundene Unternehmen. All rights reserved. Alle Rechte vorbehalten.

Beachten Sie, dass die Zeilen mit einer einfachen Schleife gelesen werden. Das Cursorattribut %NOTFOUND testet zudem auf die EXIT-Bedingung. Die Ausgabe des PL/SQL-Blockes lautet:

```
anonymous block completed
114  Raphaely
115  Khoo
116  Baida
117  Tobias
118  Himuro
119  Colmenares
```

Cursor schließen

```
...
LOOP
    FETCH c_emp_cursor INTO empno, lname;
    EXIT WHEN c_emp_cursor%NOTFOUND;
    DBMS_OUTPUT.PUT_LINE( v_empno || ' ' || v_lname);
END LOOP;
CLOSE c_emp_cursor;
END;
/
```

ORACLE

Copyright © 2014, Oracle und/oder verbundene Unternehmen. All rights reserved. Alle Rechte vorbehalten.

Eine CLOSE-Anweisung deaktiviert den Cursor, gibt den Kontextbereich frei und löscht die Prozedurvariable der aktiven Menge. Schließen Sie den Cursor, sobald die FETCH-Anweisung vollständig verarbeitet ist. Sie können den Cursor bei Bedarf erneut öffnen. Dies ist jedoch nur möglich, wenn Sie den Cursor geschlossen haben. Wenn Sie versuchen, Daten aus einem geschlossenen Cursor abzurufen, wird die Exception INVALID_CURSOR ausgelöst.

Hinweis: Der PL/SQL-Block kann zwar beendet werden, ohne den Cursor zu schließen, allerdings sollten Sie deklarierte Cursor stets explizit schließen, um Ressourcen freizugeben. Die maximale Anzahl der geöffneten Cursor pro Session ist begrenzt. Sie wird in der Parameterdatei der Datenbank durch den Parameter OPEN_CURSORS festgelegt. (Der Standardwert dieses Parameters lautet OPEN_CURSORS = 50.)

Cursor und Records

Zeilen der aktiven Menge durch Lesen der Werte in einen PL/SQL-Record verarbeiten

```
DECLARE
    CURSOR c_emp_cursor IS
        SELECT employee_id, last_name FROM employees
        WHERE department_id =30;
        v_emp_record    c_emp_cursor%ROWTYPE;
BEGIN
    OPEN c_emp_cursor;
    LOOP
        FETCH c_emp_cursor INTO v_emp_record;
        EXIT WHEN c_emp_cursor%NOTFOUND;
        DBMS_OUTPUT.PUT_LINE( v_emp_record.employee_id
                                ||' '||v_emp_record.last_name);
    END LOOP;
    CLOSE c_emp_cursor;
END;
```

ORACLE

Copyright © 2014, Oracle und/oder verbundene Unternehmen. All rights reserved. Alle Rechte vorbehalten.

Sie haben bereits gelernt, dass Sie Records definieren können, die in einer Tabelle in Spaltenform vorhanden sind. Records lassen sich auch auf Basis der gewählten Spaltenliste in expliziten Cursorn definieren. Dies ist nützlich, wenn Sie die Zeilen der aktiven Menge verarbeiten, da Sie Daten einfach in den Record lesen können. Die Werte der Zeilen werden somit direkt in die entsprechenden Felder des Records geladen.

```
anonymous block completed
114 Raphaely
115 Khoo
116 Baida
117 Tobias
118 Himuro
119 Colmenares
```

Cursor FOR-Schleifen

Syntax:

```
FOR record_name IN cursor_name LOOP
    statement1;
    statement2;
    . . .
END LOOP;
```

- Mit Cursor FOR-Schleifen lässt sich die Verarbeitung expliziter Cursor verkürzen.
- Vorgänge vom Typ OPEN, FETCH, EXIT und CLOSE werden implizit durchgeführt.
- Der Record wird implizit deklariert.

ORACLE

Copyright © 2014, Oracle und/oder verbundene Unternehmen. All rights reserved. Alle Rechte vorbehalten.

Sie haben gelernt, wie Sie Daten mithilfe von einfachen Schleifen aus Cursorn lesen. Jetzt erfahren Sie, wie Sie Zeilen in einem expliziten Cursor mit einer Cursor FOR-Schleife verarbeiten. Sie verkürzen damit den Vorgang, da Sie den Cursor öffnen, bei jeder Schleifeniteration eine Zeile lesen, die Schleife nach der letzten zu verarbeitenden Schleife beenden und den Cursor automatisch schließen. Die Schleife wird am Ende der Iteration (nachdem die letzte Zeile gelesen wurde) automatisch beendet.

Für die Syntax gilt:

record_name	Name des implizit deklarierten Records
cursor_name	PL/SQL-ID für den zuvor deklarierten Cursor

Richtlinien

- Deklarieren Sie nicht den Record, der die Schleife kontrolliert. Dieser wird implizit deklariert.
- Testen Sie die Cursorattribute gegebenenfalls während der Ausführung der Schleife.
- Falls erforderlich, geben Sie die Parameter für einen Cursor in der FOR-Anweisung nach dem Cursornamen in Klammern an.

Cursor FOR-Schleifen

```
DECLARE
    CURSOR c_emp_cursor IS
        SELECT employee_id, last_name FROM employees
        WHERE department_id =30;
BEGIN
    FOR emp_record IN c_emp_cursor
    LOOP
        DBMS_OUTPUT.PUT_LINE( emp_record.employee_id
        ||' '||emp_record.last_name);
    END LOOP;
END;
/
```

```
anonymous block completed
114 Raphaely
115 Khoo
116 Baida
117 Tobias
118 Himuro
119 Colmenares
```

ORACLE

Copyright © 2014, Oracle und/oder verbundene Unternehmen. All rights reserved. Alle Rechte vorbehalten.

Das Beispiel, in dem Daten mithilfe einer einfachen Schleife aus Cursorn gelesen werden, wurde so umgeschrieben, dass eine Cursor FOR-Schleife verwendet wird.

In diesem Fall ist `emp_record` der implizit deklarierte Record. Mit diesem impliziten Record können Sie auf die gelesenen Daten zugreifen (siehe Folie). Beachten Sie, dass keine Variablen deklariert werden, um die gelesenen Daten mithilfe der `INTO`-Klausel aufzunehmen. Der Code enthält keine `OPEN`- und `CLOSE`-Anweisung, um den Cursor zu öffnen beziehungsweise zu schließen.

Attribute von expliziten Cursorn

Mit Attributen für explizite Cursor die Statusinformationen eines Cursors ermitteln

Attribut	Typ	Beschreibung
%ISOPEN	Boolesch	Wird mit TRUE ausgewertet, wenn der Cursor geöffnet ist
%NOTFOUND	Boolesch	Wird mit TRUE ausgewertet, wenn der letzte Zeilenlesevorgang keine Zeile zurückgibt
%FOUND	Boolesch	Wird mit TRUE ausgewertet, wenn der letzte Zeilenlesevorgang eine Zeile zurückgibt. Gegenstück zu %NOTFOUND
%ROWCOUNT	Numerisch	Ergibt die Anzahl der gelesenen Zeilen

ORACLE

Copyright © 2014, Oracle und/oder verbundene Unternehmen. All rights reserved. Alle Rechte vorbehalten.

Wie bei impliziten Cursorn gibt es vier Attribute, mit denen Sie die Statusinformationen eines Cursors abrufen können. Wenn Sie diese Attribute an den Namen der Cursorvariablen anhängen, geben sie nützliche Informationen darüber zurück, wie Sie eine Anweisung zum Bearbeiten von Cursorn ausführen.

Hinweis: Sie können Cursorattribute nicht direkt in einer SQL-Anweisung referenzieren.

Attribut %ISOPEN

- Zeilen können nur gelesen werden, wenn der Cursor geöffnet ist.
- Um zu prüfen, ob der Cursor geöffnet ist, vor einem Zeilesevorgang das Cursorattribut %ISOPEN verwenden

Beispiel:

```
IF NOT c_emp_cursor%ISOPEN THEN
    OPEN c_emp_cursor;
END IF;
LOOP
    FETCH c_emp_cursor...
```



Copyright © 2014, Oracle und/oder verbundene Unternehmen. All rights reserved. Alle Rechte vorbehalten.

- Zeilen können nur gelesen werden, wenn der Cursor geöffnet ist. Um zu ermitteln, ob der Cursor geöffnet ist, verwenden Sie das Cursorattribut %ISOPEN.
- Lesen Sie Zeilen in einer Schleife. Bestimmen Sie anhand von Cursorattributen, wann die Schleife beendet werden soll.
- Verwenden Sie das Cursorattribut %ROWCOUNT für folgende Aufgaben:
 - Exakte Anzahl von Zeilen verarbeiten
 - Zeilen in einer Schleife lesen und bestimmen, wann die Schleife beendet werden soll

Hinweis: %ISOPEN gibt den Cursorstatus zurück: TRUE bei geöffnetem Cursor und FALSE bei geschlossenem Cursor.

%ROWCOUNT und %NOTFOUND – Beispiel

```
DECLARE
    CURSOR c_emp_cursor IS SELECT employee_id,
        last_name FROM employees;
    v_emp_record    c_emp_cursor%ROWTYPE;
BEGIN
    OPEN c_emp_cursor;
    LOOP
        FETCH c_emp_cursor INTO v_emp_record;
        EXIT WHEN c_emp_cursor%ROWCOUNT > 10 OR
            c_emp_cursor%NOTFOUND;
        DBMS_OUTPUT.PUT_LINE( v_emp_record.employee_id
            || ' ' || v_emp_record.last_name );
    END LOOP;
    CLOSE c_emp_cursor;
END ; /
```

anonymous block completed
174 Abel
166 Ande
130 Atkinson
105 Austin
204 Baer
116 Baida
167 Banda
172 Bates
192 Bell
151 Bernstein

ORACLE

Copyright © 2014, Oracle und/oder verbundene Unternehmen. All rights reserved. Alle Rechte vorbehalten.

Im Beispiel auf der Folie werden nacheinander die ersten zehn Mitarbeiter abgerufen. Dieses Beispiel veranschaulicht, wie Sie die Attribute %ROWCOUNT und %NOTFOUND für EXIT-Bedingungen in einer Schleife verwenden können.

Cursor FOR-Schleifen mit Unterabfragen

Sie müssen den Cursor nicht deklarieren.

```
BEGIN
    FOR emp_record IN (SELECT employee_id, last_name
        FROM employees WHERE department_id =30)
    LOOP
        DBMS_OUTPUT.PUT_LINE( emp_record.employee_id
        || ' ' || emp_record.last_name);
    END LOOP;
END;
/
```

```
anonymous block completed
114 Raphaely
115 Khoo
116 Baida
117 Tobias
118 Himuro
119 Colmenares
```

ORACLE

Copyright © 2014, Oracle und/oder verbundene Unternehmen. All rights reserved. Alle Rechte vorbehalten.

Beachten Sie, dass dieser PL/SQL-Block keinen deklarativen Bereich aufweist. Der Unterschied zwischen Cursor FOR-Schleifen mit und ohne Unterabfragen liegt in der Cursordeklaration. Wenn Sie Cursor FOR-Schleifen mit Unterabfragen erstellen, müssen Sie den Cursor nicht im deklarativen Bereich deklarieren. Stattdessen müssen Sie die SELECT-Anweisung angeben, die die aktive Menge in der Schleife selbst bestimmt.

Das Beispiel zur Veranschaulichung von Cursor FOR-Schleifen wurde umgeschrieben, um eine Cursor FOR-Schleife mit Unterabfragen darzustellen.

Hinweis: Attribute von expliziten Cursorn können nicht referenziert werden, wenn Sie in einer Cursor FOR-Schleife Unterabfragen verwenden, da Sie dem Cursor keinen expliziten Namen geben können.

Agenda

- Explizite Cursor – Definition
- Explizite Cursor verwenden
- **Cursor mit Parametern**
- Zeilen sperren und die aktuelle Zeile referenzieren

ORACLE

Copyright © 2014, Oracle und/oder verbundene Unternehmen. All rights reserved. Alle Rechte vorbehalten.

Cursor mit Parametern

Syntax:

```
CURSOR cursor_name
      [ (parameter_name datatype, ... ) ]
IS
    select_statement;
```

- Parameterwerte an einen Cursor übergeben, sobald der Cursor geöffnet und die Abfrage ausgeführt wird.
- Einen expliziten Cursor mehrmals mit jeweils einer anderen aktiven Menge öffnen

```
OPEN cursor_name (parameter_value, . . . . .) ;
```

ORACLE

Copyright © 2014, Oracle und/oder verbundene Unternehmen. All rights reserved. Alle Rechte vorbehalten.

Parameter können Sie an einen Cursor übergeben. Auf diese Weise können Sie einen expliziten Cursor in einem Block mehrmals öffnen und schließen und jedes Mal eine andere aktive Menge zurückgeben. Für jede Ausführung wird der vorherige Cursor geschlossen und mit einer neuen Gruppe von Parametern neu geöffnet.

Jeder formale Parameter in der Cursordeklaration muss einen entsprechenden tatsächlichen Parameter in der OPEN-Anweisung haben. Die Datentypen der Parameter entsprechen den Datentypen für skalare Variablen. Sie weisen ihnen jedoch keine Größe zu. Die Parameternamen dienen als Referenz im Abfrageausdruck des Cursors.

Für die Syntax gilt:

<i>cursor_name</i>	PL/SQL-ID für den deklarierten Cursor
<i>parameter_name</i>	Name eines Parameters
<i>datatype</i>	Skalarer Datentyp des Parameters
<i>select_statement</i>	SELECT-Anweisung ohne INTO-Klausel

Die Parameternotation bietet keine weitere Funktionalität, sondern dient lediglich dazu, Eingabewerte auf einfache und übersichtliche Weise anzugeben. Dies ist speziell dann nützlich, wenn Sie wiederholt denselben Cursor referenzieren.

Cursor mit Parametern

```
DECLARE
  CURSOR  c_emp_cursor (deptno NUMBER) IS
    SELECT employee_id, last_name
    FROM   employees
    WHERE  department_id = deptno;
    ...
BEGIN
  OPEN c_emp_cursor (10);
  ...
  CLOSE c_emp_cursor;
  OPEN c_emp_cursor (20);
  ...
```

```
anonymous block completed
200 Whalen
201 Hartstein
202 Fay
```

ORACLE

Copyright © 2014, Oracle und/oder verbundene Unternehmen. All rights reserved. Alle Rechte vorbehalten.

Die Datentypen der Parameter entsprechen den Datentypen für skalare Variablen. Sie weisen ihnen jedoch keine Größe zu. Die Parameternamen dienen in der Abfrage des Cursors als Referenz. Im folgenden Beispiel wird ein Cursor deklariert und mit einem Parameter definiert:

```
DECLARE
  CURSOR c_emp_cursor(deptno NUMBER) IS SELECT ...
```

Die folgenden Anweisungen öffnen den Cursor und geben unterschiedliche aktive Mengen zurück:

```
OPEN c_emp_cursor(10);
OPEN c_emp_cursor(20);
```

Sie können Parameter an den in einer Cursor FOR-Schleife verwendeten Cursor übergeben:

```
DECLARE
  CURSOR c_emp_cursor(p_deptno NUMBER, p_job VARCHAR2) IS
    SELECT ...
BEGIN
  FOR emp_record IN c_emp_cursor(10, 'Sales') LOOP ...
```

Agenda

- Explizite Cursor – Definition
- Explizite Cursor verwenden
- Cursor mit Parametern
- Zeilen sperren und die aktuelle Zeile referenzieren

ORACLE

Copyright © 2014, Oracle und/oder verbundene Unternehmen. All rights reserved. Alle Rechte vorbehalten.

FOR UPDATE-Klauseln

Syntax:

```
SELECT ...
  FROM ...
FOR UPDATE [OF column_reference] [NOWAIT | WAIT n];
```

- Mit expliziten Sperren anderen Sessions während einer Transaktion den Zugriff verweigern
- Zeilen sperren, *bevor* sie aktualisiert oder gelöscht werden

ORACLE

Copyright © 2014, Oracle und/oder verbundene Unternehmen. All rights reserved. Alle Rechte vorbehalten.

Wenn mehrere Sessions auf eine Datenbank zugreifen, kann es vorkommen, dass die Zeilen einer bestimmten Tabelle aktualisiert werden, nachdem Sie den Cursor geöffnet haben. Sie sehen die aktualisierten Daten erst, wenn Sie den Cursor erneut öffnen. Es ist daher ratsam, die Zeilen zu sperren, bevor Sie sie aktualisieren oder löschen. Zu diesem Zweck können Sie in der Cursor-abfrage die Klausel FOR UPDATE verwenden.

Für die Syntax gilt:

<i>column_reference</i>	Spalte in der abgefragten Tabelle (gegebenenfalls auch eine Liste von Spalten)
NOWAIT	Gibt einen Oracle-Serverfehler zurück, wenn die Zeilen durch eine andere Session gesperrt sind

Die Klausel FOR UPDATE ist die letzte Klausel in einer SELECT-Anweisung, die sogar nach ORDER BY (sofern vorhanden) eingefügt wird. Wenn Sie mehrere Tabellen abfragen möchten, können Sie die Zeilensperrung mit der Klausel FOR UPDATE auf bestimmte Tabellen beschränken. FOR UPDATE OF *col_name(s)* sperrt Zeilen nur in Tabellen, die *col_name(s)* enthalten.

Die Anweisung `SELECT . . . FOR UPDATE` identifiziert die zu aktualisierenden oder zu löschen Zeilen und sperrt die einzelnen Zeilen dann in der Ergebnismenge. Dies ist nützlich, wenn Sie eine Aktualisierung auf Basis der in einer Zeile vorhandenen Werte durchführen möchten. Stellen Sie in diesem Fall sicher, dass die Zeile vor der Aktualisierung nicht durch eine andere Session geändert wird.

Das optionale Schlüsselwort `NOWAIT` weist den Oracle-Server an, nicht zu warten, wenn angeforderte Zeilen durch einen anderen Benutzer gesperrt wurden. Das Programm kann somit sofort andere Aufgaben ausführen, bevor es erneut versucht, die Zeilen zu sperren. Wenn Sie das Schlüsselwort `NOWAIT` weglassen, wartet der Oracle-Server, bis die Zeilen verfügbar sind.

Beispiel:

```
DECLARE
  CURSOR c_emp_cursor IS
    SELECT employee_id, last_name, FROM employees
    WHERE department_id = 80 FOR UPDATE OF salary NOWAIT;
  ...

```

Wenn der Oracle-Server die für den Vorgang `SELECT FOR UPDATE` erforderlichen Zeilensperren nicht zuteilen kann, wartet er auf unbestimmte Zeit. Vermeiden Sie dies mit dem Schlüsselwort `NOWAIT`. Wenn die Zeilen durch eine andere Session gesperrt wurden und Sie `NOWAIT` angegeben haben, wird beim Öffnen des Cursors ein Fehler ausgegeben. Sie können versuchen, den Cursor später zu öffnen. Bei Verwendung von `WAIT` anstelle von `NOWAIT` geben Sie die Anzahl der Sekunden an, die der Server warten soll, bevor er ermittelt, ob die Sperren der Zeilen aufgehoben sind. Falls die Zeilen nach n Sekunden weiterhin gesperrt sind, wird ein Fehler zurückgegeben.

Die Klausel `FOR UPDATE OF` muss sich nicht auf eine Spalte beziehen. Geben Sie jedoch an, welche Tabellen zu sperren sind, wenn die Anweisung ein Join mehrerer Tabellen ist.

WHERE CURRENT OF-Klauseln

Syntax:

```
WHERE CURRENT OF cursor ;
```

- Aktuelle Zeile mithilfe von Cursorn aktualisieren oder löschen
- Um die Zeilen zuerst zu sperren, FOR UPDATE-Klausel in die Cursorabfrage aufnehmen
- Mit der Klausel WHERE CURRENT OF die aktuelle Zeile eines expliziten Cursors referenzieren

```
UPDATE employees  
SET salary = ...  
WHERE CURRENT OF c_emp_cursor;
```

ORACLE

Copyright © 2014, Oracle und/oder verbundene Unternehmen. All rights reserved. Alle Rechte vorbehalten.

Um auf die aktuelle Zeile eines expliziten Cursors zu verweisen, verwenden Sie die Klausel WHERE CURRENT OF in Verbindung mit der FOR UPDATE-Klausel. Die Klausel WHERE CURRENT OF wird in den UPDATE- oder DELETE-Anweisungen verwendet, während Sie die FOR UPDATE-Klausel in der Cursordeklaration angeben. Indem Sie die Klauseln kombinieren, können Sie die aktuelle Zeile in der entsprechenden Datenbanktabelle aktualisieren und löschen. Damit haben Sie die Möglichkeit, UPDATE und DELETE-Vorgänge auf die aktuelle Zeile anzuwenden, ohne die ROWID explizit referenzieren zu müssen. Sie müssen die FOR UPDATE-Klausel in die Cursorabfrage aufnehmen, sodass die Zeilen beim OPEN-Vorgang gesperrt sind.

Für die Syntax gilt:

cursor Name eines deklarierten Cursors (der mit der FOR UPDATE-Klausel deklariert worden sein muss)

Quiz

Funktionen expliziter Cursor ermöglichen Programmierern die manuelle Kontrolle über explizite Cursor im PL/SQL-Block.

- a. Richtig
- b. Falsch



Copyright © 2014, Oracle und/oder verbundene Unternehmen. All rights reserved. Alle Rechte vorbehalten.

Richtige Antwort: a

Zusammenfassung

In dieser Lektion haben Sie Folgendes gelernt:

- Cursortypen unterscheiden:
 - Implizite Cursor werden für alle DML-Anweisungen und Single Row-Abfragen verwendet.
 - Explizite Cursor werden für Abfragen verwendet, die keine, eine oder mehrere Zeilen zurückgeben.
- Explizite Cursor erstellen und verwenden
- Mehrere Zeilen in Cursorn mit einfachen und Cursor FOR-Schleifen bearbeiten
- Den Cursorstatus mithilfe von Cursorattributen auswerten
- Aktuell gelesene Zeile mit den Klauseln FOR UPDATE und WHERE CURRENT OF aktualisieren oder löschen



Copyright © 2014, Oracle und/oder verbundene Unternehmen. All rights reserved. Alle Rechte vorbehalten.

Um SQL-Anweisungen auszuführen und Verarbeitungsinformationen zu speichern, verwendet der Oracle-Server Arbeitsbereiche. Mit einem PL/SQL-Konstrukt, das als *Cursor* bezeichnet wird, können Sie einen Arbeitsbereich benennen und auf die darin gespeicherten Informationen zugreifen. Es gibt zwei Typen von Cursorn: implizite und explizite Cursor. PL/SQL deklariert einen Cursor implizit für alle SQL-Anweisungen zur Datenmanipulation, einschließlich Abfragen, die nur eine Zeile zurückgeben. Für Abfragen, die mehrere Zeilen zurückgeben, müssen Sie einen Cursor explizit deklarieren, um die Zeilen einzeln zu verarbeiten.

Jeder explizite Cursor und jede Cursorvariable hat vier Attribute: %FOUND, %ISOPEN, %NOTFOUND und %ROWCOUNT. Wenn Sie die Attribute an den Namen der Cursorvariablen anhängen, geben sie nützliche Informationen über die Ausführung einer SQL-Anweisung zurück. Sie können Cursorattribute in prozeduralen Anweisungen, aber nicht in SQL-Anweisungen verwenden.

Bearbeiten Sie die vom Cursor gelesenen Zeilen mithilfe einfacher Schleifen oder Cursor FOR-Schleifen. Wenn Sie einfache Schleifen verwenden, müssen Sie den Cursor öffnen, die Zeilen lesen und den Cursor schließen. Cursor FOR-Schleifen führen diese Vorgänge implizit durch. Zeilen, die Sie aktualisieren oder löschen, sollten Sie mit der Klausel FOR UPDATE sperren. Sie stellen damit sicher, dass die von Ihnen verwendeten Daten nicht durch eine andere Session aktualisiert werden, nachdem Sie den Cursor geöffnet haben. Um auf die aktuell vom Cursor gelesene Zeile zu verweisen, verwenden Sie die Klausel WHERE CURRENT OF in Verbindung mit der FOR UPDATE-Klausel.

Übungen zu Lektion 8 – Übersicht

Diese Übungen behandeln folgende Themen:

- Explizite Cursor zum Abfragen von Tabellenzeilen deklarieren und verwenden
- Cursor FOR-Schleife verwenden
- Cursorstatus mit Hilfe von Cursorattributen testen
- Cursor mit Parametern deklarieren und verwenden
- Klauseln FOR UPDATE und WHERE CURRENT OF verwenden

ORACLE

Copyright © 2014, Oracle und/oder verbundene Unternehmen. All rights reserved. Alle Rechte vorbehalten.

In diesen Übungen verarbeiten Sie anhand des Gelernten eine Reihe von Zeilen einer Tabelle und füllen eine andere Tabelle mithilfe einer Cursor FOR-Schleife mit den Ergebnissen. Außerdem erstellen Sie einen Cursor mit Parametern.

9

Exceptions behandeln

ORACLE

Copyright © 2014, Oracle und/oder verbundene Unternehmen. All rights reserved. Alle Rechte vorbehalten.

Ziele

Nach Ablauf dieser Lektion haben Sie folgende Ziele erreicht:

- PL/SQL-Exceptions definieren
- Nicht behandelte Exceptions erkennen
- Verschiedene Typen von PL/SQL-Exception Handlern auflisten und verwenden
- Unerwartete Fehler abfangen
- Auswirkungen der Exception-Propagierung in verschachtelten Blöcken beschreiben
- PL/SQL-Exception-Meldungen anpassen



Copyright © 2014, Oracle und/oder verbundene Unternehmen. All rights reserved. Alle Rechte vorbehalten.

Sie haben gelernt, wie Sie PL/SQL-Blöcke mit einem deklarativen und einem ausführbaren Bereich erstellen. Sie schreiben den gesamten auszuführenden SQL- und PL/SQL-Code in den ausführbaren Block.

Bisher wurde davon ausgegangen, dass der Code zufriedenstellend funktioniert, wenn Sie Fehler zur Kompilierungszeit vermeiden. Der Code kann jedoch zur Laufzeit unvorhergesehene Fehler verursachen. In dieser Lektion wird beschrieben, wie Sie diese Fehler im PL/SQL-Block beheben.

Agenda

- PL/SQL-Exceptions kennenlernen
- Exceptions abfangen



Copyright © 2014, Oracle und/oder verbundene Unternehmen. All rights reserved. Alle Rechte vorbehalten.

Exceptions – Definition

```
DECLARE
    v_lname VARCHAR2(15);
BEGIN
    SELECT last_name INTO v_lname
    FROM employees
    WHERE first_name='John';
    DBMS_OUTPUT.PUT_LINE ('John''s last name is : ' ||v_lname);
END;
```



```
Script Output X | Task completed in 0.019 seconds
Error starting at line 3 in command:
DECLARE
    v_lname VARCHAR2(15);
BEGIN
    SELECT last_name INTO v_lname FROM employees WHERE
        first_name='John';
    DBMS_OUTPUT.PUT_LINE ('John''s last name is : ' ||v_lname);
END;
Error report:
ORA-01422: exact fetch returns more than requested number of rows
ORA-06512: at line 4
01422. 00000 - "exact fetch returns more than requested number of rows"
*Cause: The number specified in exact fetch is less than the rows returned.
*Action: Rewrite the query or change number of rows requested
```

ORACLE

Copyright © 2014, Oracle und/oder verbundene Unternehmen. All rights reserved. Alle Rechte vorbehalten.

Sehen Sie sich das Beispiel auf der Folie an. Der Code ist frei von Syntaxfehlern. Der anonyme Block müsste also von Ihnen erfolgreich ausgeführt werden können. Die SELECT-Anweisung im Block soll den Nachnamen von John abrufen.

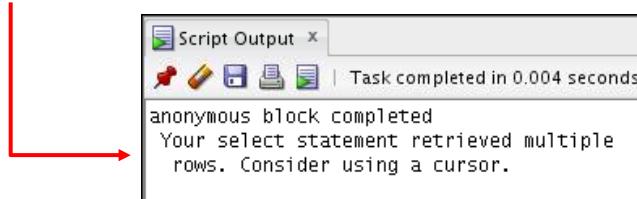
Wenn Sie den Code ausführen, wird jedoch der folgende Fehlerbericht angezeigt:

```
Error report:
ORA-01422: exact fetch returns more than requested number of rows
ORA-06512: at line 4
01422. 00000 - "exact fetch returns more than requested number of rows"
*Cause: The number specified in exact fetch is less than the rows returned.
*Action: Rewrite the query or change number of rows requested
```

Der Code funktioniert nicht erwartungsgemäß. Sie haben erwartet, dass die SELECT-Anweisung nur eine Zeile abruft, sie hat jedoch mehrere Zeilen abgerufen. Diese zur Laufzeit auftretenden Fehler werden als *Exceptions* bezeichnet. Exceptions führen dazu, dass der PL/SQL-Block beendet wird. Sie können Exceptions dieser Art im PL/SQL-Block behandeln.

Exceptions behandeln – Beispiel

```
DECLARE
    v_lname VARCHAR2(15);
BEGIN
    SELECT last_name INTO v_lname
    FROM employees
    WHERE first_name='John';
    DBMS_OUTPUT.PUT_LINE ('John''s last name is :' ||v_lname);
EXCEPTION
    WHEN TOO_MANY_ROWS THEN
        DBMS_OUTPUT.PUT_LINE (' Your select statement retrieved
multiple rows. Consider using a cursor.');
END;
/
```



ORACLE

Copyright © 2014, Oracle und/oder verbundene Unternehmen. All rights reserved. Alle Rechte vorbehalten.

Sie haben bereits gelernt, wie Sie PL/SQL-Blöcke erstellen, die einen deklarativen Bereich (beginnt mit dem Schlüsselwort `DECLARE`) und einen ausführbaren Bereich (beginnt und endet mit den Schlüsselwörtern `BEGIN` und `END`) umfassen.

Zur Exception-Behandlung fügen Sie einen weiteren optionalen Bereich hinzu, den sogenannten *Exception-Bereich*.

- Er beginnt mit dem Schlüsselwort `EXCEPTION`.
- Wenn er vorhanden ist, muss dies der letzte Bereich in einem PL/SQL-Block sein.

Beispiel

Im Beispiel auf der Folie wurde der Code der vorherigen Folie umgeschrieben, um die auftretene Exception zu behandeln. Auch die Ausgabe des Codes wird auf der Folie gezeigt.

Indem Sie den Bereich `EXCEPTION` in den Code aufnehmen, wird das PL/SQL-Programm nicht abrupt beendet. Bei Auslösung der Exception übernimmt der Exception-Bereich die Kontrolle, und alle Anweisungen im Exception-Bereich werden ausgeführt. Die Beendigung des PL/SQL-Blockes erfolgt auf normale Weise.

Exceptions mit PL/SQL

- Exceptions sind PL/SQL-Fehler, die bei der Ausführung von Programmen ausgelöst werden.
- Exceptions können wie folgt ausgelöst werden:
 - Implizit durch den Oracle-Server
 - Explizit durch das Programm
- Die Exception-Behandlung erfolgt:
 - durch Auffangen mit einem Handler
 - durch Propagierung an die aufrufende Umgebung

ORACLE

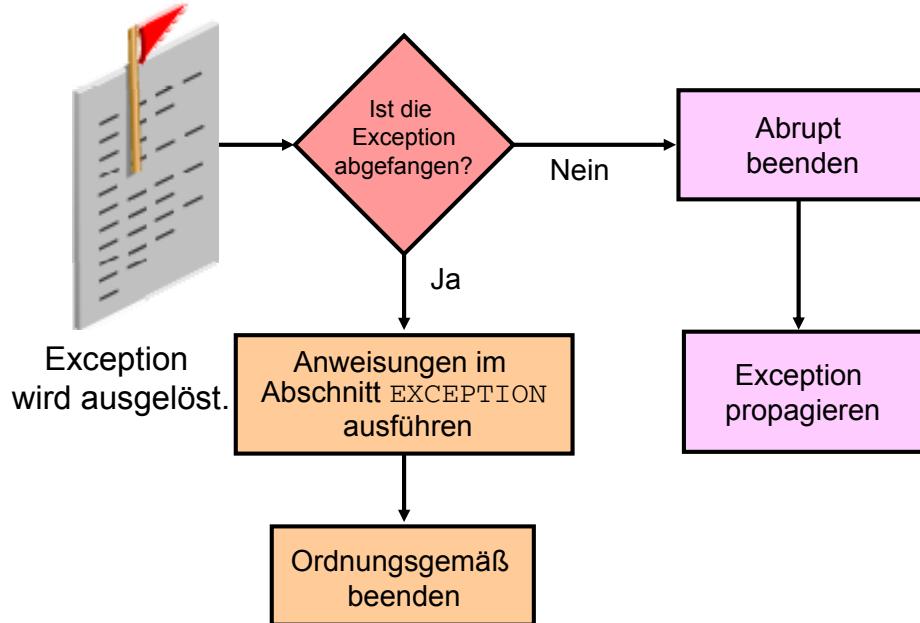
Copyright © 2014, Oracle und/oder verbundene Unternehmen. All rights reserved. Alle Rechte vorbehalten.

Eine Exception ist ein Fehler in PL/SQL, der bei der Ausführung eines Blockes ausgelöst wird. Blöcke werden immer beendet, wenn PL/SQL eine Exception auslöst. Wenn Sie jedoch einen Exception Handler angeben, werden vor Beendigung des Blockes noch abschließende Aktionen ausgeführt.

Zwei Methoden zum Auslösen von Exceptions

- Ein Oracle-Fehler tritt auf, und die damit verknüpfte Exception wird automatisch ausgelöst. Beispiel: Der Fehler ORA-01403 tritt auf, weil mit einer SELECT-Anweisung keine Zeilen aus der Datenbank abgerufen wurden. In diesem Fall löst PL/SQL die Exception NO_DATA_FOUND aus. Die Fehler werden in vordefinierte Exceptions konvertiert.
- Je nach Geschäftsfunktionalität des Programms müssen Sie eine Exception eventuell explizit auslösen. Setzen Sie zu diesem Zweck im Block eine RAISE-Anweisung ab. Die ausgelöste Exception kann benutzerdefiniert oder vordefiniert sein. Es gibt auch einige nicht vordefinierte Oracle-Fehler. Zu diesen Fehlern gehören alle Oracle-Standardfehler, die nicht vordefiniert sind. Sie können Exceptions explizit deklarieren und den nicht vordefinierten Oracle-Fehlern zuordnen.

Exceptions behandeln



ORACLE

Copyright © 2014, Oracle und/oder verbundene Unternehmen. All rights reserved. Alle Rechte vorbehalten.

Exceptions abfangen

Um Exceptions abzufangen, nehmen Sie einen EXCEPTION-Bereich in das PL/SQL-Programm auf. Wenn die Exception im ausführbaren Bereich des Blockes ausgelöst wird, verzweigt die Verarbeitung auf den entsprechenden Exception Handler im Exception-Bereich des Blockes. Wenn PL/SQL die Exception erfolgreich behandelt, wird sie nicht an den umschließenden Block oder die aufrufende Umgebung propagiert. Der PL/SQL-Block wird erfolgreich beendet.

Exceptions propagieren

Wenn die Exception im ausführbaren Bereich des Blockes ausgelöst wird und kein entsprechender Exception Handler vorhanden ist, wird der PL/SQL-Block mit einem Fehler beendet, und die Exception wird an einen umschließenden Block oder die aufrufende Umgebung propagiert. Die aufrufende Umgebung kann eine beliebige Anwendung sein (beispielsweise SQL*Plus, das das PL/SQL-Programm auruft).

Exception-Typen

- Vordefinierte Oracle-Serverfehler
 - Nicht vordefinierte Oracle-Serverfehler
- } Implizit ausgelöst
-
- Benutzerdefinierte Fehler
- Explizit ausgelöst

ORACLE

Copyright © 2014, Oracle und/oder verbundene Unternehmen. All rights reserved. Alle Rechte vorbehalten.

Es gibt drei Typen von Exceptions:

Exception	Beschreibung	Anweisungen zur Behandlung
Vordefinierter Oracle-Serverfehler	Einer von ungefähr 20 Fehlern, die in PL/SQL-Code am häufigsten vorkommen	Sie müssen diese Exceptions nicht deklarieren. Sie werden vom Oracle-Server vordefiniert und implizit ausgelöst.
Nicht vordefinierter Oracle-Serverfehler	Alle sonstigen Standardfehler des Oracle-Servers	Diese Fehler müssen Sie im deklarativen Bereich deklarieren. Der Oracle-Server löst den Fehler implizit aus, und Sie können den Fehler im Exception Handler abfangen.
Benutzerdefinierter Fehler	Eine Bedingung, die der Entwickler als anormal definiert	Sie müssen diese im deklarativen Bereich deklarieren und explizit auslösen.

Hinweis: Einige Anwendungstools mit clientseitigem PL/SQL (zum Beispiel Oracle Developer Forms) enthalten eigene Exceptions.

Agenda

- PL/SQL-Exceptions kennenlernen
- Exceptions abfangen



Copyright © 2014, Oracle und/oder verbundene Unternehmen. All rights reserved. Alle Rechte vorbehalten.

Exceptions abfangen – Syntax

```
EXCEPTION
  WHEN exception1 [OR exception2 . . .] THEN
    statement1;
    statement2;
    . . .
  [WHEN exception3 [OR exception4 . . .] THEN
    statement1;
    statement2;
    . . .]
  [WHEN OTHERS THEN
    statement1;
    statement2;
    . . .]
```



Copyright © 2014, Oracle und/oder verbundene Unternehmen. All rights reserved. Alle Rechte vorbehalten.

Sie können Fehler abfangen, indem Sie einen entsprechenden Handler in den Bereich für die Exception-Behandlung des PL/SQL-Blockes aufnehmen. Jeder Handler enthält eine `WHEN`-Klausel, die einen Exception-Namen und eine Anweisungsfolge angibt. Die Anweisungsfolge wird ausgeführt, wenn die Exception ausgelöst wird.

Zur Behandlung bestimmter Exceptions können Sie eine beliebige Anzahl von Handlern in einen `EXCEPTION`-Bereich aufnehmen. Jede Exception kann jedoch nur einen Handler haben.

Die Syntax zum Auffangen von Exceptions umfasst die folgenden Elemente:

<code>exception</code>	Standardname einer vordefinierten Exception oder Name einer benutzerdefinierten Exception, die im deklarativen Bereich deklariert ist
<code>statement</code>	Eine oder mehrere PL/SQL- oder SQL-Anweisungen
<code>OTHERS</code>	Optionale Klausel für die Exception-Behandlung, die alle nicht explizit behandelten Exceptions abfängt

Exception Handler WHEN OTHERS

Wie bereits erwähnt, fängt der Bereich für die Exception-Behandlung nur angegebene Exceptions ab.

Exceptions, die nicht angegeben sind, werden mit dem Exception Handler `OTHERS` abgefangen. Diese Option fängt alle noch nicht behandelten Exceptions ab. Aus diesem Grund muss der Handler `OTHERS` der letzte definierte Handler sein (sofern Sie ihn verwenden).

Beispiel:

```
WHEN NO_DATA_FOUND THEN  
    statement1;  
    ...  
WHEN TOO_MANY_ROWS THEN  
    statement1;  
    ...  
WHEN OTHERS THEN  
    statement1;
```

Beispiel

Sehen Sie sich das vorangegangene Beispiel an. Wenn das Programm die Exception `NO_DATA_FOUND` auslöst, werden die Anweisungen im entsprechenden Handler ausgeführt. Wird die Exception `TOO_MANY_ROWS` ausgelöst, werden ebenfalls die Anweisungen im entsprechenden Handler ausgeführt. Wird jedoch eine andere Exception ausgelöst, werden die Anweisungen im Exception Handler `OTHERS` ausgeführt.

Der Handler `OTHERS` fängt alle noch nicht abgefangenen Exceptions ab. Einige Oracle-Tools haben eigene vordefinierte Exceptions, mit denen Sie Ereignisse in der Anwendung auslösen können. Der Handler `OTHERS` fängt auch diese Exceptions ab.

Exceptions abfangen – Richtlinien

- Das Schlüsselwort EXCEPTION steht am Anfang des Bereichs für die Exception-Behandlung.
- Es sind mehrere Exception Handler zulässig.
- Vor Verlassen des Blockes wird nur ein Handler verarbeitet.
- WHEN OTHERS ist die letzte Klausel.

ORACLE

Copyright © 2014, Oracle und/oder verbundene Unternehmen. All rights reserved. Alle Rechte vorbehalten.

- Sie beginnen den Bereich für die Exception-Behandlung des Blockes mit dem Schlüsselwort EXCEPTION.
- Definieren Sie für den Block mehrere Exception Handler mit jeweils eigenen Aktionen.
- Wenn eine Exception auftritt, verarbeitet PL/SQL vor dem Verlassen des Blockes nur einen Handler.
- Fügen Sie die Klausel OTHERS an letzter Stelle nach allen anderen Klauseln für die Exception-Behandlung ein.
- Es ist nur eine OTHERS-Klausel zulässig.
- In Zuweisungsanweisungen oder SQL-Anweisungen können keine Exceptions verwendet werden.

Vordefinierte Oracle-Serverfehler abfangen

- Vordefinierte Namen in der Exception-Behandlungsroutine referenzieren
- Beispiele für vordefinierte Exceptions:
 - NO_DATA_FOUND
 - TOO_MANY_ROWS
 - INVALID_CURSOR
 - ZERO_DIVIDE
 - DUP_VAL_ON_INDEX



Copyright © 2014, Oracle und/oder verbundene Unternehmen. All rights reserved. Alle Rechte vorbehalten.

Sie können einen vordefinierten Oracle-Serverfehler abfangen, indem Sie seinen vordefinierten Namen innerhalb der entsprechenden Exception-Behandlungsroutine referenzieren.

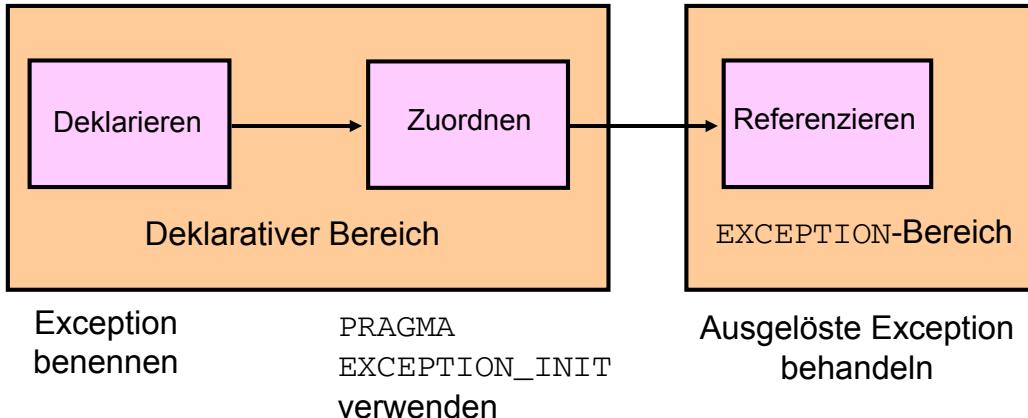
Eine vollständige Liste vordefinierter Exceptions finden Sie im Dokument *PL/SQL User's Guide and Reference*.

Hinweis: PL/SQL deklariert vordefinierte Exceptions im Package STANDARD.

Name der Exception	Nummer des Oracle-Serverfehlers	Beschreibung
ACCESS_INTO_NULL	ORA-06530	Es wurde versucht, den Attributen eines nicht initialisierten Objekts Werte zuzuweisen.
CASE_NOT_FOUND	ORA-06592	Keiner der Fälle in den WHEN-Klauseln einer CASE-Anweisung wurde ausgewählt, und es ist keine ELSE-Klausel vorhanden.
COLLECTION_IS_NULL	ORA-06531	Es wurde versucht, auf nicht initialisierte Nested Tables oder VARRAYS andere Collection-Methoden als EXISTS anzuwenden.
CURSOR_ALREADY_OPEN	ORA-06511	Es wurde versucht, einen bereits geöffneten Cursor zu öffnen.
DUP_VAL_ON_INDEX	ORA-00001	Es wurde versucht, einen Wert mehrfach einzufügen.
INVALID_CURSOR	ORA-01001	Ein ungültiger Cursorvorgang ist aufgetreten.
INVALID_NUMBER	ORA-01722	Die Konvertierung einer Zeichenfolge in eine Zahl war nicht erfolgreich.
LOGIN_DENIED	ORA-01017	Anmeldung beim Oracle-Server mit ungültigem Benutzernamen oder ungültigem Kennwort
NO_DATA_FOUND	ORA-01403	Keine Rückgabe von Daten in SELECT-Anweisung von Single-Row-Abfrage
NOT_LOGGED_ON	ORA-01012	Das PL/SQL-Programm hat einen Datenbankaufruf abgesetzt, ohne mit dem Oracle-Server verbunden zu sein.
PROGRAM_ERROR	ORA-06501	Internes PL/SQL-Problem
ROWTYPE_MISMATCH	ORA-06504	Die zur Zuweisung gehörenden Host- und PL/SQL-Cursorvariablen haben inkompatible Rückgabetypen.

Name der Exception	Nummer des Oracle-Serverfehlers	Beschreibung
STORAGE_ERROR	ORA-06500	Zu wenig Speicher für PL/SQL oder Speicher beschädigt
SUBSCRIPT_BEYOND_COUNT	ORA-06533	Eine Nested Table oder ein VARRAY-Element wurde mit einer Indexnummer referenziert, die größer als die Anzahl der Elemente in der Collection ist.
SUBSCRIPT_OUTSIDE_LIMIT	ORA-06532	Eine Nested Table oder ein VARRAY-Element wurde mithilfe einer Indexnummer referenziert, die außerhalb des gültigen Bereichs liegt (zum Beispiel -1).
SYS_INVALID_ROWID	ORA-01410	Die Konvertierung einer Zeichenfolge in eine universelle ROWID war nicht erfolgreich, da die Zeichenfolge keine gültige ROWID darstellt.
TIMEOUT_ON_RESOURCE	ORA-00051	Beim Warten des Oracle-Servers auf eine Ressource ist ein Timeout aufgetreten.
TOO_MANY_ROWS	ORA-01422	Bei einer SELECT-Anweisung für eine Single-Row-Abfrage wurden mehrere Zeilen zurückgegeben.
VALUE_ERROR	ORA-06502	Beim Konvertieren, beim Abschneiden, in der Arithmetik oder bei der Größen-einschränkung ist ein Fehler aufgetreten.
ZERO_DIVIDE	ORA-01476	Es wurde versucht, durch 0 zu teilen.

Nicht vordefinierte Oracle-Serverfehler abfangen



ORACLE

Copyright © 2014, Oracle und/oder verbundene Unternehmen. All rights reserved. Alle Rechte vorbehalten.

Nicht vordefinierte Exceptions ähneln vordefinierten Exceptions, sind allerdings nicht als PL/SQL-Exceptions im Oracle-Server definiert. Es handelt sich um Standardfehler von Oracle. Exceptions für Oracle-Standardfehler erstellen Sie mit der Funktion `PRAGMA EXCEPTION_INIT`. Diese Exceptions werden als nicht vordefinierte Exceptions bezeichnet.

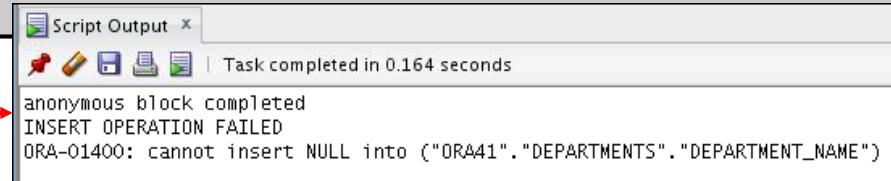
Sie können nicht vordefinierte Oracle-Serverfehler abfangen, indem Sie sie zunächst deklarieren. Die deklarierte Exception wird implizit ausgelöst. In PL/SQL weist `PRAGMA EXCEPTION_INIT` den Compiler an, einer Oracle-Fehlernummer einen Exception-Namen zuzuordnen. Auf diese Weise können Sie jede interne Exception anhand des Namens referenzieren und einen speziellen Handler dafür erstellen.

Hinweis: Mit dem Schlüsselwort `PRAGMA` (auch als *Pseudoanweisung* bezeichnet) geben Sie an, dass es sich um eine Compileranweisung handelt, die bei Ausführung des PL/SQL-Blockes nicht verarbeitet wird. Sie weist den PL/SQL-Compiler vielmehr an, alle Vorkommen des Exception-Namens innerhalb des Blockes als die zugeordnete Nummer für einen Oracle-Serverfehler zu interpretieren.

Nicht vordefinierte Fehler abfangen – Beispiel

Oracle-Serverfehler 01400 ("cannot insert NULL") abfangen:

```
DECLARE
    e_insert_excep EXCEPTION;          ← 1
    PRAGMA EXCEPTION_INIT(e_insert_excep, -01400); ← 2
BEGIN
    INSERT INTO departments
        (department_id, department_name) VALUES (280, NULL);
EXCEPTION
    WHEN e_insert_excep THEN          ← 3
        DBMS_OUTPUT.PUT_LINE('INSERT OPERATION FAILED');
        DBMS_OUTPUT.PUT_LINE(SQLERRM);
END;
/
```



ORACLE

Copyright © 2014, Oracle und/oder verbundene Unternehmen. All rights reserved. Alle Rechte vorbehalten.

Das Beispiel veranschaulicht die drei Schritte, die zum Abfangen eines nicht vordefinierten Fehlers auszuführen sind:

1. Deklarieren Sie den Namen der Exception im deklarativen Bereich mit folgender Syntax.

```
exception      EXCEPTION;
```

In dieser Syntax ist *exception* der Name der Exception.

2. Ordnen Sie die deklarierte Exception der Nummer des standardmäßigen Oracle-Serverfehlers zu. Verwenden Sie dazu die Funktion PRAGMA EXCEPTION_INIT. Syntax:

```
PRAGMA EXCEPTION_INIT(exception, error_number);
```

In dieser Syntax ist *exception* die zuvor deklarierte Exception und *error_number* die Nummer eines standardmäßigen Oracle-Serverfehlers.

3. Referenzieren Sie die deklarierte Exception innerhalb der entsprechenden Exception-Behandlungsroutine.

Beispiel

Im Beispiel auf der Folie wird versucht, den NULL-Wert für die Spalte *department_name* der Tabelle *departments* einzufügen. Der Vorgang ist jedoch nicht erfolgreich, da *department_name* eine NOT NULL-Spalte ist. Beachten Sie die folgende Zeile im Beispiel:

```
DBMS_OUTPUT.PUT_LINE(SQLERRM);
```

Mit der Funktion SQLERRM rufen Sie die Fehlermeldung ab. SQLERRM wird auf den folgenden Folien näher erläutert.

Exceptions abfangen – Funktionen

- SQLCODE: Gibt den numerischen Wert für den Fehlercode zurück
- SQLERRM: Gibt die der Fehlernummer zugeordnete Meldung zurück

ORACLE

Copyright © 2014, Oracle und/oder verbundene Unternehmen. All rights reserved. Alle Rechte vorbehalten.

Wenn eine Exception auftritt, können Sie den zugeordneten Fehlercode oder die Fehlermeldung mit zwei Funktionen identifizieren. Auf Grundlage der folgenden Werte für Code oder Meldung können Sie die auszuführenden Maßnahmen festlegen.

SQLCODE gibt die Oracle-Fehlernummer für interne Exceptions zurück. SQLERRM gibt die der Fehlernummer zugeordnete Meldung zurück.

Funktion	Beschreibung
SQLCODE	Gibt den numerischen Wert des Fehlercodes zurück. (Sie können diesen einer NUMBER-Variablen zuweisen.)
SQLERRM	Gibt die der Fehlernummer zugeordnete Meldung zurück

Werte von SQLCODE – Beispiele

Wert von SQLCODE	Beschreibung
0	Keine Exception aufgetreten
1	Benutzerdefinierte Exception
+100	Exception NO_DATA_FOUND
Negative Zahl	Nummer eines anderen Oracle-Serverfehlers

Exceptions abfangen – Funktionen

```
DECLARE
    error_code      NUMBER;
    error_message   VARCHAR2(255);
BEGIN
    ...
EXCEPTION
    ...
    WHEN OTHERS THEN
        ROLLBACK;
        error_code := SQLCODE ;
        error_message := SQLERRM ;
        INSERT INTO errors (e_user, e_date, error_code,
        error_message) VALUES (USER, SYSDATE, error_code,
        error_message);
END;
/
```

ORACLE

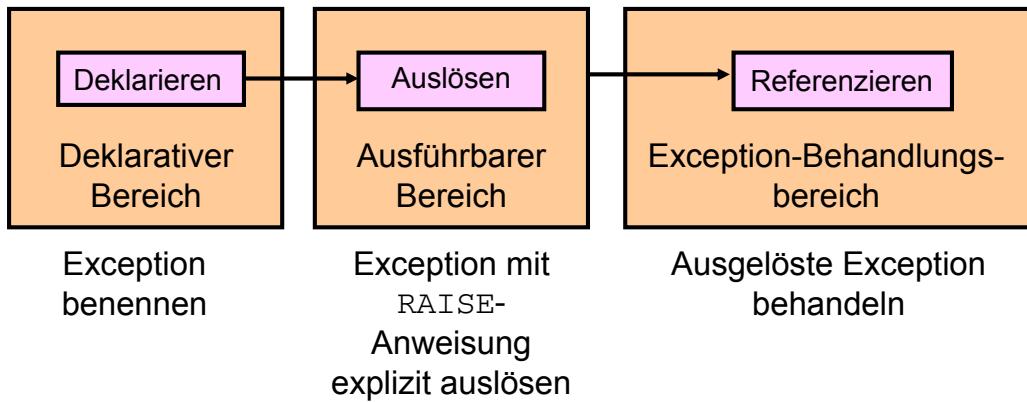
Copyright © 2014, Oracle und/oder verbundene Unternehmen. All rights reserved. Alle Rechte vorbehalten.

Wenn eine Exception im Exception Handler `WHEN OTHERS` abgefangen wird, können Sie die betreffenden Fehler mit einer Reihe generischer Funktionen ermitteln. Im Beispiel auf der Folie sind die Werte von `SQLCODE` und `SQLERRM` Variablen zugeordnet, die von einer SQL-Anweisung verwendet werden.

`SQLCODE` oder `SQLERRM` dürfen jedoch nicht direkt in einer SQL-Anweisung verwendet werden. Stattdessen müssen Sie die Werte lokalen Variablen zuordnen und diese dann in der SQL-Anweisung einsetzen (siehe folgendes Beispiel):

```
DECLARE
    err_num NUMBER;
    err_msg VARCHAR2(100);
BEGIN
    ...
EXCEPTION
    ...
    WHEN OTHERS THEN
        err_num := SQLCODE;
        err_msg := SUBSTR(SQLERRM, 1, 100);
        INSERT INTO errors VALUES (err_num, err_msg);
END;
/
```

Benutzerdefinierte Exceptions abfangen



ORACLE

Copyright © 2014, Oracle und/oder verbundene Unternehmen. All rights reserved. Alle Rechte vorbehalten.

Mit PL/SQL können Sie je nach Anforderungen Ihrer Anwendung eigene Exceptions definieren. Sie können beispielsweise Benutzer auffordern, eine Abteilungsnummer einzugeben. Definieren Sie Exceptions, wenn Fehlerbedingungen in den Eingabedaten behandelt werden sollen. Prüfen Sie, ob die Abteilungsnummer vorhanden ist. Ist dies nicht der Fall, müssen Sie möglicherweise die benutzerdefinierte Exception auslösen.

PL/SQL-Exceptions müssen:

- im deklarativen Bereich eines PL/SQL-Blockes deklariert sein
- explizit mit RAISE-Anweisungen ausgelöst werden
- im EXCEPTION-Bereich behandelt werden

Benutzerdefinierte Exceptions abfangen

```
DECLARE
    v_deptno NUMBER := 500;
    v_name VARCHAR2(20) := 'Testing';
    e_invalid_department EXCEPTION; ← 1
BEGIN
    UPDATE departments
    SET department_name = v_name
    WHERE department_id = v_deptno;
    IF SQL%NOTFOUND THEN
        RAISE e_invalid_department; ← 2
    END IF;
    COMMIT;
EXCEPTION
    WHEN e_invalid_department THEN
        DBMS_OUTPUT.PUT_LINE('No such department id.');
END;
/
```

The Oracle output window shows:

Script Output X
Task comp
anonymous block completed
No such department id.

ORACLE

Copyright © 2014, Oracle und/oder verbundene Unternehmen. All rights reserved. Alle Rechte vorbehalten.

Sie fangen benutzerdefinierte Exceptions ab, indem Sie sie deklarieren und explizit auslösen.

1. Deklarieren Sie den Namen der benutzerdefinierten Exception im deklarativen Bereich.

Syntax:

```
exception EXCEPTION;
```

In dieser Syntax ist *exception* der Name der Exception.

2. Lösen Sie die Exception im ausführbaren Bereich mithilfe der RAISE-Anweisung explizit aus.

Syntax:

```
RAISE exception;
```

In dieser Syntax ist *exception* die zuvor deklarierte Exception.

3. Referenzieren Sie die deklarierte Exception innerhalb der entsprechenden Exception- Behandlungsroutine.

Beispiel

Der auf der Folie gezeigte Block aktualisiert den Namen (*department_name*) einer Abteilung. Der Benutzer gibt die Abteilungsnummer und den neuen Namen an. Falls die angegebene Abteilungsnummer nicht vorhanden ist, werden in der Tabelle *departments* keine Zeilen aktualisiert. Es wird eine Exception ausgelöst, und der Benutzer erhält eine Meldung mit der Information, dass eine ungültige Abteilungsnummer eingegeben wurde.

Hinweis: Um dieselbe Exception erneut auszulösen und an die aufrufende Umgebung zurück zu propagieren, verwenden Sie die RAISE-Anweisung alleine in einem Exception Handler.

Exceptions in Unterblöcken propagieren

Unterblöcke können eine Exception behandeln oder an den umschließenden Block übergeben.

```
DECLARE
    . . .
    e_no_rows      exception;
    e_integrity     exception;
    PRAGMA EXCEPTION_INIT (e_integrity, -2292);
BEGIN
    FOR c_record IN emp_cursor LOOP
        BEGIN
            SELECT ... ;
            UPDATE ... ;
            IF SQL%NOTFOUND THEN
                RAISE e_no_rows;
            END IF;
        END;
    END LOOP;
EXCEPTION
    WHEN e_integrity THEN ...
    WHEN e_no_rows THEN ...
END;
/
```

ORACLE

Copyright © 2014, Oracle und/oder verbundene Unternehmen. All rights reserved. Alle Rechte vorbehalten.

Wenn ein Unterblock eine Exception behandelt, endet er normal. Der umschließende Block übernimmt sofort nach der im Unterblock ausgeführten END-Anweisung die Kontrolle.

Löst jedoch eine PL/SQL-Anweisung eine Exception aus und der aktuelle Block enthält keinen Handler für die Exception, wird die Exception so lange an nachfolgende umschließende Blöcke propagiert, bis ein Handler gefunden wird. Wenn keiner dieser Blöcke die Exception behandelt, wird in der Hostumgebung eine unbehandelte Exception ausgegeben.

Wird die Exception an einen umschließenden Block propagiert, werden die verbleibenden ausführbaren Aktionen in diesem Block umgangen.

Ein Vorteil dieses Verhaltens ist, dass Sie Anweisungen, die eine eigene exklusive Fehlerbehandlung benötigen, in einen eigenen Block einfügen können, während eher allgemeine Fehlerbehandlungen im umschließenden Block erfolgen.

Beachten Sie im Beispiel, dass die Exceptions (`no_rows` und `integrity`) im äußeren Block deklariert sind. Wenn die Exception `no_rows` ausgelöst wird, sucht PL/SQL im inneren Block nach der Exception, die im Unterblock behandelt werden soll. Da die Exception nicht im Unterblock behandelt wird, wird sie an den äußeren Block propagiert, wo PL/SQL nach dem Handler sucht.

RAISE-Anweisungen

- Stoppt die normale Ausführung eines PL/SQL-Blockes oder Unterprogramms und übergibt die Kontrolle an einen Exception Handler
- Löst vordefinierte Exceptions oder benutzerdefinierte Exceptions explizit aus
- Syntax:

```
RAISE exception_name ;
```

ORACLE

Copyright © 2014, Oracle und/oder verbundene Unternehmen. All rights reserved. Alle Rechte vorbehalten.

In der Syntax ist `exception_name` der Name einer vordefinierten oder benutzerdefinierten Exception. `exception_name` ist nur in einem Exception Handler optional, in dem die aktuelle Exception der Standard ist. Außerhalb eines Exception Handlers muss der Name der Exception angegeben werden.

RAISE_APPLICATION_ERROR-Prozeduren

Syntax:

```
raise_application_error (error_number,  
                      message[, {TRUE | FALSE}]);
```

- Mit dieser Prozedur setzen Sie benutzerdefinierte Fehlermeldungen aus Stored Subprograms ab.
- Sie können der Anwendung Fehler melden und die Rückgabe nicht behandelter Exceptions vermeiden.



Copyright © 2014, Oracle und/oder verbundene Unternehmen. All rights reserved. Alle Rechte vorbehalten.

Mit der Prozedur RAISE_APPLICATION_ERROR übermitteln Sie eine vordefinierte Exception interaktiv, indem Sie Fehlercode und Fehlermeldung zurückgeben, die nicht standardmäßig sind. Mit der Prozedur RAISE_APPLICATION_ERROR können Sie den Anwendungen Fehler melden und die Rückgabe von nicht behandelten Exceptions vermeiden.

Für die Syntax gilt:

error_number Benutzerdefinierte Zahl für die Exception zwischen –20.000 und –20.999

message Benutzerdefinierte Meldung für die Exception (Zeichenfolge mit bis zu 2.048 Byte)

TRUE | FALSE Optionaler boolescher Parameter (Wenn TRUE, wird der Fehler im Stack vorheriger Fehler platziert. Wenn FALSE (Standard), ersetzt der Fehler alle vorhergehenden Fehler.)

RAISE_APPLICATION_ERROR-Prozeduren

- Werden an zwei Stellen verwendet:
 - Im ausführbaren Bereich
 - Im Exception-Bereich
- Gibt Fehlerbedingungen in einer Weise an den Benutzer zurück, die mit anderen Oracle-Serverfehlern übereinstimmt

ORACLE

Copyright © 2014, Oracle und/oder verbundene Unternehmen. All rights reserved. Alle Rechte vorbehalten.

Sie können die Prozedur `RAISE_APPLICATION_ERROR` im ausführbaren Bereich und/oder im Exception-Bereich eines PL/SQL-Programms verwenden. Der zurückgegebene Fehler stimmt mit der Art und Weise überein, wie der Oracle-Server vordefinierte, nicht definierte oder benutzerdefinierte Fehler generiert. Der Benutzer sieht die Fehlernummer und die Fehlermeldung.

RAISE_APPLICATION_ERROR-Prozeduren

Ausführbarer Bereich:

```
BEGIN
...
    DELETE FROM employees
        WHERE manager_id = v_mgr;
    IF SQL%NOTFOUND THEN
        RAISE_APPLICATION_ERROR(-20202,
            'This is not a valid manager');
    END IF;
    ...

```

Exception-Bereich:

```
...
EXCEPTION
    WHEN NO_DATA_FOUND THEN
        RAISE_APPLICATION_ERROR (-20201,
            'Manager is not a valid employee.');
END;
/
```

ORACLE

Copyright © 2014, Oracle und/oder verbundene Unternehmen. All rights reserved. Alle Rechte vorbehalten.

Die Folie zeigt, dass die Prozedur RAISE_APPLICATION_ERROR im ausführbaren Bereich und Exception-Bereich eines PL/SQL-Programms verwendet werden kann.

Dies ist ein weiteres Beispiel für die Verwendung der Prozedur RAISE_APPLICATION_ERROR:

```
DECLARE
    e_name EXCEPTION;
BEGIN
    ...
    DELETE FROM employees
    WHERE last_name = 'Higgins';
    IF SQL%NOTFOUND THEN RAISE e_name;
    END IF;
EXCEPTION
    WHEN e_name THEN
        RAISE_APPLICATION_ERROR (-20999, 'This is not a valid last
name');    ...
END;
/
```

Quiz

Sie können Fehler abfangen, indem Sie in den Bereich für die Exception-Behandlung des PL/SQL-Blockes einen entsprechenden Handler aufnehmen.

- a. Richtig
- b. Falsch



Copyright © 2014, Oracle und/oder verbundene Unternehmen. All rights reserved. Alle Rechte vorbehalten.

Richtige Antwort: a

Sie können Fehler abfangen, indem Sie einen entsprechenden Handler in den Bereich für die Exception-Behandlung des PL/SQL-Blockes aufnehmen. Jeder Handler enthält eine WHEN-Klausel, die einen Exception-Namen und eine Anweisungsfolge angibt. Die Anweisungsfolge wird ausgeführt, wenn die Exception ausgelöst wird. Zur Behandlung bestimmter Exceptions können Sie eine beliebige Anzahl von Handlern in einen EXCEPTION-Bereich aufnehmen. Jede Exception kann jedoch nur einen Handler haben.

Zusammenfassung

In dieser Lektion haben Sie Folgendes gelernt:

- PL/SQL-Exceptions definieren
- PL/SQL-Blöcken einen EXCEPTION-Bereich hinzufügen, um Exceptions zur Laufzeit zu behandeln
- Verschiedene Exception-Typen behandeln:
 - Vordefinierte Exceptions
 - Nicht vordefinierte Exceptions
 - Benutzerdefinierte Exceptions
- Exceptions in verschachtelten Blöcken propagieren und Anwendungen aufrufen

ORACLE

Copyright © 2014, Oracle und/oder verbundene Unternehmen. All rights reserved. Alle Rechte vorbehalten.

In dieser Lektion haben Sie gelernt, verschiedene Typen von Exceptions zu behandeln. In PL/SQL werden Warnungen oder Fehlerbedingungen zur Laufzeit als Exceptions bezeichnet. Vordefinierte Exceptions sind vom Oracle-Server definierte Fehlerbedingungen. Nicht vordefinierte Exceptions können beliebige standardmäßige Oracle-Serverfehler sein. Benutzerdefinierte Exceptions sind anwendungsspezifische Exceptions. Mit der Funktion PRAGMA EXCEPTION_INIT können Sie einem Oracle-Serverfehler den Namen einer deklarierten Exception zuordnen.

Im deklarativen Bereich von PL/SQL-Blöcken lassen sich eigene Exceptions definieren. Um beispielsweise überzogene Bankkonten zu kennzeichnen, können Sie eine Exception namens INSUFFICIENT_FUNDS definieren.

Bei einem Fehler wird eine Exception ausgelöst. Die normale Ausführung stoppt und übergibt die Kontrolle an den Bereich zur Exception-Behandlung des PL/SQL-Blockes. Interne Exceptions werden implizit (automatisch) vom Laufzeitsystem ausgelöst, während Sie benutzerdefinierte Exceptions explizit auslösen müssen. Um ausgelöste Exceptions zu behandeln, erstellen Sie separate Routinen, die als Exception Handler bezeichnet werden.

Übungen zu Lektion 9 – Übersicht

Diese Übungen behandeln folgende Themen:

- Benutzerdefinierte Exceptions erstellen und aufrufen
- Benannte Oracle-Server-Exceptions behandeln



Copyright © 2014, Oracle und/oder verbundene Unternehmen. All rights reserved. Alle Rechte vorbehalten.

In diesen Übungen erstellen Sie Exception Handler für eine vordefinierte Exception und für eine standardmäßige Oracle-Server-Exception.

Stored Procedures und Stored Functions – Einführung

ORACLE

Copyright © 2014, Oracle und/oder verbundene Unternehmen. All rights reserved. Alle Rechte vorbehalten.

Ziele

Nach Ablauf dieser Lektion haben Sie folgende Ziele erreicht:

- Zwischen anonymen Blöcken und Unterprogrammen unterscheiden
- Einfache Prozeduren erstellen und mit anonymen Blöcken aufrufen
- Einfache Funktionen erstellen
- Einfache Funktionen erstellen, die Parameter annehmen
- Zwischen Prozeduren und Funktionen unterscheiden



Copyright © 2014, Oracle und/oder verbundene Unternehmen. All rights reserved. Alle Rechte vorbehalten.

Anonyme Blöcke haben Sie bereits kennengelernt. In dieser Lektion erhalten Sie eine Einführung in benannte Blöcke, die auch als *Unterprogramme* bezeichnet werden. Prozeduren und Funktionen sind PL/SQL-Unterprogramme. In dieser Lektion lernen Sie den Unterschied zwischen anonymen Blöcken und Unterprogrammen kennen.

Agenda

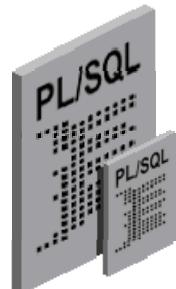
- Prozeduren und Funktionen – Einführung
- Prozeduren – Vorschau
- Funktionen – Vorschau

ORACLE

Copyright © 2014, Oracle und/oder verbundene Unternehmen. All rights reserved. Alle Rechte vorbehalten.

Prozeduren und Funktionen

- Sind benannte PL/SQL-Blöcke
- Werden als PL/SQL-Unterprogramme bezeichnet
- Haben ähnliche Blockstrukturen wie anonyme Blöcke:
 - Optionaler deklarativer Bereich (ohne Schlüsselwort `DECLARE`)
 - Obligatorischer ausführbarer Bereich
 - Optionaler Bereich zur Exception-Behandlung



ORACLE

Copyright © 2014, Oracle und/oder verbundene Unternehmen. All rights reserved. Alle Rechte vorbehalten.

Bislang wurden in diesem Kurs als Beispiele für PL/SQL-Code nur anonyme Blöcke besprochen. Wie der Name bereits sagt, sind *anonyme* Blöcke unbenannte ausführbare PL/SQL-Blöcke. Da sie unbenannt sind, können sie weder wiederverwendet noch zur späteren Verwendung gespeichert werden.

Prozeduren und Funktionen sind benannte PL/SQL-Blöcke, die auch als *Unterprogramme* bezeichnet werden. Diese Unterprogramme werden kompiliert und in der Datenbank gespeichert. Unterprogramme weisen eine ähnliche Blockstruktur wie anonyme Blöcke auf. Allerdings können Sie sie nicht nur auf Schemaebene, sondern auch in einem anderen PL/SQL-Block deklarieren. Unterprogramme enthalten folgende Bereiche:

- **Deklarativer Bereich:** Unterprogramme können einen optionalen deklarativen Bereich enthalten. Anders als bei anonymen Blöcken wird der optionale deklarative Bereich von Unterprogrammen nicht mit dem Schlüsselwort `DECLARE` eingeleitet, sondern mit dem Schlüsselwort `IS` oder `AS`.
- **Ausführbarer Abschnitt:** Dies ist der obligatorische Bereich des Unterprogramms, der die Implementierung der Geschäftslogik enthält. Anhand des Codes in diesem Bereich können Sie die Geschäftsfunktionalität des Unterprogramms leicht bestimmen. Dieser Bereich beginnt und endet mit den Schlüsselwörtern `BEGIN` und `END`.
- **Exception-Bereich:** Dieser Bereich zur Behandlung von Exceptions ist optional.

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	Sind benannt und können daher von anderen Anwendungen aufgerufen werden
Geben keine Werte zurück	Funktionen müssen Werte zurückgeben.
Können keine Parameter annehmen	Können Parameter annehmen

ORACLE®

Copyright © 2014, 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 herausgestellt.

Anonyme Blöcke sind keine persistenten Datenbankobjekte. Sie werden bei jeder Ausführung kompiliert. Sie werden nicht zur Wiederverwendung in der Datenbank gespeichert. Wenn Sie einen anonymen Block wiederverwenden möchten, müssen Sie das Skript, das den anonymen Block erstellt, erneut ausführen. Die Blöcke werden rekompiliert und ausgeführt.

Prozeduren und Funktionen werden kompiliert und in der Datenbank in kompilierter Form gespeichert. Sie werden nur bei Änderungen rekompiliert. Da diese Unterprogramme in der Datenbank gespeichert sind, können sie von allen Anwendungen verwendet werden, die über die entsprechende Berechtigung verfügen. Die aufrufende Anwendung kann Parameter an die Prozedur übergeben, wenn die Prozedur auf die Annahme von Parameter ausgerichtet ist. Analog kann eine aufrufende Anwendung einen Wert abrufen, wenn sie eine Funktion oder Prozedur aufruft.

Agenda

- Prozeduren und Funktionen – Einführung
- Prozeduren – Vorschau
- Funktionen – Vorschau

ORACLE

Copyright © 2014, Oracle und/oder verbundene Unternehmen. All rights reserved. Alle Rechte vorbehalten.

Prozeduren – Syntax

```
CREATE [OR REPLACE] PROCEDURE procedure_name
  [(argument1 [mode1] datatype1,
    argument2 [mode2] datatype2,
    . . .)]
IS|AS
procedure_body;
```



Copyright © 2014, Oracle und/oder verbundene Unternehmen. All rights reserved. Alle Rechte vorbehalten.

Die Folie zeigt die Syntax, mit der Sie Prozeduren erstellen. Für die Syntax gilt:

procedure_name Name der zu erstellenden Prozedur

argument Name des Prozedurparameters. Jedem Argument ist ein Modus und ein Datentyp zugeordnet. Sie können eine beliebige Anzahl von durch Komma getrennten Argumenten angeben.

mode Argumentmodus:
IN (Standard)
OUT
IN OUT

datatype Datentyp des zugeordneten Parameters. Der Datentyp von Parametern darf keine explizite Größe aufweisen. Verwenden Sie stattdessen %TYPE.

Procedure_body PL/SQL-Block, der den Code bildet

Die Argumentliste ist in Prozedurdeklarationen optional. Prozeduren werden im Kurs *Oracle Database: Develop PL/SQL Program Units* ausführlich behandelt.

Prozeduren erstellen

```
...
CREATE TABLE dept AS SELECT * FROM departments;
CREATE PROCEDURE add_dept IS
    v_dept_id dept.department_id%TYPE;
    v_dept_name dept.department_name%TYPE;
BEGIN
    v_dept_id:=280;
    v_dept_name:='ST-Curriculum';
    INSERT INTO dept(department_id,department_name)
    VALUES(v_dept_id,v_dept_name);
    DBMS_OUTPUT.PUT_LINE(' Inserted ' || SQL%ROWCOUNT
    || ' row ');
END;
```

ORACLE

Copyright © 2014, Oracle und/oder verbundene Unternehmen. All rights reserved. Alle Rechte vorbehalten.

Im Codebeispiel fügt die Prozedur `add_dept` eine neue Abteilung mit der Abteilungsnummer 280 und dem Abteilungsnamen `ST-Curriculum` ein.

Darüber hinaus veranschaulicht das Beispiel:

- Der deklarative Bereich der Prozedur beginnt nicht mit dem Schlüsselwort `DECLARE`, sondern unmittelbar hinter der Prozedurdeklaration.
- Die Prozedur deklariert die beiden Variablen `dept_id` und `dept_name`.
- Die Prozedur prüft mit dem impliziten Cursorattribut oder dem SQL-Attribut `SQL%ROWCOUNT`, ob die Zeile erfolgreich eingefügt wurde. In diesem Fall müsste der Wert 1 zurückgegeben werden.

Hinweis: Auf der folgenden Seite erhalten Sie weitere Hinweise zum Beispiel.

Prozeduren – Beispiel

Hinweise

- Beim Erstellen von Objekten werden die Einträge in die Tabelle `user_objects` geschrieben. Wenn der Code auf der Folie erfolgreich ausgeführt wird, können Sie in der Tabelle `user_objects` prüfen, ob die neuen Objekte vorhanden sind. Setzen Sie dazu folgenden Befehl ab:

```
SELECT object_name,object_type FROM user_objects;
```

OBJECT_NAME	OBJECT_TYPE
35 GREET	PROCEDURE
36 DEPT_PKG	PACKAGE
37 DEPT_PKG	PACKAGE BODY
38 RETIRED_EMPS	TABLE
39 ERROR_PKG	PACKAGE
40 EMPL	TABLE
41 MESSAGES	TABLE
42 HELLO	PROCEDURE
43 EMP	TABLE
44 DEPT	TABLE
45 ADD_DEPT	PROCEDURE

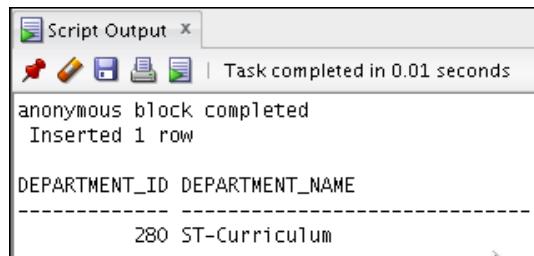
- Die Quelle der Prozedur wird in der Tabelle `user_source` gespeichert. Sie können die Quelle mit dem folgenden Befehl prüfen:

```
SELECT * FROM user_source WHERE name='ADD_DEPT';
```

NAME	TYPE	LINE	TEXT
1 ADD_DEPT PROCEDURE		1	PROCEDURE add_dept IS
2 ADD_DEPT PROCEDURE		2	v_dept_id dept.department_id%TYPE;
3 ADD_DEPT PROCEDURE		3	v_dept_name dept.department_name%TYPE;
4 ADD_DEPT PROCEDURE		4	BEGIN
5 ADD_DEPT PROCEDURE		5	v_dept_id:=280;
6 ADD_DEPT PROCEDURE		6	v_dept_name:='ST-Curriculum';
7 ADD_DEPT PROCEDURE		7	INSERT INTO dept(department_id,department_name)
8 ADD_DEPT PROCEDURE		8	VALUES(v_dept_id,v_dept_name);
9 ADD_DEPT PROCEDURE		9	DBMS_OUTPUT.PUT_LINE(' Inserted SQL%ROWCOUNT row ');
10 ADD_DEPT PROCEDURE		10	END;

Prozeduren aufrufen

```
...
BEGIN
  add_dept;
END;
/
SELECT department_id, department_name FROM dept
WHERE department_id=280;
```



The screenshot shows the 'Script Output' window from Oracle SQL Developer. It displays the results of an anonymous block execution. The output includes a message indicating the task completed in 0.01 seconds, a confirmation of the anonymous block completion, and a successful insert of one row. A final SELECT statement shows a single result row: DEPARTMENT_ID 280 and DEPARTMENT_NAME ST-Curriculum.

DEPARTMENT_ID	DEPARTMENT_NAME
280	ST-Curriculum

ORACLE

Copyright © 2014, Oracle und/oder verbundene Unternehmen. All rights reserved. Alle Rechte vorbehalten.

Die Folie zeigt, wie Sie eine Prozedur aus einem anonymen Block aufrufen. Der Aufruf der Prozedur muss in den ausführbaren Bereich des anonymen Blockes aufgenommen werden. Sie können die Prozedur auch aus einer beliebigen Anwendung aufrufen (zum Beispiel einer Forms- oder Java-Anwendung). Mit der `SELECT`-Anweisung im Code prüfen Sie, ob die Zeile erfolgreich eingefügt wurde.

Sie können Prozeduren auch mit der SQL-Anweisung `CALL <procedure_name>` aufrufen.

Agenda

- Prozeduren und Funktionen – Einführung
- Prozeduren – Vorschau
- Funktionen – Vorschau

ORACLE

Copyright © 2014, Oracle und/oder verbundene Unternehmen. All rights reserved. Alle Rechte vorbehalten.

Funktionen – Syntax

```
CREATE [OR REPLACE] FUNCTION function_name
  [(argument1 [mode1] datatype1,
    argument2 [mode2] datatype2,
    . . .)]
  RETURN datatype
  IS|AS
  function_body;
```



Copyright © 2014, Oracle und/oder verbundene Unternehmen. All rights reserved. Alle Rechte vorbehalten.

Die Folie zeigt die Syntax, mit der Sie Funktionen erstellen. Für die Syntax gilt:

<i>function_name</i>	Name der zu erstellenden Funktion
<i>argument</i>	Name des Funktionsparameters (Jedes Argument ist einem Modus und einem Datentyp zugeordnet. Sie können eine beliebige Anzahl von durch Komma getrennten Argumenten angeben. Das Argument wird übergeben, wenn Sie die Funktion aufrufen.)
<i>mode</i>	Parametertyp (Es dürfen nur IN-Parameter deklariert werden.)
<i>datatype</i>	Datentyp des zugeordneten Parameters
RETURN <i>datatype</i>	Datentyp des von der Funktion zurückgegebenen Wertes
<i>function_body</i>	PL/SQL-Block, der den Funktionscode bildet

Die Argumentliste ist in Funktionsdeklarationen optional. Im Gegensatz zu Prozeduren müssen Funktionen einen Wert an das aufrufende Programm zurückgeben. Daher enthält die Syntax den Rückgabewert *return_type*, mit dem Sie den Datentyp des von der Funktion zurückzugebenden Wertes angeben. Prozeduren können Werte über einen OUT- oder IN OUT-Parameter zurückgeben.

Funktionen erstellen

```
CREATE FUNCTION check_sal RETURN Boolean IS
  v_dept_id employees.department_id%TYPE;
  v_empno   employees.employee_id%TYPE;
  v_sal      employees.salary%TYPE;
  v_avg_sal  employees.salary%TYPE;
BEGIN
  v_empno:=205;
  SELECT salary,department_id INTO v_sal,v_dept_id FROM
employees
  WHERE employee_id= v_empno;
  SELECT avg(salary) INTO v_avg_sal FROM employees WHERE
department_id=v_dept_id;
  IF v_sal > v_avg_sal THEN
    RETURN TRUE;
  ELSE
    RETURN FALSE;
  END IF;
EXCEPTION
  WHEN NO_DATA_FOUND THEN
    RETURN NULL;
END;
```

ORACLE

Copyright © 2014, Oracle und/oder verbundene Unternehmen. All rights reserved. Alle Rechte vorbehalten.

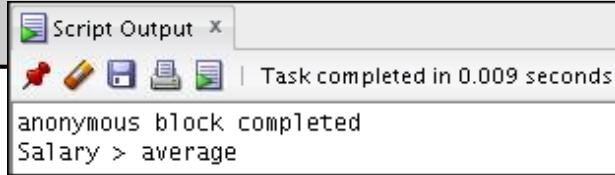
Funktionen – Beispiel

Mit der Funktion `check_sal` ermitteln Sie, ob das Gehalt eines bestimmten Mitarbeiters über oder unter dem Durchschnittsgehalt der Mitarbeiter in derselben Abteilung liegt. Die Funktion gibt `TRUE` zurück, wenn das Gehalt des Mitarbeiters über dem Durchschnittsgehalt der Mitarbeiter der Abteilung liegt. Andernfalls lautet der Rückgabewert `FALSE`. Die Funktion gibt `NULL` zurück, wenn die Exception `NO_DATA_FOUND` ausgelöst wird.

Beachten Sie, dass die Funktion auf den Mitarbeiter mit der Personalnummer 205 prüft. Die Funktion ist für die Suche nach ausschließlich dieser Personalnummer hartcodiert. Wenn Sie auch nach anderen Mitarbeitern suchen möchten, müssen Sie die Funktion ändern. Sie können dies umgehen, indem Sie die Funktion so deklarieren, dass sie ein Argument annimmt. Anschließend können Sie die Personalnummer als Parameter übergeben.

Funktionen aufrufen

```
BEGIN
  IF (check_sal IS NULL) THEN
    DBMS_OUTPUT.PUT_LINE('The function returned
      NULL due to exception');
  ELSIF (check_sal) THEN
    DBMS_OUTPUT.PUT_LINE('Salary > average');
  ELSE
    DBMS_OUTPUT.PUT_LINE('Salary < average');
  END IF;
END;
/
```



ORACLE

Copyright © 2014, Oracle und/oder verbundene Unternehmen. All rights reserved. Alle Rechte vorbehalten.

Nehmen Sie den Aufruf der Funktion in den ausführbaren Bereich des anonymen Blockes auf. Die Funktion wird im Rahmen einer Anweisung aufgerufen. Beachten Sie, dass die Funktion `check_sal` entweder Boolean oder NULL zurückgibt. Der Aufruf der Funktion wird daher als bedingter Ausdruck für den `IF`-Block aufgenommen.

Hinweis: Mit dem `DESCRIBE`-Befehl können Sie die Argumente prüfen und den Typ der Funktion zurückgeben:

```
DESCRIBE check_sal;
```

Parameter an Funktionen übergeben

```
DROP FUNCTION check_sal;
CREATE FUNCTION check_sal(p_empno employees.employee_id%TYPE)
RETURN Boolean IS
    v_dept_id employees.department_id%TYPE;
    v_sal      employees.salary%TYPE;
    v_avg_sal employees.salary%TYPE;
BEGIN
    SELECT salary,department_id INTO v_sal,v_dept_id FROM employees
        WHERE employee_id=p_empno;
    SELECT avg(salary) INTO v_avg_sal FROM employees
        WHERE department_id=v_dept_id;
    IF v_sal > v_avg_sal THEN
        RETURN TRUE;
    ELSE
        RETURN FALSE;
    END IF;
EXCEPTION
    ...
END;
```

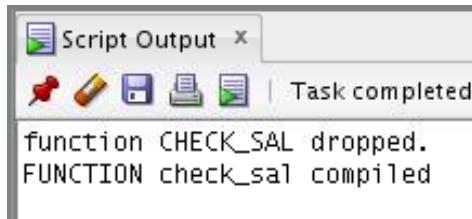
ORACLE

Copyright © 2014, Oracle und/oder verbundene Unternehmen. All rights reserved. Alle Rechte vorbehalten.

Wie bereits erwähnt, wurde die Funktion hartcodiert, um das Gehalt des Mitarbeiters mit der Personalnummer 205 zu prüfen. Der Code auf dieser Folie entfernt diese Einschränkung, da er die Personalnummer als Parameter annimmt. Sie können jetzt verschiedene Personalnummern übergeben und das jeweilige Gehalt prüfen.

Weitere Informationen zu Funktionen erhalten Sie im Kurs *Oracle Database: Develop PL/SQL Program Units*.

Das Codebeispiel auf der Folie ergibt folgende Ausgabe:



The screenshot shows the 'Script Output' window from Oracle SQL Developer. The window title is 'Script Output'. Below the title bar, there are several icons: a red square with a white exclamation mark, a pencil, a floppy disk, a printer, and a green checkmark. To the right of these icons is the text 'Task completed'. The main content area of the window displays the following text:
function CHECK_SAL dropped.
FUNCTION check_sal compiled

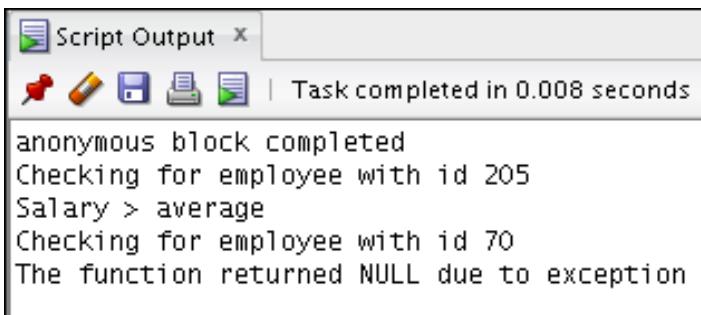
Funktionen mit einem Parameter aufrufen

```
BEGIN
    DBMS_OUTPUT.PUT_LINE('Checking for employee with id 205');
    IF (check_sal(205) IS NULL) THEN
        DBMS_OUTPUT.PUT_LINE('The function returned
            NULL due to exception');
    ELSIF (check_sal(205)) THEN
        DBMS_OUTPUT.PUT_LINE('Salary > average');
    ELSE
        DBMS_OUTPUT.PUT_LINE('Salary < average');
    END IF;
    DBMS_OUTPUT.PUT_LINE('Checking for employee with id 70');
    IF (check_sal(70) IS NULL) THEN
        DBMS_OUTPUT.PUT_LINE('The function returned
            NULL due to exception');
    ELSIF (check_sal(70)) THEN
        ...
    END IF;
END;
/
```

ORACLE

Copyright © 2014, Oracle und/oder verbundene Unternehmen. All rights reserved. Alle Rechte vorbehalten.

Der Code auf der Folie ruft die Funktion zweimal auf und übergibt dabei Parameter. Er generiert die folgende Ausgabe:



The screenshot shows a 'Script Output' window from Oracle SQL Developer. The window title is 'Script Output'. At the top, there are icons for running, stopping, and saving, followed by the message 'Task completed in 0.008 seconds'. The main area of the window displays the following text:
anonymous block completed
Checking for employee with id 205
Salary > average
Checking for employee with id 70
The function returned NULL due to exception

Quiz

Unterprogramme:

- a. sind benannte PL/SQL-Blöcke und können von anderen Anwendungen aufgerufen werden
- b. werden nur einmal kompiliert
- c. werden in der Datenbank gespeichert
- d. müssen keine Werte zurückgeben, wenn sie Funktionen sind
- e. können Parameter annehmen

ORACLE

Copyright © 2014, Oracle und/oder verbundene Unternehmen. All rights reserved. Alle Rechte vorbehalten.

Richtige Antworten: a, b, c, e

Zusammenfassung

In dieser Lektion haben Sie Folgendes gelernt:

- Einfache Prozeduren erstellen
- Prozeduren aus anonymen Blöcken aufrufen
- Einfache Funktionen erstellen
- Einfache Funktionen erstellen, die Parameter annehmen
- Funktionen aus anonymen Blöcken aufrufen



Copyright © 2014, Oracle und/oder verbundene Unternehmen. All rights reserved. Alle Rechte vorbehalten.

Mit anonymen Blöcken können Sie in PL/SQL beliebige Funktionalität entwickeln. Die Haupteinschränkung bei anonymen Blöcken besteht jedoch darin, dass sie nicht gespeichert werden und daher nicht wiederverwendbar sind.

Anstelle von anonymen Blöcken können Sie PL/SQL-Unterprogramme erstellen. Prozeduren und Funktionen werden als Unterprogramme (benannte PL/SQL-Blöcke) bezeichnet. Unterprogramme drücken wiederverwendbare Logik durch Parametrisierung aus. Prozeduren und Funktionen weisen eine ähnliche Struktur wie anonyme Blöcke auf. Diese Unterprogramme werden in der Datenbank gespeichert und sind daher wiederverwendbar.

Übungen zu Lektion 10 – Übersicht

Diese Übung behandelt folgende Themen:

- Vorhandenen anonymen Block in Prozedur konvertieren
- Prozedur so ändern, dass sie Parameter annimmt
- Anonymen Block zum Aufrufen der Prozedur erstellen



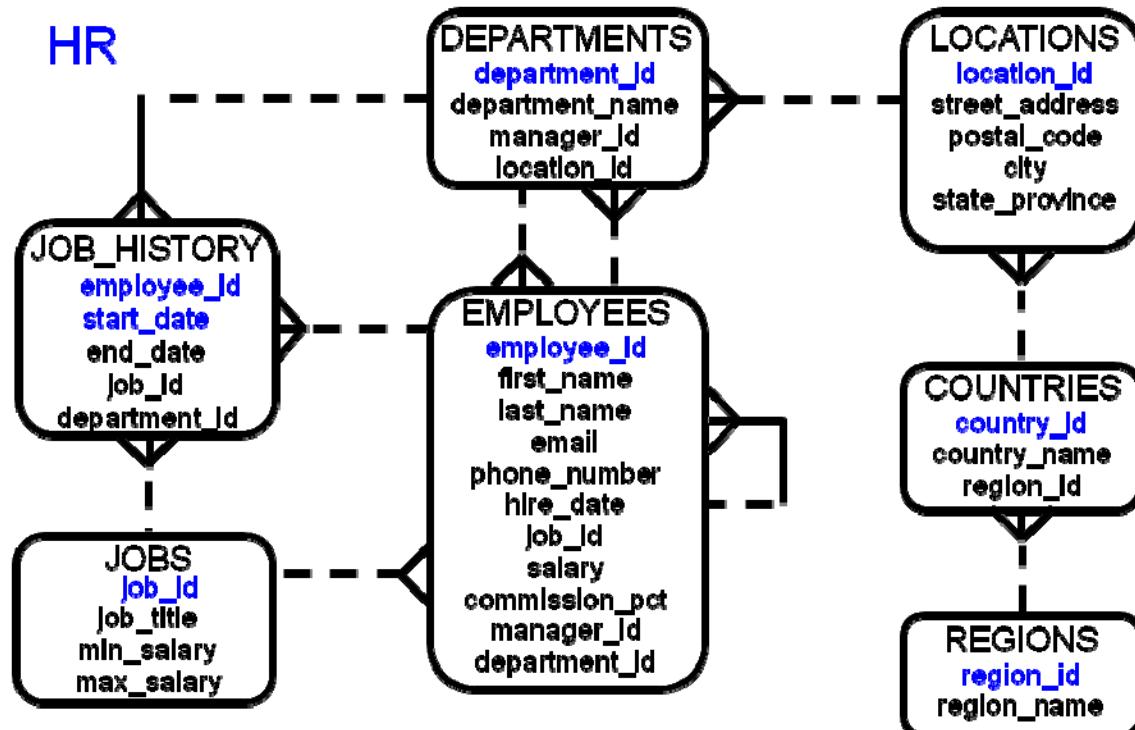
Copyright © 2014, Oracle und/oder verbundene Unternehmen. All rights reserved. Alle Rechte vorbehalten.

Anhang A:

Tabellenbeschreibungen

und -daten

ENTITY RELATIONSHIP DIAGRAM



Tabellen im Schema

```
SELECT * FROM tab;
```

TNAME	TABTYPE	CLUSTERID
COUNTRIES	TABLE	
DEPARTMENTS	TABLE	
EMPLOYEES	TABLE	
EMP_DETAILS_VIEW	VIEW	
JOB_HISTORY	TABLE	
JOBS	TABLE	
LOCATIONS	TABLE	
REGIONS	TABLE	

8 rows selected.

Tabelle regions

```
DESCRIBE regions
```

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

```
SELECT * FROM regions;
```

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

Tabelle countries

DESCRIBE countries

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

SELECT * FROM countries;

CO	COUNTRY_NAME	REGION_ID
AR	Argentina	2
AU	Australia	3
BE	Belgium	1
BR	Brazil	2
CA	Canada	2
CH	Switzerland	1
CN	China	3
DE	Germany	1
DK	Denmark	1
EG	Egypt	4
FR	France	1
HK	HongKong	3
IL	Israel	4
IN	India	3
CO	COUNTRY_NAME	REGION_ID
IT	Italy	1
JP	Japan	3
KW	Kuwait	4
MX	Mexico	2
NG	Nigeria	4
NL	Netherlands	1
SG	Singapore	3
UK	United Kingdom	1
US	United States of America	2
ZM	Zambia	4
ZW	Zimbabwe	4

25 rows selected.

Tabelle locations

DESCRIBE locations;

Name	Null?	Type
LOCATION_ID	NOT NULL	NUMBER(4)
STREET_ADDRESS		VARCHAR2(40)
POSTAL_CODE		VARCHAR2(12)
CITY	NOT NULL	VARCHAR2(30)
STATE_PROVINCE		VARCHAR2(25)
COUNTRY_ID		CHAR(2)

SELECT * FROM locations;

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

23 rows selected.

Tabelle departments

```
DESCRIBE departments
```

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

```
SELECT * FROM departments;
```

DEPARTMENT_ID	DEPARTMENT_NAME	MANAGER_ID	LOCATION_ID
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
DEPARTMENT_ID	DEPARTMENT_NAME	MANAGER_ID	LOCATION_ID
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

27 rows selected.

Tabelle jobs

DESCRIBE jobs

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

SELECT * FROM jobs;

JOB_ID	JOB_TITLE	MIN_SALARY	MAX_SALARY
AD_PRES	President	20000	40000
AD_VP	Administration Vice President	15000	30000
AD_ASST	Administration Assistant	3000	6000
FI_MGR	Finance Manager	8200	16000
FI_ACCOUNT	Accountant	4200	9000
AC_MGR	Accounting Manager	8200	16000
AC_ACCOUNT	Public Accountant	4200	9000
SA_MAN	Sales Manager	10000	20000
SA_REP	Sales Representative	6000	12000
PU_MAN	Purchasing Manager	8000	15000
PU_CLERK	Purchasing Clerk	2500	5500
ST_MAN	Stock Manager	5500	8500
ST_CLERK	Stock Clerk	2000	5000
SH_CLERK	Shipping Clerk	2500	5500
JOB_ID	JOB_TITLE	MIN_SALARY	MAX_SALARY
IT_PROG	Programmer	4000	10000
MK_MAN	Marketing Manager	9000	15000
MK_REP	Marketing Representative	4000	9000
HR_REP	Human Resources Representative	4000	9000
PR_REP	Public Relations Representative	4500	10500

19 rows selected.

Tabelle employees

```
DESCRIBE employees
```

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

Die Überschriften der Spalten commission_pct, manager_id und department_id wurden im folgenden Screenshot mit comm, mgrid beziehungsweise deptid abgekürzt, damit alle Tabellenwerte auf die Seite passen.

```
SELECT * FROM employees;
```

EMPLOYEE_ID	FIRST_NAME	LAST_NAME	EMAIL	PHONE_NUMBER	HIRE_DATE	JOB_ID	SALARY	comm	mgrid	deptid
100	Steven	King	SKING	515.123.4567	17-JUN-87	AD_PRES	24000			90
101	Neena	Kochhar	NKOCHHAR	515.123.4568	21-SEP-89	AD_VP	17000		100	90
102	Lex	De Haan	LDEHAAN	515.123.4569	13-JAN-93	AD_VP	17000		100	90
103	Alexander	Hunold	AHUNOLD	590.423.4567	03-JAN-90	IT_PROG	9000		102	60
104	Bruce	Ernst	BERNST	590.423.4568	21-MAY-91	IT_PROG	6000		103	60
105	David	Austin	DAUSTIN	590.423.4569	26-JUN-97	IT_PROG	4800		103	60
106	Valli	Pataballa	VPATABAL	590.423.4560	05-FEB-98	IT_PROG	4800		103	60
107	Diana	Lorentz	DLORENTZ	590.423.5567	07-FEB-99	IT_PROG	4200		103	60
108	Nancy	Greenberg	NGREENBE	515.124.4569	17-AUG-94	FI_MGR	12000		101	100
109	Daniel	Faviet	DFAVIET	515.124.4169	16-AUG-94	FI_ACCOUNT	9000		108	100
110	John	Chen	JCHEN	515.124.4269	28-SEP-97	FI_ACCOUNT	8200		108	100
111	Ismael	Sciarrra	ISCIARRA	515.124.4369	30-SEP-97	FI_ACCOUNT	7700		108	100
112	Jose Manuel	Urman	JMURMAN	515.124.4469	07-MAR-98	FI_ACCOUNT	7800		108	100
113	Luis	Popp	LPOPP	515.124.4567	07-DEC-99	FI_ACCOUNT	6900		108	100
EMPLOYEE_ID	FIRST_NAME	LAST_NAME	EMAIL	PHONE_NUMBER	HIRE_DATE	JOB_ID	SALARY	comm	mgrid	deptid
114	Den	Raphaely	DRAPHEAL	515.127.4561	07-DEC-94	PU_MAN	11000		100	30
115	Alexander	Khoo	AKHOO	515.127.4562	18-MAY-95	PU_CLERK	3100		114	30
116	Shelli	Baida	SBAIDA	515.127.4563	24-DEC-97	PU_CLERK	2900		114	30
117	Sigal	Tobias	STOBIAS	515.127.4564	24-JUL-97	PU_CLERK	2800		114	30
118	Guy	Himuro	GHIMURO	515.127.4565	15-NOV-98	PU_CLERK	2600		114	30
119	Karen	Colmenares	KCOLMENA	515.127.4566	10-AUG-99	PU_CLERK	2500		114	30
120	Matthew	Weiss	MWEISS	650.123.1234	18-JUL-96	ST_MAN	8000		100	50
121	Adam	Fripp	AFRIPP	650.123.2234	10-APR-97	ST_MAN	8200		100	50
122	Payam	Kaufling	PKAUFLIN	650.123.3234	01-MAY-95	ST_MAN	7900		100	50
123	Shanta	Vollman	SVOLLMAN	650.123.4234	10-OCT-97	ST_MAN	6500		100	50
124	Kevin	Mourgos	KMOURGOS	650.123.5234	16-NOV-99	ST_MAN	5800		100	50
125	Julia	Nayer	JNAYER	650.124.1214	16-JUL-97	ST_CLERK	3200		120	50
126	Irene	Mikkilineni	IMIKKILI	650.124.1224	28-SEP-98	ST_CLERK	2700		120	50
127	James	Landry	JLANDRY	650.124.1334	14-JAN-99	ST_CLERK	2400		120	50

EMPLOYEE_ID	FIRST_NAME	LAST_NAME	EMAIL	PHONE_NUMBER	HIRE_DATE	JOB_ID	SALARY	comm	mgrid	deptid
128	Steven	Markle	SMARKLE	650.124.1434	08-MAR-00	ST_CLERK	2200		120	50
129	Laura	Bissot	LBISSOT	650.124.5234	20-AUG-97	ST_CLERK	3300		121	50
130	Mozhe	Atkinson	MATKINSO	650.124.6234	30-OCT-97	ST_CLERK	2800		121	50
131	James	Marlow	JAMRLOW	650.124.7234	16-FEB-97	ST_CLERK	2500		121	50
132	TJ	Olson	TJOLSON	650.124.8234	10-APR-99	ST_CLERK	2100		121	50
133	Jason	Mallin	JMALLIN	650.127.1934	14-JUN-96	ST_CLERK	3300		122	50
134	Michael	Rogers	MROGERS	650.127.1834	26-AUG-98	ST_CLERK	2900		122	50
135	Ki	Gee	KGEE	650.127.1734	12-DEC-99	ST_CLERK	2400		122	50
136	Hazel	Philtanker	PHILTAN	650.127.1634	06-FEB-00	ST_CLERK	2200		122	50
137	Renske	Ladwig	RLADWIG	650.121.1234	14-JUL-95	ST_CLERK	3600		123	50
138	Stephen	Stiles	SSTILES	650.121.2034	26-OCT-97	ST_CLERK	3200		123	50
139	John	Seo	JSEO	650.121.2019	12-FEB-98	ST_CLERK	2700		123	50
140	Joshua	Patel	JPATEL	650.121.1834	06-APR-98	ST_CLERK	2500		123	50
141	Trenna	Rajs	TRAJS	650.121.8009	17-OCT-95	ST_CLERK	3500		124	50
EMPLOYEE_ID	FIRST_NAME	LAST_NAME	EMAIL	PHONE_NUMBER	HIRE_DATE	JOB_ID	SALARY	comm	mgrid	deptid
142	Curtis	Davies	CDAMES	650.121.2994	29-JAN-97	ST_CLERK	3100		124	50
143	Randall	Matos	RMATOS	650.121.2874	15-MAR-98	ST_CLERK	2600		124	50
144	Peter	Vargas	PVARGAS	650.121.2004	09-JUL-98	ST_CLERK	2500		124	50
145	John	Russell	JRUSSEL	011.44.1344.429268	01-OCT-96	SA_MAN	14000	.4	100	80
146	Karen	Partners	KPARTNER	011.44.1344.467268	05-JAN-97	SA_MAN	13500	.3	100	80
147	Alberto	Errazuriz	AERRAZUR	011.44.1344.429278	10-MAR-97	SA_MAN	12000	.3	100	80
148	Gerald	Cambrault	GCAMBRAU	011.44.1344.619268	15-OCT-99	SA_MAN	11000	.3	100	80
149	Beni	Zlotkey	EZLOTKEY	011.44.1344.429018	29-JAN-00	SA_MAN	10500	.2	100	80
150	Peter	Tucker	PTUCKER	011.44.1344.129268	30-JAN-97	SA REP	10000	.3	145	80
151	David	Bernstein	DBERNSTE	011.44.1344.345268	24-MAR-97	SA REP	9500	.25	145	80
152	Peter	Hall	PHALL	011.44.1344.478968	20-AUG-97	SA REP	9000	.25	145	80
153	Christopher	Olsen	COLSEN	011.44.1344.498718	30-MAR-98	SA REP	8000	.2	145	80
154	Nanette	Cambrault	NCAMBRAU	011.44.1344.987668	09-DEC-98	SA REP	7500	.2	145	80
155	Oliver	Tuvault	OTUVAVLT	011.44.1344.486508	23-NOV-99	SA REP	7000	.15	145	80
EMPLOYEE_ID	FIRST_NAME	LAST_NAME	EMAIL	PHONE_NUMBER	HIRE_DATE	JOB_ID	SALARY	comm	mgrid	deptid
156	Janette	King	JKING	011.44.1345.429268	30-JAN-96	SA REP	10000	.35	146	80
157	Patrick	Sully	PSULLY	011.44.1345.929268	04-MAR-96	SA REP	9500	.35	146	80
158	Allan	McEwen	AMCEWEN	011.44.1345.829268	01-AUG-96	SA REP	9000	.35	146	80
159	Lindsey	Smith	LSMITH	011.44.1346.729268	10-MAR-97	SA REP	8000	.3	146	80
160	Louise	Doran	LDORAN	011.44.1346.629268	15-DEC-97	SA REP	7500	.3	146	80
161	Sarath	Sewall	SSEWALL	011.44.1345.529268	03-NOV-98	SA REP	7000	.25	146	80
162	Clara	Mishney	CMISHNEY	011.44.1346.129268	11-NOV-97	SA REP	10500	.25	147	80
163	Danielle	Greene	DGREENE	011.44.1346.229268	19-MAR-99	SA REP	9500	.15	147	80
164	Mattea	Marvins	MMARVINS	011.44.1346.329268	24-JAN-00	SA REP	7200	.1	147	80
165	David	Lee	DLEE	011.44.1346.529268	23-FEB-00	SA REP	6800	.1	147	80
166	Sundar	Ande	SANDE	011.44.1346.629268	24-MAR-00	SA REP	6400	.1	147	80
167	Amit	Banda	ABANDA	011.44.1346.729268	21-APR-00	SA REP	6200	.1	147	80
168	Lisa	Ozer	LOZER	011.44.1343.929268	11-MAR-97	SA REP	11500	.25	148	80
169	Harrison	Bloom	HBLOOM	011.44.1343.829268	23-MAR-98	SA REP	10000	.2	148	80

EMPLOYEE_ID	FIRST_NAME	LAST_NAME	EMAIL	PHONE_NUMBER	HIRE_DATE	JOB_ID	SALARY	comm	mgrid	deptid
170	Tayler	Fox	TFFOX	011.44.1343.729268	24-JAN-98	SA_REP	9600	.2	148	80
171	William	Smith	WWSMITH	011.44.1343.629268	23-FEB-99	SA_REP	7400	.15	148	80
172	Elizabeth	Bates	EBATES	011.44.1343.529268	24-MAR-99	SA_REP	7300	.15	148	80
173	Sundita	Kumar	SKUMAR	011.44.1343.329268	21-APR-00	SA_REP	6100	.1	148	80
174	Elen	Abel	EABEL	011.44.1644.429267	11-MAY-96	SA_REP	11000	.3	149	80
175	Alyssa	Hutton	AHUTTON	011.44.1644.429266	19-MAR-97	SA_REP	8800	.25	149	80
176	Jonathon	Taylor	JTAYLOR	011.44.1644.429265	24-MAR-98	SA_REP	8600	.2	149	80
177	Jack	Livingston	JLIVINGS	011.44.1644.429264	23-APR-98	SA_REP	8400	.2	149	80
178	Kimberely	Grant	KGRANT	011.44.1644.429263	24-MAY-99	SA_REP	7000	.15	149	
179	Charles	Johnson	CJOHNSON	011.44.1644.429262	04-JAN-00	SA_REP	6200	.1	149	80
180	Winston	Taylor	WTAYLOR	650.507.9876	24-JAN-98	SH_CLERK	3200		120	50
181	Jean	Fleur	JFLEUR	650.507.9877	23-FEB-98	SH_CLERK	3100		120	50
182	Martha	Sullivan	MSULLIVA	650.507.9878	21-JUN-99	SH_CLERK	2500		120	50
183	Girard	Geoni	GGEOINI	650.507.9879	03-FEB-00	SH_CLERK	2800		120	50
EMPLOYEE_ID	FIRST_NAME	LAST_NAME	EMAIL	PHONE_NUMBER	HIRE_DATE	JOB_ID	SALARY	comm	mgrid	deptid
184	Nandita	Sarchand	NSARCHAN	650.509.1876	27-JAN-98	SH_CLERK	4200		121	50
185	Alexis	Bull	ABULL	650.509.2876	20-FEB-97	SH_CLERK	4100		121	50
186	Julia	Dellinger	JDELLING	650.509.3876	24-JUN-98	SH_CLERK	3400		121	50
187	Anthony	Cabrio	ACABRIO	650.509.4876	07-FEB-99	SH_CLERK	3000		121	50
188	Kelly	Chung	KCHUNG	650.505.1876	14-JUN-97	SH_CLERK	3800		122	50
189	Jennifer	Dilly	JDILLY	650.505.2876	13-AUG-97	SH_CLERK	3600		122	50
190	Timothy	Gates	TGATES	650.505.3876	11-JUL-98	SH_CLERK	2900		122	50
191	Randall	Perkins	RPERKINS	650.505.4876	19-DEC-99	SH_CLERK	2500		122	50
192	Sarah	Bell	SBELL	650.501.1876	04-FEB-96	SH_CLERK	4000		123	50
193	Britney	Everett	BEVERETT	650.501.2876	03-MAR-97	SH_CLERK	3900		123	50
194	Samuel	McCain	SMCCAIN	650.501.3876	01-JUL-98	SH_CLERK	3200		123	50
195	Vance	Jones	VJONES	650.501.4876	17-MAR-99	SH_CLERK	2800		123	50
196	Alana	Walsh	AWALSH	650.507.9811	24-APR-98	SH_CLERK	3100		124	50
197	Kevin	Feehey	KFEENEY	650.507.9822	23-MAY-98	SH_CLERK	3000		124	50
EMPLOYEE_ID	FIRST_NAME	LAST_NAME	EMAIL	PHONE_NUMBER	HIRE_DATE	JOB_ID	SALARY	comm	mgrid	deptid
198	Donald	OConnell	DOCONNEL	650.507.9833	21-JUN-99	SH_CLERK	2600		124	50
199	Douglas	Grant	DGRANT	650.507.9844	13-JAN-00	SH_CLERK	2600		124	50
200	Jennifer	Whalen	JWHALEN	515.123.4444	17-SEP-87	AD_ASST	4400		101	10
201	Michael	Hartstein	MHARTSTE	515.123.5555	17-FEB-96	MK_MAN	13000		100	20
202	Pat	Fay	PFAY	603.123.6666	17-AUG-97	MK REP	6000		201	20
203	Susan	Mavris	SMAVRIS	515.123.7777	07-JUN-94	HR REP	6500		101	40
204	Hermann	Baer	HBAER	515.123.8888	07-JUN-94	PR REP	10000		101	70
205	Shelley	Higgins	SHIGGINS	515.123.8080	07-JUN-94	AC_MGR	12000		101	110
206	William	Gietz	WGIETZ	515.123.8181	07-JUN-94	AC_ACCOUNT	8300		205	110

107 rows selected.

Tabelle job_history

```
DESCRIBE job_history
```

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

EMPLOYEE_ID	START_DATE	END_DATE	JOB_ID	deptid
102	13-JAN-93	24-JUL-98	IT_PROG	60
101	21-SEP-89	27-OCT-93	AC_ACCOUNT	110
101	28-OCT-93	15-MAR-97	AC_MGR	110
201	17-FEB-96	19-DEC-99	MK_REP	20
114	24-MAR-98	31-DEC-99	ST_CLERK	50
122	01-JAN-99	31-DEC-99	ST_CLERK	50
200	17-SEP-87	17-JUN-93	AD_ASST	90
176	24-MAR-98	31-DEC-98	SA_REP	80
176	01-JAN-99	31-DEC-99	SA_MAN	80
200	01-JUL-94	31-DEC-98	AC_ACCOUNT	90

10 rows selected.

JB SQL Developer

ORACLE

Copyright © 2014, Oracle und/oder verbundene Unternehmen. All rights reserved. Alle Rechte vorbehalten.

Ziele

Nach Beendigung dieses Anhangs haben Sie folgende Ziele erreicht:

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

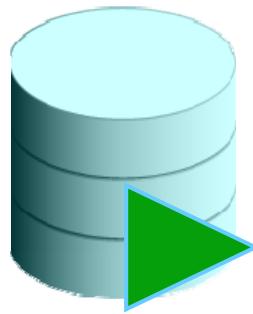


Copyright © 2014, Oracle und/oder verbundene Unternehmen. All rights reserved. Alle Rechte vorbehalten.

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

Was ist Oracle SQL Developer?

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



SQL Developer

ORACLE®

Copyright © 2014, Oracle und/oder verbundene Unternehmen. All rights reserved. Alle Rechte vorbehalten.

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

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

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

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

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

SQL Developer – Spezifikationen

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

ORACLE

Copyright © 2014, Oracle und/oder verbundene Unternehmen. All rights reserved. Alle Rechte vorbehalten.

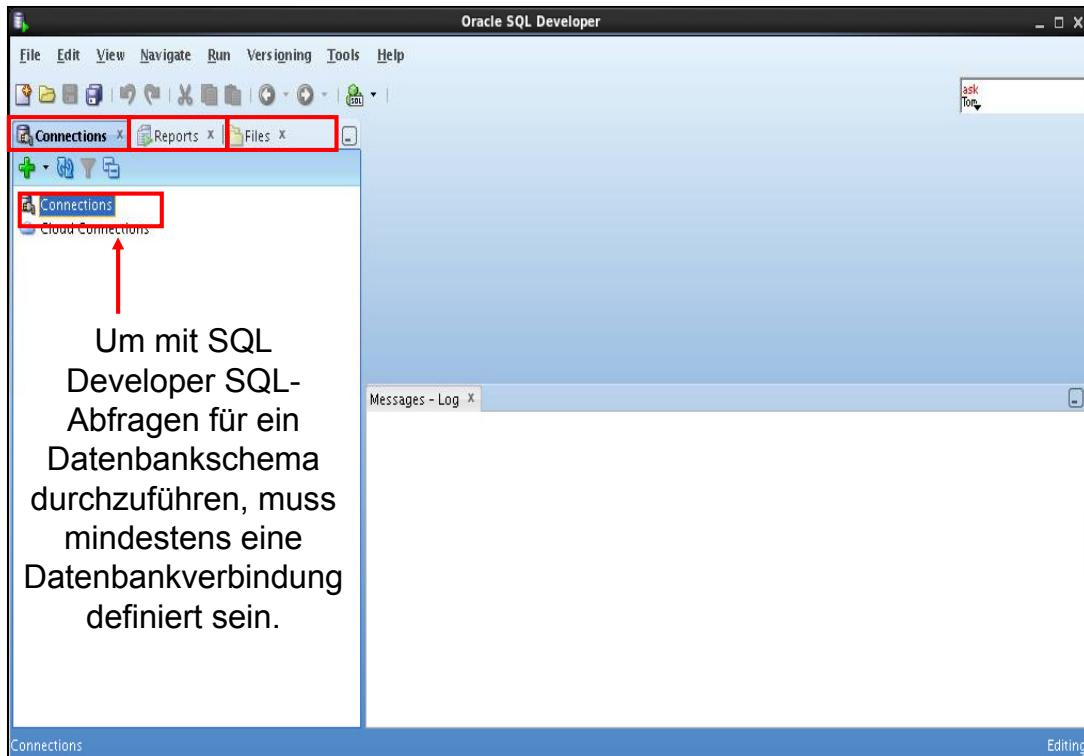
Oracle SQL Developer ist standardmäßig im Lieferumfang von Oracle Database 12c Release 1 enthalten. SQL Developer wurde mithilfe der Oracle JDeveloper-IDE (Integrated Development Environment, integrierte Entwicklungsumgebung) in Java entwickelt. Somit handelt es sich um ein plattformübergreifendes Tool,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

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

Hinweis:

- Für Oracle Database 12c Release 1 müssen Sie SQL Developer herunterladen und installieren. SQL Developer kann kostenlos über folgenden Link heruntergeladen werden:
<http://www.oracle.com/technetwork/developer-tools/sql-developer/downloads/index.html>
- Anweisungen zur Installation von SQL Developer finden Sie auf der Website unter:
<http://www.oracle.com/technetwork/developer-tools/sql-developer/overview/index.html>

SQL Developer 3.2 – Benutzeroberfläche



ORACLE

Copyright © 2014, Oracle und/oder verbundene Unternehmen. All rights reserved. Alle Rechte vorbehalten.

Die Benutzeroberfläche von SQL Developer umfasst (von links nach rechts) drei Hauptregisterkarten für die Navigation:

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

Allgemeine Navigation und Verwendung

In SQL Developer navigieren Sie auf der linken Seite, um Objekte zu suchen und zu wählen. Auf der rechten Seite werden Informationen über die gewählten Objekte angezeigt. Über Voreinstellungen können Sie viele Aspekte der Darstellung und des Verhaltens von SQL Developer anpassen.

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

Menüs

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

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

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

Datenbankverbindungen erstellen

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

ORACLE

Copyright © 2014, Oracle und/oder verbundene Unternehmen. All rights reserved. Alle Rechte vorbehalten.

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

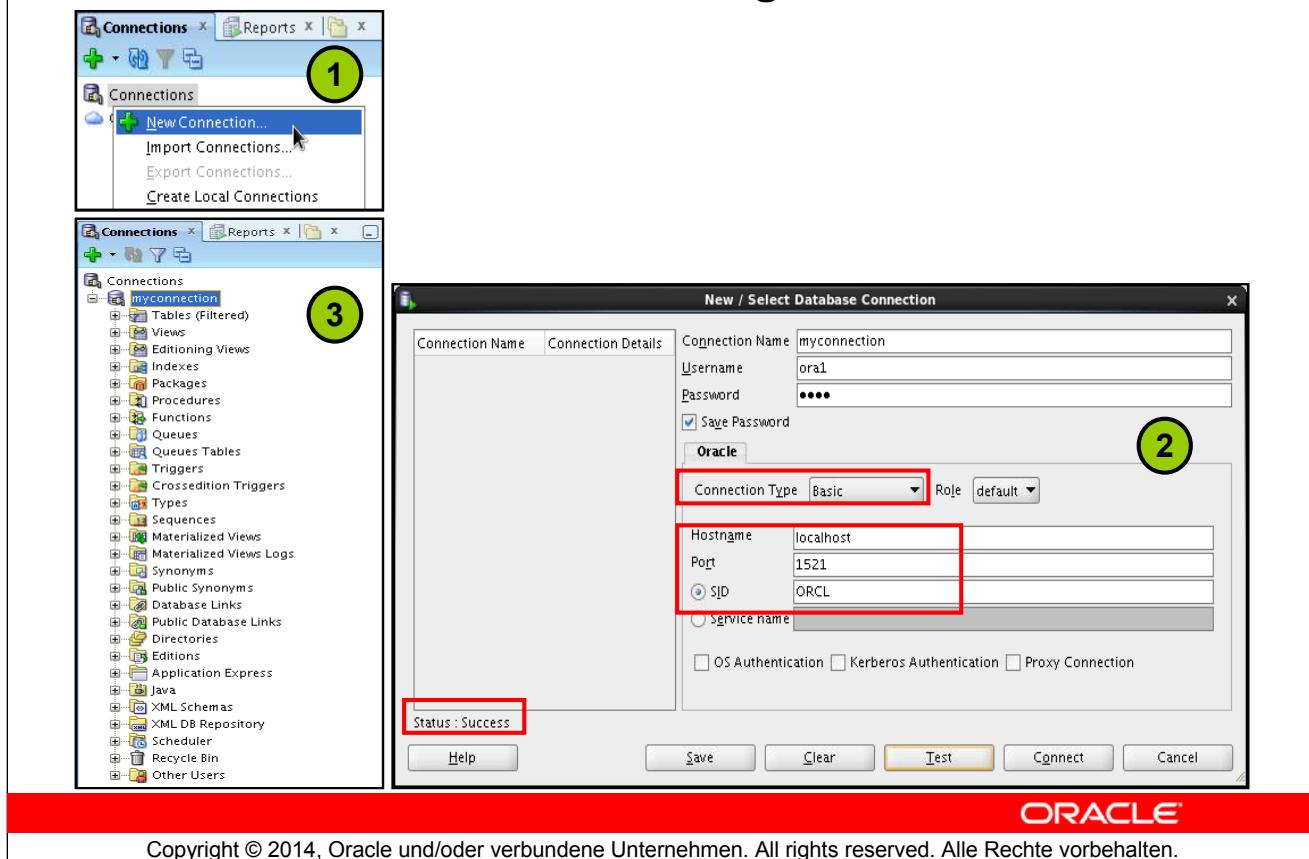
Sie können Verbindungen für mehrere Datenbanken und mehrere Schemas erstellen und testen. Standardmäßig befindet sich die Datei `tnsnames.ora` im Verzeichnis `$ORACLE_HOME/network/admin`. Die Datei kann jedoch auch in dem durch die Umgebungsvariable `TNS_ADMIN` oder durch einen Registrierungswert angegebenen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

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

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

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

Datenbankverbindungen erstellen



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

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

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

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

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

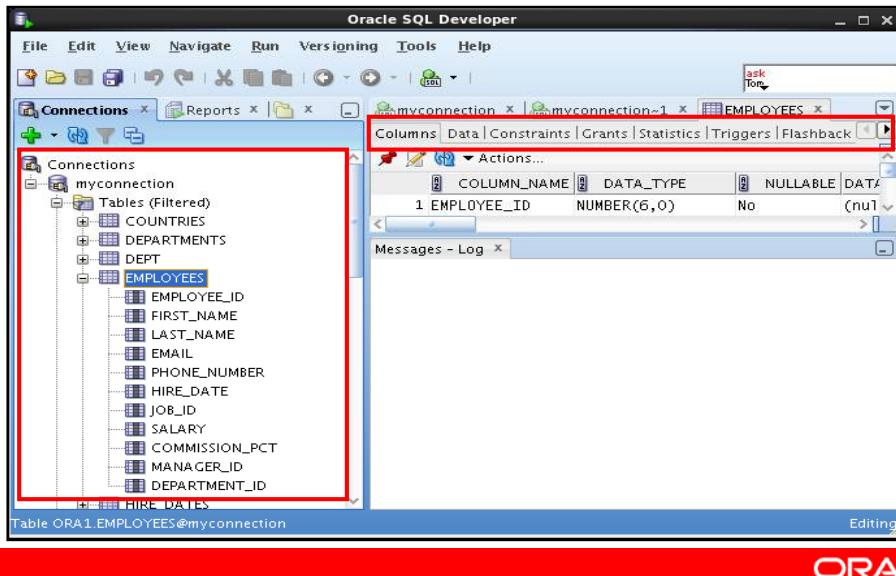
3. Die Verbindung wird dem Connections Navigator hinzugefügt. Sie können sie einblenden, um die Datenbankobjekte und Objektdefinitionen anzuzeigen, etwa Abhängigkeiten, Details und Statistiken.

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

Datenbankobjekte durchsuchen

Mit dem Connections Navigator können Sie:

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



Copyright © 2014, Oracle und/oder verbundene Unternehmen. All rights reserved. Alle Rechte vorbehalten.

ORACLE

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

In SQL Developer navigieren Sie auf der linken Seite, um Objekte zu suchen und zu wählen. Auf der rechten Seite werden Informationen über die gewählten Objekte angezeigt. Über Voreinstellungen können Sie zahlreiche Aspekte der Darstellung von SQL Developer anpassen.

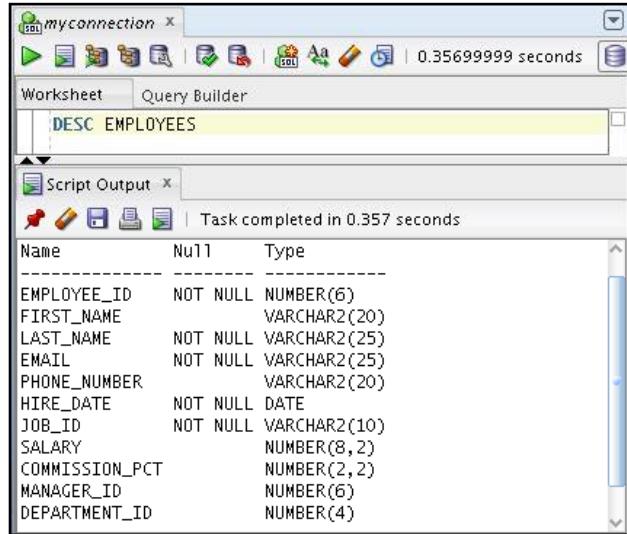
Die Definitionen der Objekte werden in verschiedenen Registerkarten angezeigt, die Informationen aus dem Data Dictionary enthalten. Wenn Sie beispielsweise eine Tabelle im Navigator wählen, werden alle Einzelheiten über Spalten, Constraints, Berechtigungen, Statistiken, Trigger und so weiter in einem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

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

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

Tabellenstruktur anzeigen

Struktur einer Tabelle mit dem Befehl DESCRIBE anzeigen:



The screenshot shows the Oracle SQL Developer interface. In the top-left window, titled 'myconnection x', there is a toolbar with various icons. Below the toolbar, the title bar says 'Worksheet' and 'Query Builder'. A yellow-highlighted row in the worksheet area contains the text 'DESC EMPLOYEES'. To the right of the worksheet is a 'Script Output' window with a title bar 'Script Output x'. It displays the output of the DESCRIBE command, which lists the columns of the EMPLOYEES table with their names, nullability, and data types. The output is as follows:

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

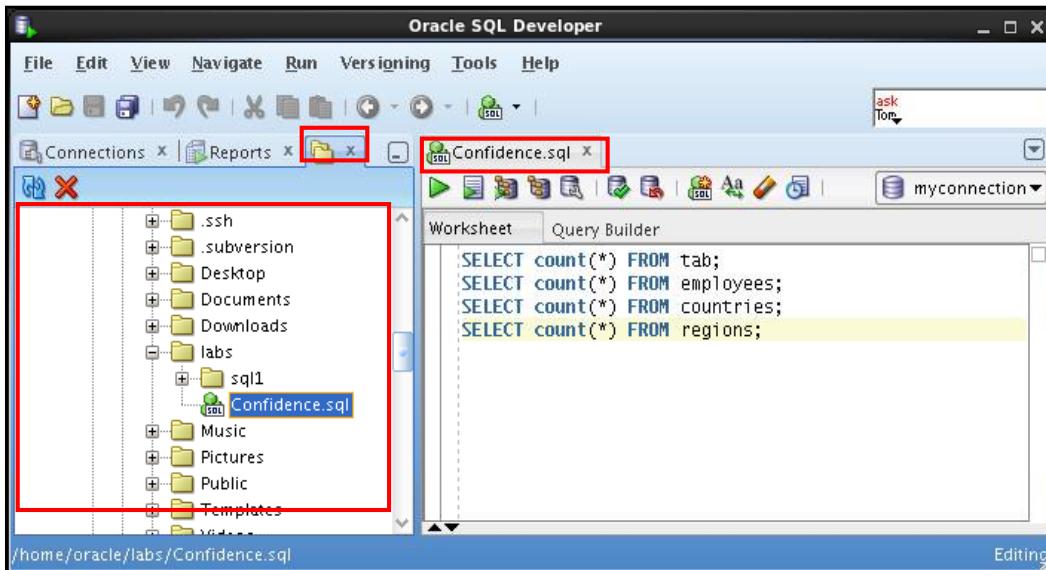
ORACLE

Copyright © 2014, Oracle und/oder verbundene Unternehmen. All rights reserved. Alle Rechte vorbehalten.

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

Dateien durchsuchen

Mit dem File Navigator das Dateisystem untersuchen und Systemdateien öffnen



ORACLE

Copyright © 2014, Oracle und/oder verbundene Unternehmen. All rights reserved. Alle Rechte vorbehalten.

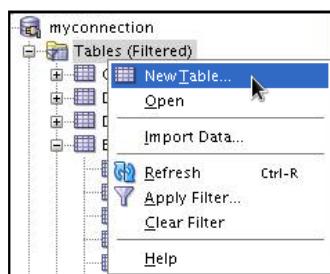
Datenbankobjekte durchsuchen

Mit dem File Navigator können Sie Systemdateien durchsuchen und öffnen.

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

Schemaobjekte erstellen

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



ORACLE®

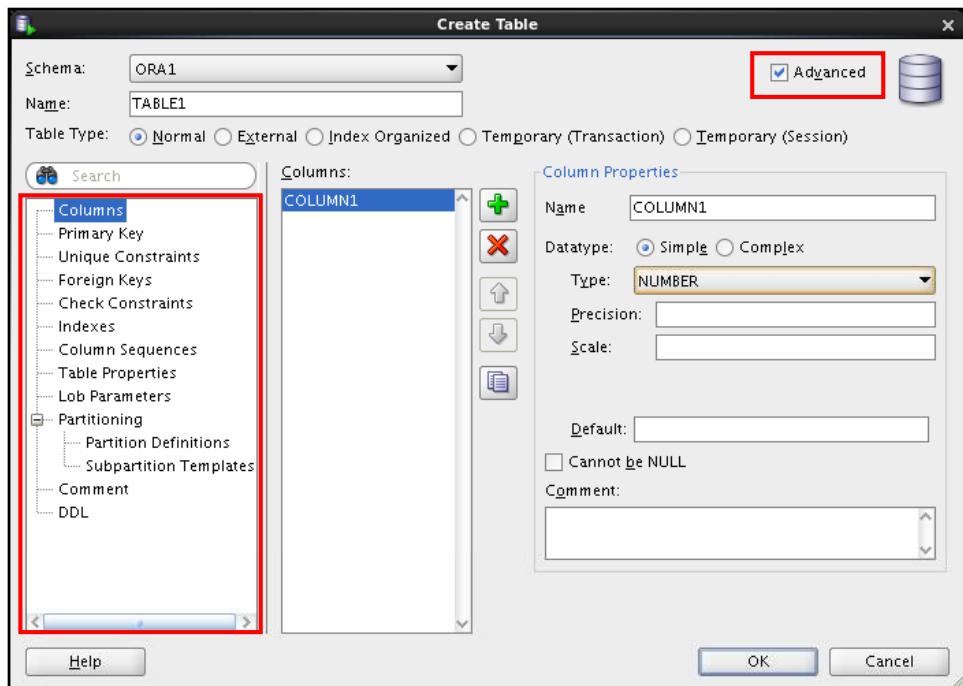
Copyright © 2014, Oracle und/oder verbundene Unternehmen. All rights reserved. Alle Rechte vorbehalten.

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

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

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

Neue Tabellen erstellen – Beispiel



ORACLE

Copyright © 2014, Oracle und/oder verbundene Unternehmen. All rights reserved. Alle Rechte vorbehalten.

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

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

Das Beispiel auf der Folie zeigt, wie die Tabelle **DEPENDENTS** mit aktiviertem Kontrollkästchen **Advanced** erstellt wird.

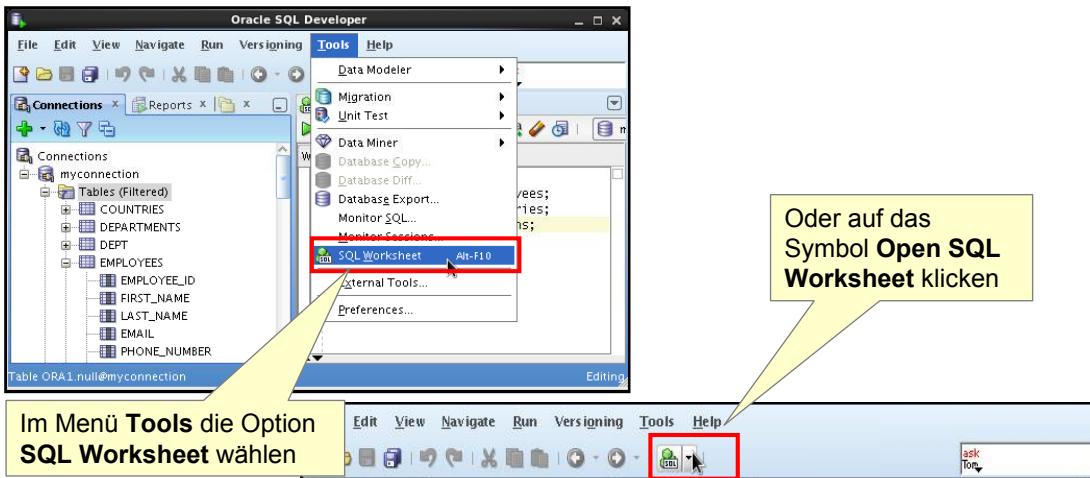
Um eine neue Tabelle zu erstellen, gehen Sie wie folgt vor:

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

Es empfiehlt sich, darüber hinaus in der Registerkarte **Primary Key** einen Primärschlüssel anzugeben. Die erstellte Tabelle kann auch bearbeitet werden. Klicken Sie hierfür im Connections Navigator mit der rechten Maustaste auf die gewünschte Tabelle, und wählen Sie **Edit**.

SQL Worksheet – Einsatzmöglichkeiten

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



Copyright © 2014, Oracle und/oder verbundene Unternehmen. All rights reserved. Alle Rechte vorbehalten.

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

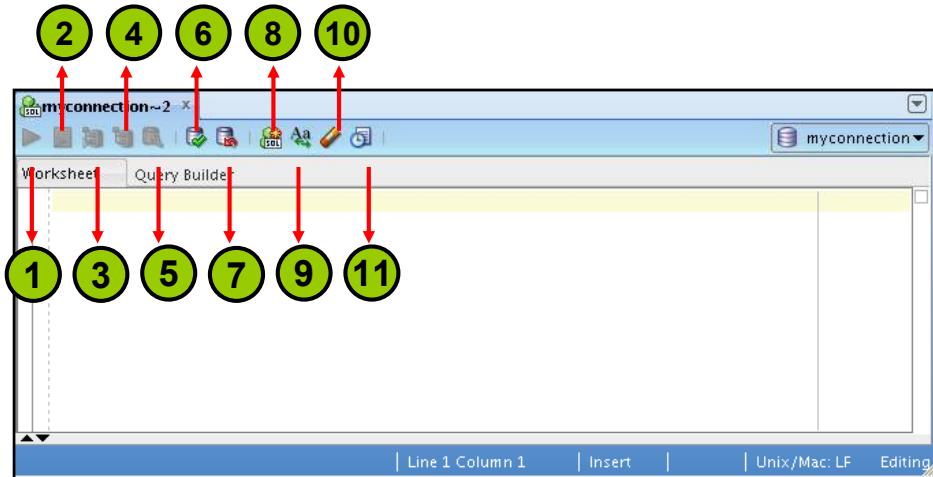
Sie können angeben, welche Aktionen die mit dem Worksheet verknüpfte Datenbankverbindung verarbeiten kann. Beispiele:

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

Um ein SQL Worksheet-Fenster anzuzeigen, führen Sie eine der folgenden Aktionen aus:

- Wählen Sie im Menü **Tools** die Option **SQL Worksheet**.
- Klicken Sie auf das Symbol **Open SQL Worksheet**.

SQL Worksheet – Einsatzmöglichkeiten



ORACLE

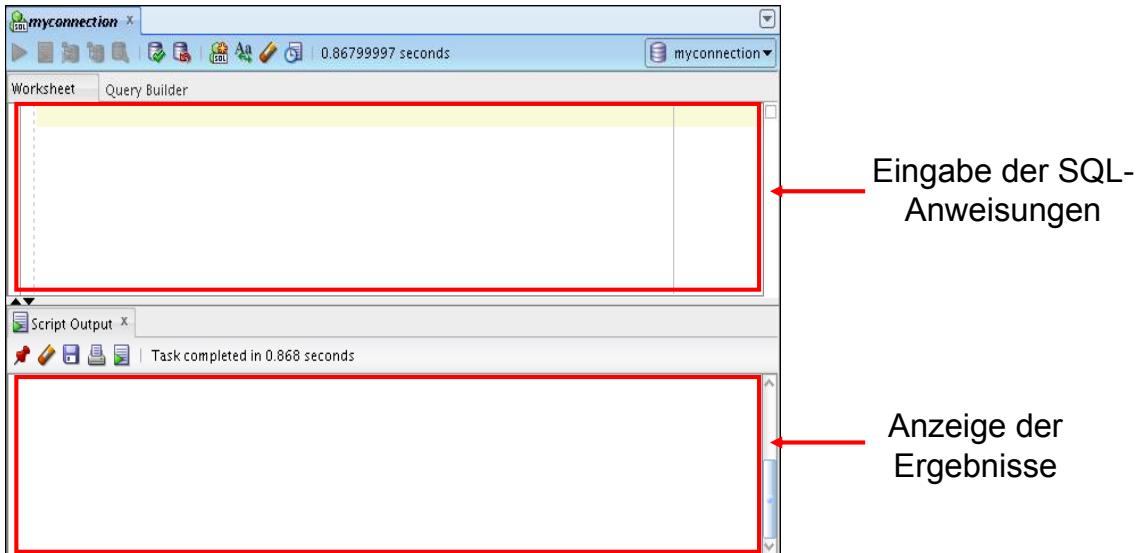
Copyright © 2014, Oracle und/oder verbundene Unternehmen. All rights reserved. Alle Rechte vorbehalten.

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

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

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

SQL Worksheet – Einsatzmöglichkeiten



ORACLE

Copyright © 2014, Oracle und/oder verbundene Unternehmen. All rights reserved. Alle Rechte vorbehalten.

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

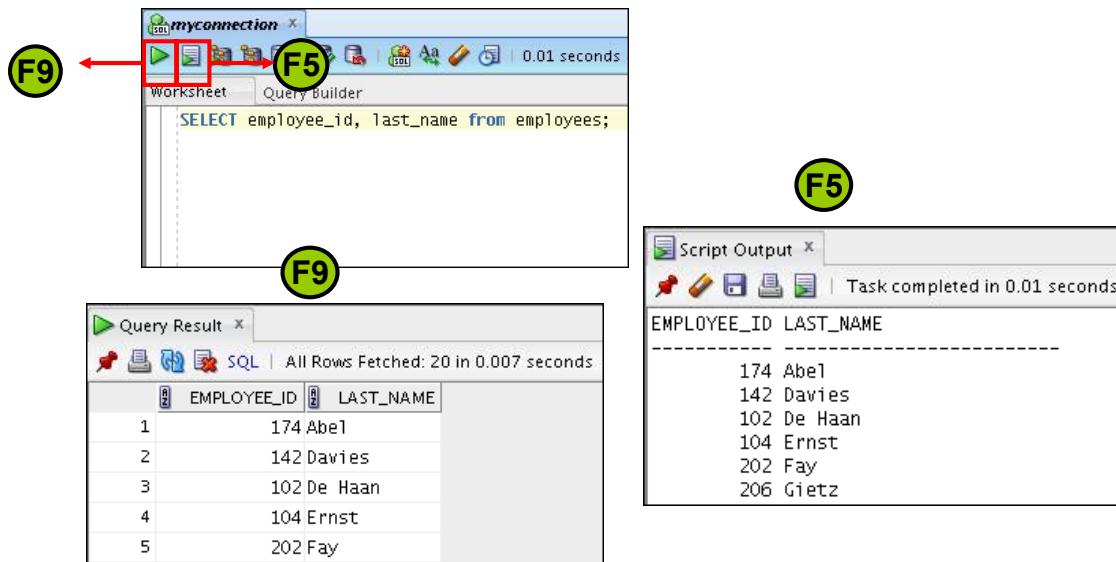
Das SQL Worksheet unterstützt derzeit eine Reihe von SQL*Plus-Befehlen. Nicht unterstützte SQL*Plus-Befehle werden ignoriert und nicht an die Oracle-Datenbank gesendet. Mit dem SQL Worksheet lassen sich SQL-Anweisungen und einige SQL*Plus-Befehle ausführen.

Um ein SQL Worksheet-Fenster anzuzeigen, führen Sie eine der folgenden Aktionen aus:

- Wählen Sie im Menü **Tools** die Option **SQL Worksheet**.
- Klicken Sie auf das Symbol **Open SQL Worksheet**.

SQL-Anweisungen ausführen

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

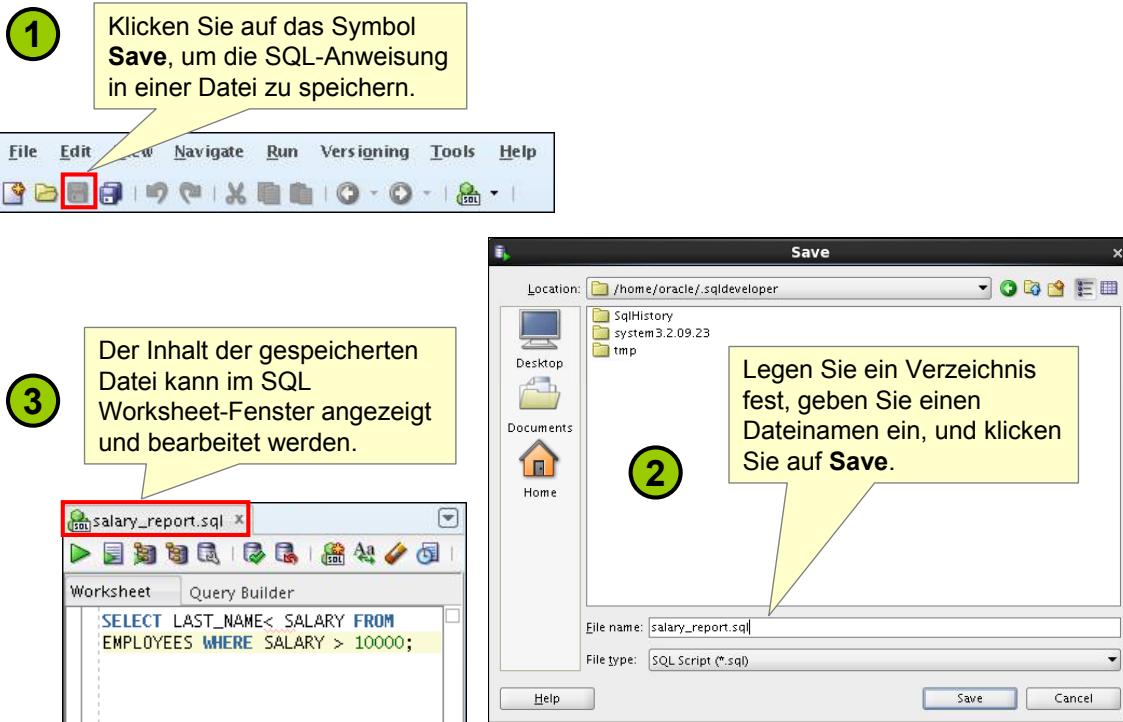


ORACLE

Copyright © 2014, Oracle und/oder verbundene Unternehmen. All rights reserved. Alle Rechte vorbehalten.

Das Beispiel auf der Folie zeigt zwei verschiedene Ausgaben für dieselbe Abfrage. In einem Fall wurde die Abfrage über die Taste F9 oder das Symbol **Execute Statement** und im anderen Fall über die Taste F5 oder das Symbol **Run Script** ausgeführt.

SQL-Skripte speichern



ORACLE

Copyright © 2014, Oracle und/oder verbundene Unternehmen. All rights reserved. Alle Rechte vorbehalten.

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

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

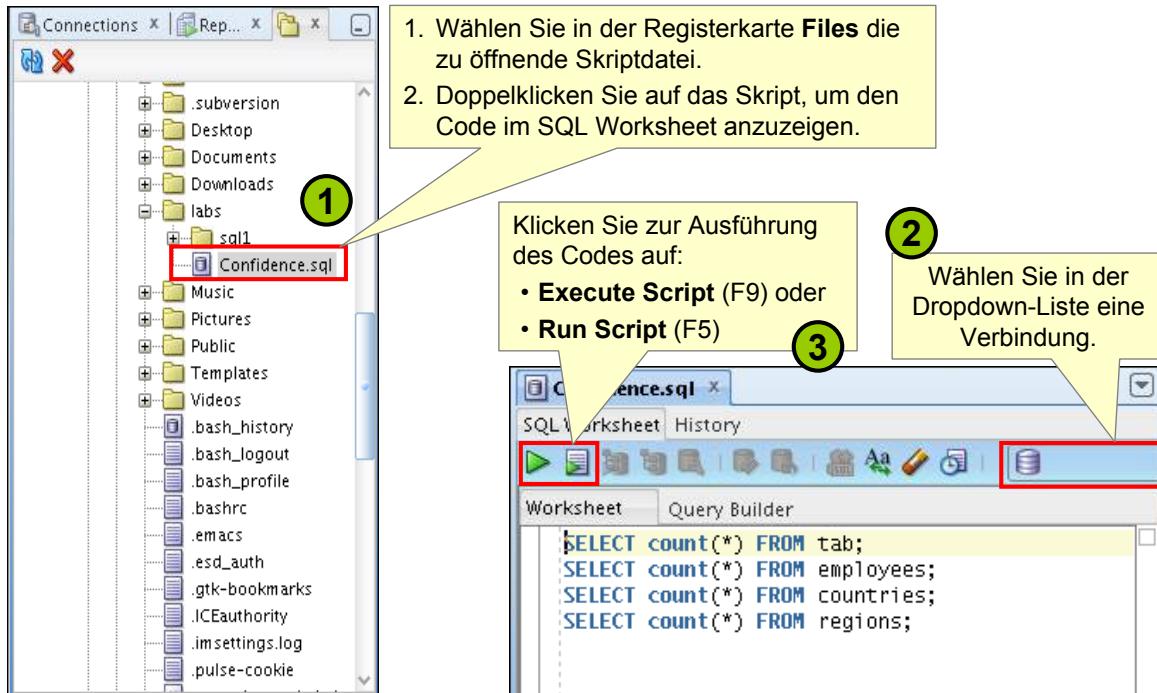
Nachdem Sie den Inhalt in einer Datei gespeichert haben, wird im Fenster **Enter SQL Statement** eine Registerkarte mit dem Dateinhalt angezeigt. Es können mehrere Dateien gleichzeitig geöffnet sein. Jede Datei wird in Form einer Registerkarte angezeigt.

Skriptpfad

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

Gespeicherte Skriptdateien ausführen –

1. Methode



ORACLE

Copyright © 2014, Oracle und/oder verbundene Unternehmen. All rights reserved. Alle Rechte vorbehalten.

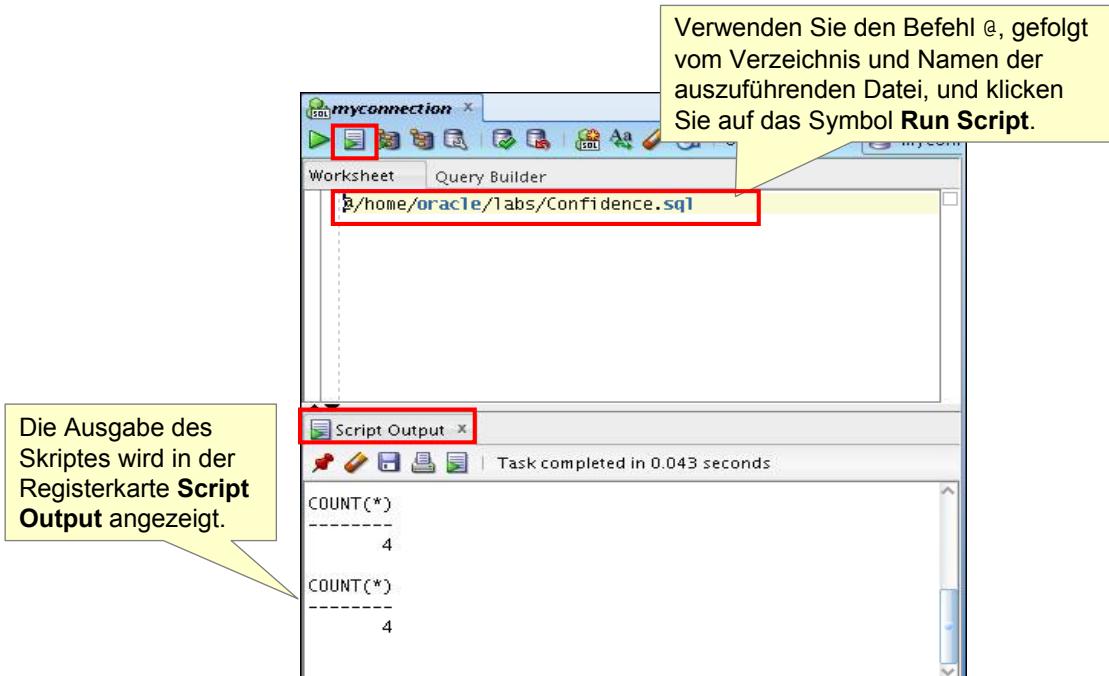
Um eine Skriptdatei zu öffnen und den Code im SQL Worksheet-Bereich anzuzeigen, gehen Sie wie folgt vor:

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

Sie können auch wie folgt vorgehen:

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

Gespeicherte Skriptdateien ausführen – 2. Methode



ORACLE

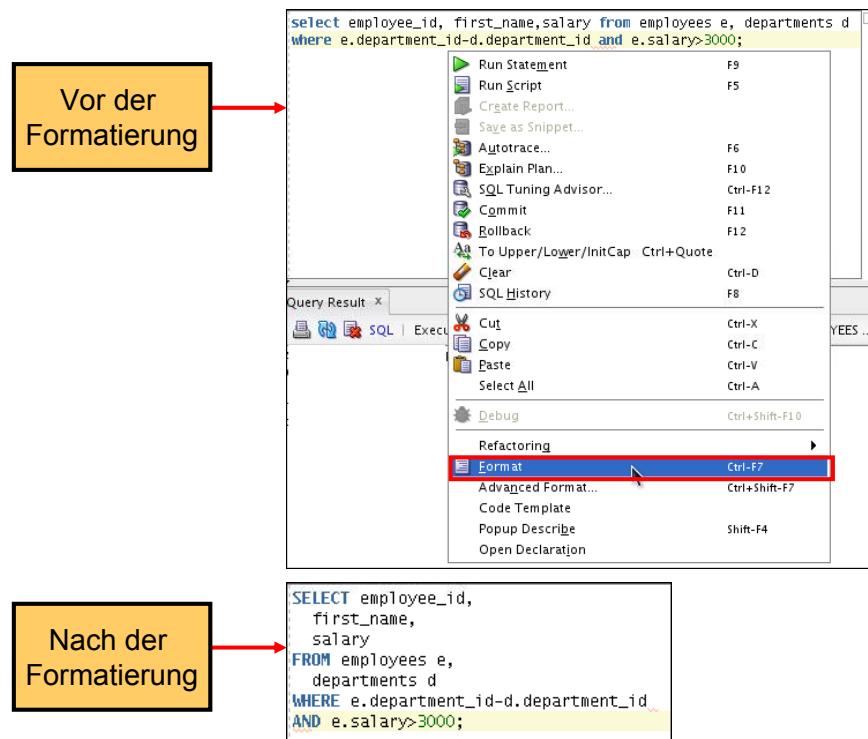
Copyright © 2014, Oracle und/oder verbundene Unternehmen. All rights reserved. Alle Rechte vorbehalten.

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

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

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

SQL-Code formatieren



ORACLE

Copyright © 2014, Oracle und/oder verbundene Unternehmen. All rights reserved. Alle Rechte vorbehalten.

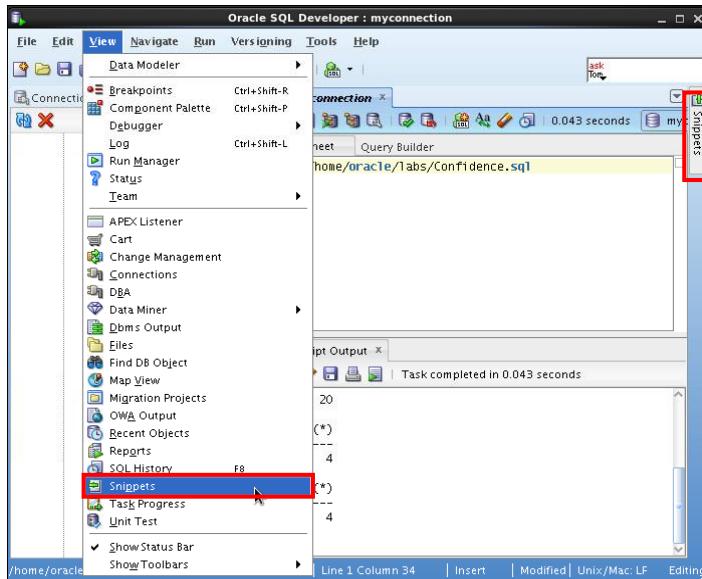
Es kann sinnvoll sein, die Einrückungen, Zeichenabstände, Groß-/Kleinschreibung und Zeilenabstände im SQL-Code übersichtlicher zu gestalten. SQL Developer bietet ein Feature zur Formatierung von SQL-Code.

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

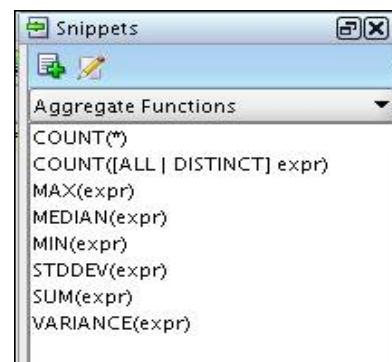
Im Beispiel auf der Folie werden die Schlüsselwörter im SQL-Code vor der Formatierung nicht in Großbuchstaben dargestellt, und die Anweisung weist nicht die richtigen Einrückungen auf. Nach der Formatierung ist der SQL-Code übersichtlicher, da die Schlüsselwörter groß geschrieben sind und die Anweisung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

Snippets

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



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



Copyright © 2014, Oracle und/oder verbundene Unternehmen. All rights reserved. Alle Rechte vorbehalten.

ORACLE

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

Um Snippets anzuzeigen, wählen Sie im Menü **View** die Option **Snippets**.

Daraufhin wird rechts das Fenster **Snippets** angezeigt. Wählen Sie in der Dropdown-Liste eine Gruppe. Über die Schaltfläche **Snippets** am rechten Fensterrand können Sie das Fenster **Snippets** einblenden, wenn es ausgeblendet wurde.

Snippets – Beispiel



ORACLE

Copyright © 2014, Oracle und/oder verbundene Unternehmen. All rights reserved. Alle Rechte vorbehalten.

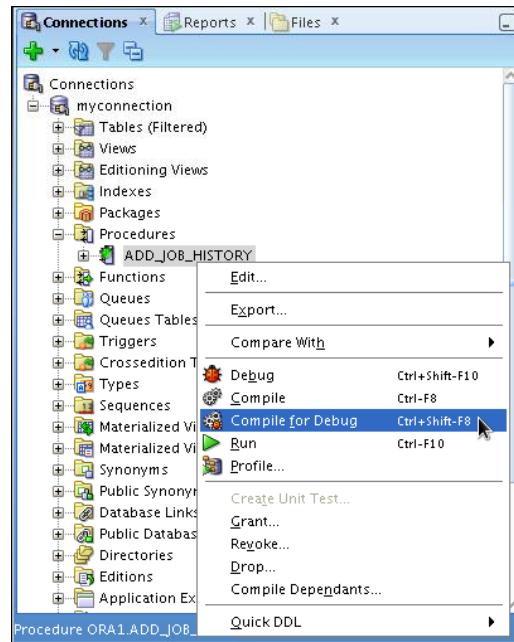
Um ein Snippet in den Code in einem SQL Worksheet oder in eine PL/SQL-Funktion oder -Prozedur einzufügen, ziehen Sie es aus dem Fenster **Snippets** an die gewünschte Stelle im Code. Danach können Sie die Syntax bearbeiten, sodass die SQL-Funktion im aktuellen Kontext gültig ist. Um eine kurze Beschreibung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

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

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

Prozeduren und Funktionen debuggen

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



ORACLE

Copyright © 2014, Oracle und/oder verbundene Unternehmen. All rights reserved. Alle Rechte vorbehalten.

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

- **Find Execution Point:** Hiermit wechseln Sie zum nächsten Ausführungspunkt.
- **Resume:** Hiermit setzen Sie die Ausführung fort.
- **Step Over:** Hiermit überspringen Sie die nächste Methode und wechseln zur nächsten Anweisung nach der Methode.
- **Step Into:** Hiermit wechseln Sie zur ersten Anweisung in der nächsten Methode.
- **Step Out:** Hiermit verlassen Sie die aktuelle Methode und wechseln zur nächsten Anweisung.
- **Step to End of Method:** Hiermit wechseln Sie zur letzten Anweisung der aktuellen Methode.
- **Pause:** Hiermit unterbrechen Sie die Ausführung, beenden sie jedoch nicht. Sie können die Ausführung wieder aufnehmen.
- **Terminate:** Hiermit brechen Sie die Ausführung ab und beenden sie. Sie können die Ausführung ab diesen Punkt nicht wieder aufnehmen. Um die Funktion oder Prozedur von Anfang an auszuführen oder zu debuggen, klicken Sie in der Symbolleiste der Registerkarte **Source** auf das Symbol **Run** oder **Debug**.
- **Garbage Collection:** Hiermit entfernen Sie ungültige Objekte aus dem Cache, um für gültige Objekte Platz zu machen, auf die häufiger zugegriffen wird.

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

Datenbankberichte

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

Owner	Name	Type	Referenced_Owner	Referenced_Name	Referenced_Type
APEX_040100 APEX	PROCEDURE APEX_040100	WAV_FLOW			PACKAGE
APEX_040100 APEX	PROCEDURE APEX_040100	WAV_FLOW_ISC			PACKAGE
APEX_040100 APEX	PROCEDURE APEX_040100	WAV_FLOW_SECURITY			PACKAGE
APEX_040100 APEX	PROCEDURE SYS	STANDARD			PACKAGE
APEX_040100 APEX	PROCEDURE SYS	SYS_STUB_FOR_PURITY_ANALYSIS			PACKAGE
APEX_040100 APEXWS	PACKAGE SYS	STANDARD			PACKAGE
APEX_040100 APEX_ADMIN	PROCEDURE APEX_040100	F			PROCEDURE
APEX_040100 APEX_ADMIN	PROCEDURE SYS	STANDARD			PACKAGE
APEX_040100 APEX_ADMIN	PROCEDURE SYS	SYS_STUB_FOR_PURITY_ANALYSIS			FUNCTION
APEX_040100 APEX_APPLICATIONS	VIEW APEX_040100	NV			TABLE
APEX_040100 APEX_APPLICATIONS	VIEW APEX_040100	WAV_FLOWS			TABLE
APEX_040100 APEX_APPLICATIONS	VIEW APEX_040100	WAV_FLOW_APPLICATION_GROUPS			TABLE
APEX_040100 APEX_APPLICATIONS	VIEW APEX_040100	WAV_FLOW_AUTHENTICATIONS			TABLE
APEX_040100 APEX_APPLICATIONS	VIEW APEX_040100	WAV_FLOW_COMPANIES			TABLE
APEX_040100 APEX_APPLICATIONS	VIEW APEX_040100	WAV_FLOW_COMPANY_SCHEMAS			TABLE
APEX_040100 APEX_APPLICATIONS	VIEW APEX_040100	WAV_FLOW_COMPUTATIONS			TABLE
APEX_040100 APEX_APPLICATIONS	VIEW APEX_040100	WAV_FLOW_ICON_BAR			TABLE
APEX_040100 APEX_APPLICATIONS	VIEW APEX_040100	WAV_FLOW_INSTALL_SCRIPTS			TABLE
APEX_040100 APEX_APPLICATIONS	VIEW APEX_040100	WAV_FLOW_ITEMS			TABLE
APEX_040100 APEX_APPLICATIONS	VIEW APEX_040100	WAV_FLOW_LANGUAGE_MAP			TABLE

ORACLE

Copyright © 2014, Oracle und/oder verbundene Unternehmen. All rights reserved. Alle Rechte vorbehalten.

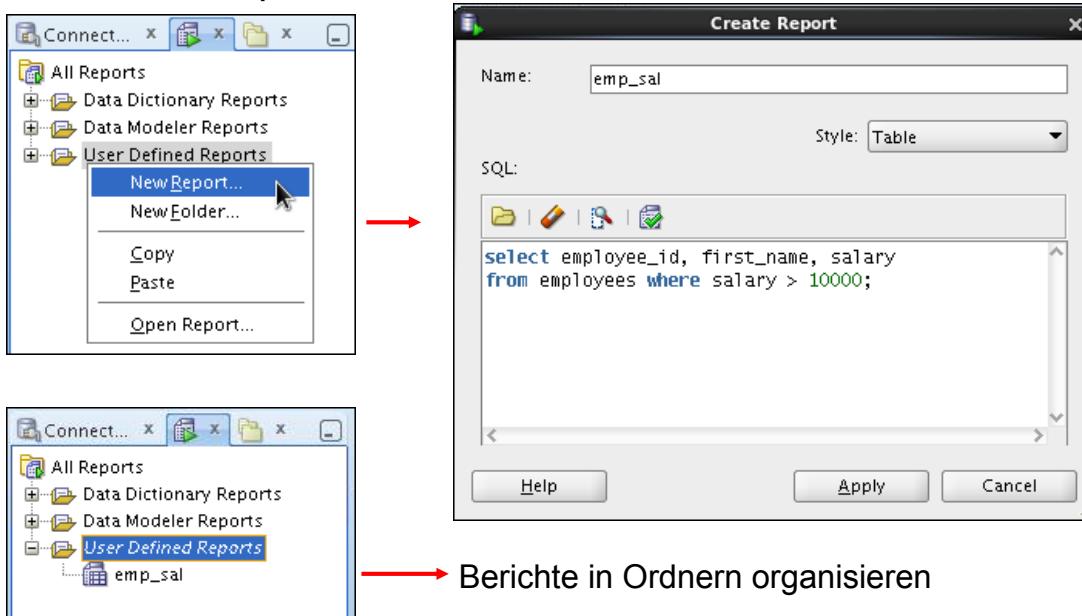
SQL Developer bietet zahlreiche Berichte zur Datenbank und ihren Objekten. Diese Berichte sind in folgende Kategorien unterteilt:

- **About Your Database**
- **Database Administration**
- **Table**
- **PL/SQL**
- **Security**
- **XML**
- **Jobs**
- **Streams**
- **All Objects**
- **Data Dictionary**
- **User Defined Reports**

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

Benutzerdefinierte Berichte erstellen

Benutzerdefinierte Berichte zur späteren Wiederverwendung erstellen und speichern



ORACLE

Copyright © 2014, Oracle und/oder verbundene Unternehmen. All rights reserved. Alle Rechte vorbehalten.

Benutzerdefinierte Berichte sind Berichte, die von SQL Developer-Benutzern erstellt werden.

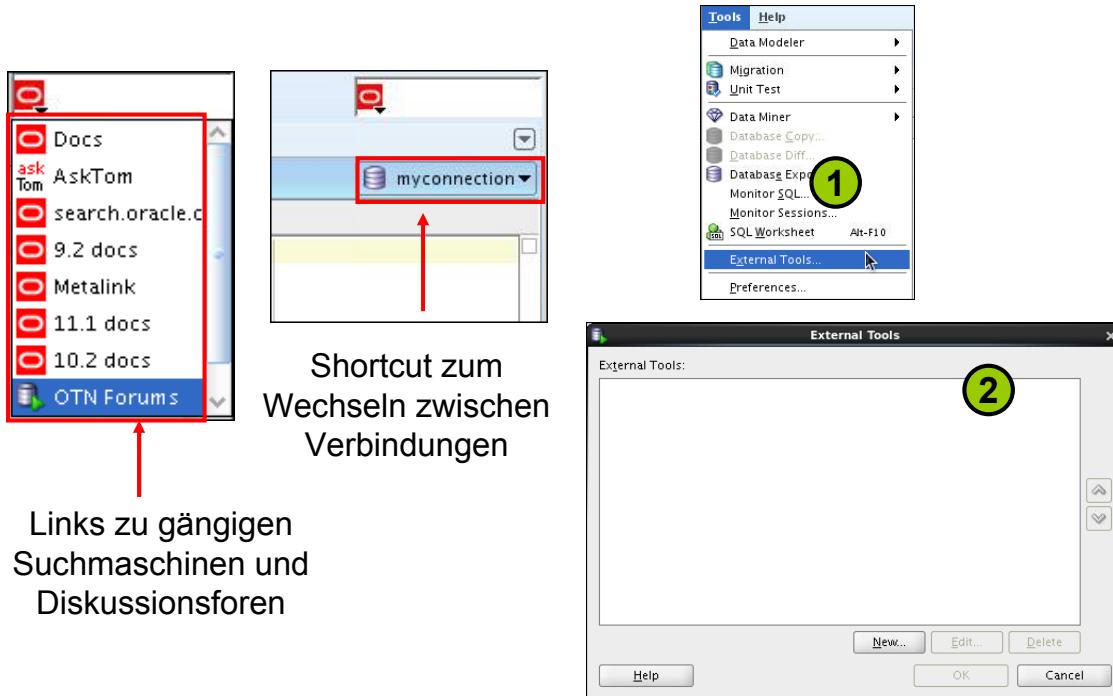
Um einen benutzerdefinierten Bericht zu erstellen, gehen Sie wie folgt vor:

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

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

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

Suchmaschinen und externe Tools



Copyright © 2014, Oracle und/oder verbundene Unternehmen. All rights reserved. Alle Rechte vorbehalten.

ORACLE

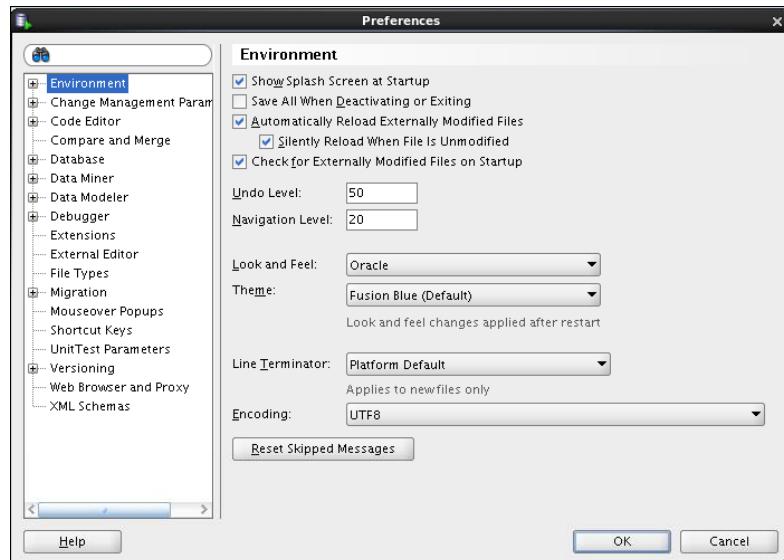
Um die Produktivität von SQL-Entwicklern zu erhöhen, wurde SQL Developer um Quicklinks zu gängigen Suchmaschinen und Diskussionsforen wie AskTom oder Google erweitert. Außerdem stehen Shortcut-Symbole zu häufig verwendeten Tools wie Notepad, Microsoft Word oder Dreamweaver zur Verfügung.

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

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

Voreinstellungen festlegen

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



ORACLE

Copyright © 2014, Oracle und/oder verbundene Unternehmen. All rights reserved. Alle Rechte vorbehalten.

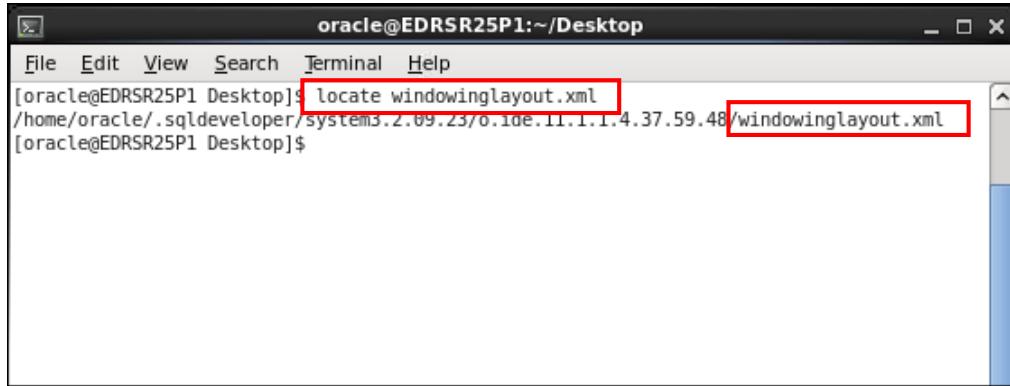
Viele Aspekte der Benutzeroberfläche und der Umgebung von SQL Developer können Sie anpassen, indem Sie die SQL Developer-Voreinstellungen entsprechend Ihren Anforderungen ändern. Um SQL Developer-Voreinstellungen zu ändern, wählen Sie im Menü **Tools** die Option **Preferences**.

Die Voreinstellungen sind in folgende Kategorien eingeteilt:

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

- **Mouseover Popups**
- **Shortcut Keys**
- **UnitTest Parameters**
- **Versioning**
- **Web Browser and Proxy**
- **XML Schemas**

SQL Developer-Layout zurücksetzen



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

A screenshot of a terminal window titled "oracle@EDRSR25P1:~/Desktop". The window contains a menu bar with File, Edit, View, Search, Terminal, and Help. Below the menu is a command prompt. The user has run the command "locate windowinglayout.xml". The output shows a single file path: "/home/oracle/.sqldeveloper/system3.2.09.23.0.10e.11.1.1.4.37.59.48/windowinglayout.xml". The entire output line is highlighted with a red box.

ORACLE

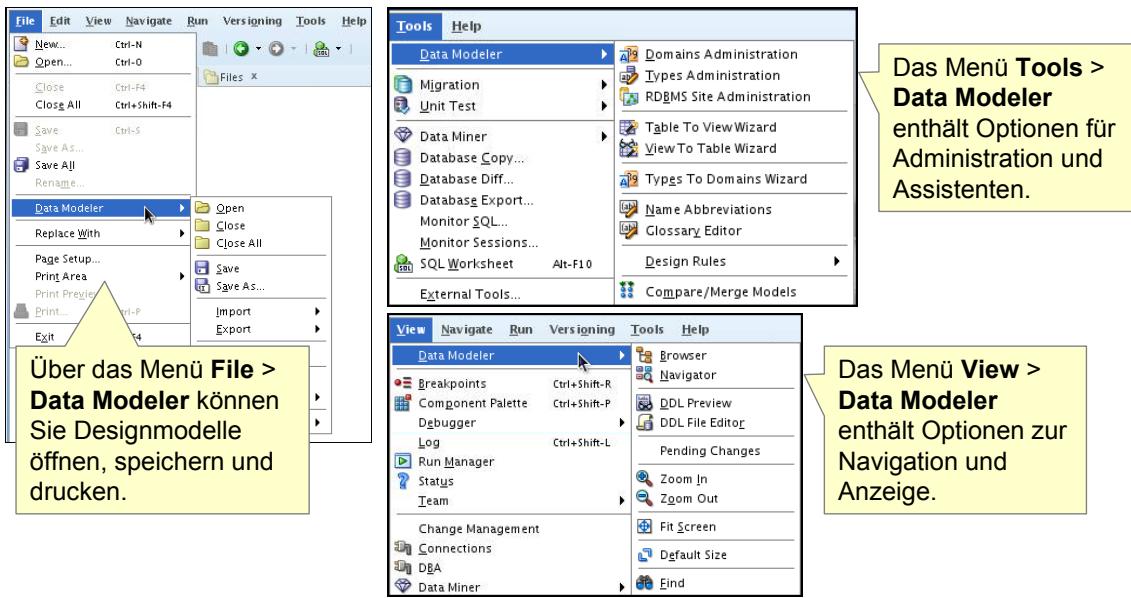
Copyright © 2014, Oracle und/oder verbundene Unternehmen. All rights reserved. Alle Rechte vorbehalten.

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

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

Data Modeler in SQL Developer

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



ORACLE

Copyright © 2014, Oracle und/oder verbundene Unternehmen. All rights reserved. Alle Rechte vorbehalten.

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

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

Um Data Modeler in einem Bereich anzuzeigen, wählen Sie im Menü **Tools** die Option **Data Modeler**. Das Untermenü **Data Modeler** im Menü **Tools** enthält zusätzliche Befehle, über die Sie beispielsweise Designregeln und Voreinstellungen festlegen können.

Zusammenfassung

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

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



Copyright © 2014, Oracle und/oder verbundene Unternehmen. All rights reserved. Alle Rechte vorbehalten.

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



ORACLE

Copyright © 2014, Oracle und/oder verbundene Unternehmen. All rights reserved. Alle Rechte vorbehalten.

Ziele

Nach Beendigung dieses Anhangs haben Sie folgende Ziele erreicht:

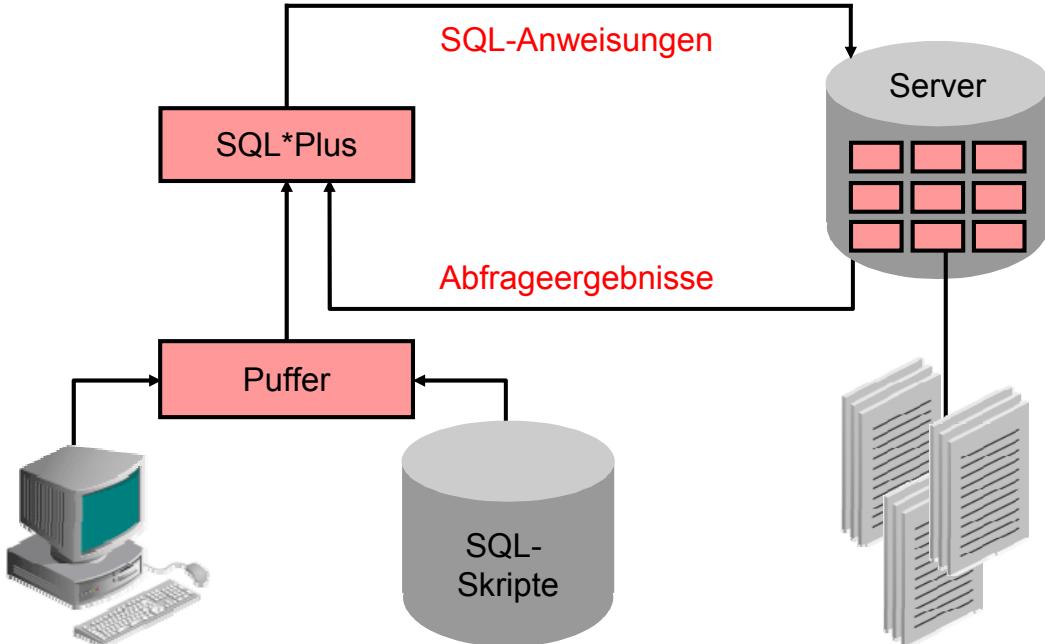
- Bei SQL*Plus anmelden
- SQL-Befehle bearbeiten
- Ausgabe mithilfe von SQL*Plus-Befehlen formatieren
- Mit Skriptdateien interagieren



Copyright © 2014, Oracle und/oder verbundene Unternehmen. All rights reserved. Alle Rechte vorbehalten.

In bestimmten Situationen bietet es sich an, SELECT-Anweisungen zu erstellen, die wiederholt eingesetzt werden können. In diesem Anhang wird die Ausführung von SQL-Anweisungen mithilfe von SQL*Plus-Befehlen behandelt. Außerdem wird beschrieben, wie Sie die Ausgabe mit SQL*Plus-Befehlen formatieren, SQL-Befehle bearbeiten und Skripte in SQL*Plus speichern.

SQL und SQL*Plus – Interaktion



ORACLE

Copyright © 2014, Oracle und/oder verbundene Unternehmen. All rights reserved. Alle Rechte vorbehalten.

SQL und SQL*Plus

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

Merkmale von SQL

- Kann von allen Benutzern verwendet werden, auch von Benutzern ohne oder mit geringen Programmiererfahrungen
- Ist eine nicht prozedurale Sprache
- Reduziert die zum Erstellen und Verwalten von Systemen benötigte Zeit
- Ist an das Englische angelehnt

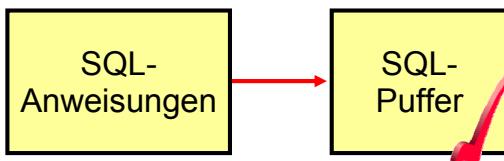
Merkmale von SQL*Plus

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

SQL-Anweisungen und SQL*Plus-Befehle – Vergleich

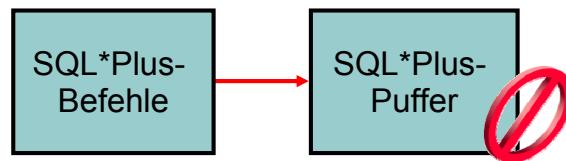
SQL

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



SQL*Plus

- ist eine Umgebung
- Proprietäres Oracle-Tool
- Schlüsselwörter dürfen abgekürzt werden.
- Befehle erlauben keine Bearbeitung von Werten in der Datenbank.



ORACLE

Copyright © 2014, Oracle und/oder verbundene Unternehmen. All rights reserved. Alle Rechte vorbehalten.

In der folgenden Tabelle werden SQL und SQL*Plus miteinander verglichen:

SQL	SQL*Plus
Programmiersprache für die Kommunikation mit dem Oracle-Server, um auf Daten zuzugreifen	Erkennt SQL-Anweisungen und sendet sie an den Server
Basiert auf Standard-SQL des ANSI (American National Standards Institute)	Proprietäre Schnittstelle zur Ausführung von SQL-Anweisungen von Oracle
Bearbeitet Daten und Tabellendefinitionen in der Datenbank	Erlaubt keine Bearbeitung von Werten in der Datenbank
Wird in einer oder mehreren Zeilen in den SQL-Puffer eingegeben	Wird Zeile für Zeile eingegeben und nicht im SQL-Puffer gespeichert
Enthält kein Fortsetzungszeichen	Verwendet den Bindestrich (-) als Fortsetzungszeichen, wenn der Befehl länger als eine Zeile ist
Kann nicht abgekürzt werden	Kann abgekürzt werden
Verwendet ein Abschlusszeichen, um Befehle unmittelbar auszuführen	Erfordert kein Abschlusszeichen; führt Befehle sofort aus
Verwendet Funktionen, um bestimmte Formatierungen vorzunehmen	Verwendet Befehle, um Daten zu formatieren

SQL*Plus – Übersicht

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

ORACLE

Copyright © 2014, Oracle und/oder verbundene Unternehmen. All rights reserved. Alle Rechte vorbehalten.

SQL*Plus

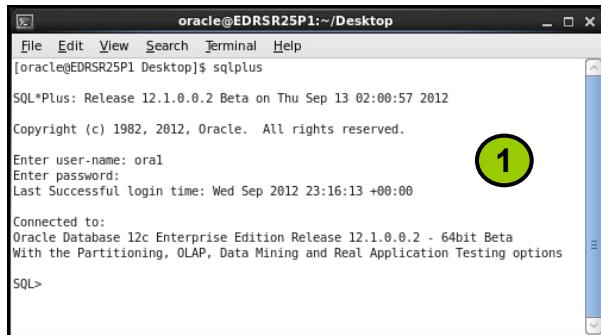
In der SQL*Plus-Umgebung können Sie folgende Aktionen ausführen:

- SQL-Anweisungen ausführen, um Daten aus der Datenbank abzurufen, zu ändern, hinzuzufügen und zu entfernen
- Abfrageergebnisse formatieren, speichern und als Berichte ausgeben sowie auf Grundlage der Ergebnisse Berechnungen ausführen
- Skriptdateien erstellen, um SQL-Anweisungen für die spätere Wiederverwendung zu speichern

SQL*Plus-Befehle lassen sich in folgende Hauptkategorien unterteilen:

Kategorie	Zweck
Umgebung	Allgemeines Verhalten von SQL-Anweisungen für die Session beeinflussen
Formatieren	Abfrageergebnisse formatieren
Dateibearbeitung	Skriptdateien speichern, laden und ausführen
Ausführung	SQL-Anweisungen aus dem SQL-Puffer an den Oracle-Server senden
Bearbeitung	SQL-Anweisungen im Puffer bearbeiten
Interaktion	Variablen erstellen und an SQL-Anweisungen übergeben, Variablenwerte ausgeben und Meldungen auf dem Bildschirm anzeigen
Verschiedenes	Bei der Datenbank anmelden, SQL*Plus-Umgebung bearbeiten und Spaltendefinitionen anzeigen

Bei SQL*Plus anmelden



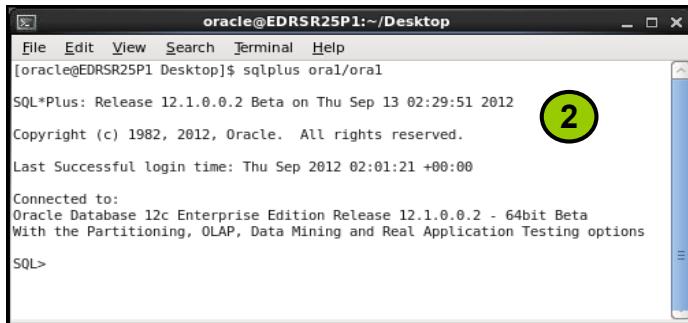
```
oracle@EDRSR25P1:~/Desktop
[oracle@EDRSR25P1 Desktop]$ sqlplus
SQL*Plus: Release 12.1.0.0.2 Beta on Thu Sep 13 02:00:57 2012
Copyright (c) 1982, 2012, Oracle. All rights reserved.

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

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

SQL>
```

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



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

Last Successful login time: Thu Sep 2012 02:01:21 +00:00

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

SQL>
```

ORACLE

Copyright © 2014, Oracle und/oder verbundene Unternehmen. All rights reserved. Alle Rechte vorbehalten.

Wie Sie SQL*Plus aufrufen, hängt von der Art des Betriebssystems ab, unter dem Sie Oracle Database ausführen.

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

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

Für die Syntax gilt:

<code>username</code>	Ihr Benutzername für die Datenbank
<code>password</code>	Ihr Kennwort für die Datenbank (sichtbar, wenn es hier eingegeben wird)
<code>@database</code>	Verbindungszeichenfolge für die Datenbank

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

Tabellenstruktur anzeigen

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

```
DESC [RIBE] tablename
```

ORACLE

Copyright © 2014, Oracle und/oder verbundene Unternehmen. All rights reserved. Alle Rechte vorbehalten.

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

Für die Syntax gilt:

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

Beschreiben Sie die Tabelle `DEPARTMENTS` mit folgendem Befehl:

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

Tabellenstruktur anzeigen

```
DESCRIBE departments
```

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

ORACLE

Copyright © 2014, Oracle und/oder verbundene Unternehmen. All rights reserved. Alle Rechte vorbehalten.

Das Beispiel auf der Folie zeigt Informationen über die Struktur der Tabelle DEPARTMENTS.

Ergebnis:

Null: Gibt an, ob eine Spalte Daten enthalten muss. (NOT NULL zeigt an, dass eine Spalte Daten enthalten muss.)

Type: Zeigt den Datentyp einer Spalte an

SQL*Plus – Bearbeitungsbefehle

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

ORACLE

Copyright © 2014, Oracle und/oder verbundene Unternehmen. All rights reserved. Alle Rechte vorbehalten.

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

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

Richtlinien

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

SQL*Plus – Bearbeitungsbefehle

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

ORACLE

Copyright © 2014, Oracle und/oder verbundene Unternehmen. All rights reserved. Alle Rechte vorbehalten.

Befehl	Beschreibung
<code>I [NPUT]</code>	Fügt eine unbestimmte Anzahl von Zeilen ein
<code>I [NPUT] <i>text</i></code>	Fügt eine Zeile mit Text (<i>text</i>) ein
<code>L [IST]</code>	Listet alle Zeilen im SQL-Puffer auf
<code>L [IST] <i>n</i></code>	Listet eine Zeile auf (angegeben durch <i>n</i>)
<code>L [IST] <i>m n</i></code>	Listet einen Bereich von Zeilen auf (<i>m</i> bis einschließlich <i>n</i>)
<code>R [UN]</code>	Zeigt die aktuell im Puffer befindliche SQL-Anweisung an und führt sie aus
<code><i>n</i></code>	Macht <i>n</i> zur aktuellen Zeile
<code><i>n text</i></code>	Ersetzt Zeile <i>n</i> durch Text (<i>text</i>)
<code>0 <i>text</i></code>	Fügt eine Zeile vor Zeile 1 ein

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

LIST, n und APPEND

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

```
1
1* SELECT last_name
```

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

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

ORACLE

Copyright © 2014, Oracle und/oder verbundene Unternehmen. All rights reserved. Alle Rechte vorbehalten.

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

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

Befehl CHANGE

```
LIST
```

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

```
c/employees/departments
```

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

```
LIST
```

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

ORACLE®

Copyright © 2014, Oracle und/oder verbundene Unternehmen. All rights reserved. Alle Rechte vorbehalten.

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

SQL*Plus – Dateibefehle

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

ORACLE

Copyright © 2014, Oracle und/oder verbundene Unternehmen. All rights reserved. Alle Rechte vorbehalten.

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

Befehl	Beschreibung
<code>SAV[E] filename [.ext] [REP[LACE]APP[END]]</code>	Speichert den aktuellen Inhalt des SQL-Puffers in einer Datei. Verwenden Sie APPEND, um den Pufferinhalt einer vorhandenen Datei hinzuzufügen, oder REPLACE, um eine vorhandene Datei zu ersetzen. Die Standarderweiterung lautet .sql.
<code>GET filename [.ext]</code>	Schreibt den Inhalt einer zuvor gespeicherten Datei in den SQL-Puffer. Die Standarderweiterung für den Dateinamen lautet .sql.
<code>STA[RT] filename [.ext]</code>	Führt eine zuvor gespeicherte Befehlsdatei aus
<code>@ filename</code>	Führt eine zuvor gespeicherte Befehlsdatei aus (identisch mit START)
<code>ED[IT]</code>	Ruft den Editor auf und speichert den Pufferinhalt in einer Datei namens afiedt.buf
<code>ED[IT] [filename[.ext]]</code>	Ruft den Editor auf, um den Inhalt einer gespeicherten Datei zu bearbeiten
<code>SPO[OL] [filename[.ext]] OFF OUT</code>	Speichert die Abfrageergebnisse in einer Datei. Mit OFF wird die Spooldatei geschlossen. Mit OUT wird die Spooldatei geschlossen, und die Dateiergebnisse werden an den Drucker gesendet.
<code>EXIT</code>	Beendet SQL*Plus

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

```
LIST
```

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

```
SAVE my_query
```

```
Created file my_query
```

```
START my_query
```

LAST_NAME	MANAGER_ID	DEPARTMENT_ID
King		90
Kochhar	100	90
...		
107 rows selected.		

ORACLE®

Copyright © 2014, Oracle und/oder verbundene Unternehmen. All rights reserved. Alle Rechte vorbehalten.

SAVE

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

START

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

```
@my_query
```

Befehl SERVEROUTPUT

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

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



Copyright © 2014, Oracle und/oder verbundene Unternehmen. All rights reserved. Alle Rechte vorbehalten.

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

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

```
SET SERVEROUTPUT ON
```

Hinweis

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

SQL*Plus-Befehl SPOOL

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

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

ORACLE

Copyright © 2014, Oracle und/oder verbundene Unternehmen. All rights reserved. Alle Rechte vorbehalten.

Der Befehl SPOOL speichert die Abfrageergebnisse in einer Datei oder sendet die Datei optional an einen Drucker. Der Befehl SPOOL wurde erweitert. Sie können eine Datei jetzt mit einem Anhang versehen oder eine vorhandene Datei ersetzen. Vorher konnten Sie mit SPOOL lediglich Dateien erstellen (und ersetzen). REPLACE ist der Standardwert.

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

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

Befehl AUTOTRACE

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

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

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

ORACLE

Copyright © 2014, Oracle und/oder verbundene Unternehmen. All rights reserved. Alle Rechte vorbehalten.

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

Hinweise

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

Zusammenfassung

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

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



Copyright © 2014, Oracle und/oder verbundene Unternehmen. All rights reserved. Alle Rechte vorbehalten.

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

Häufig verwendete SQL-Befehle

ORACLE

Copyright © 2014, Oracle und/oder verbundene Unternehmen. All rights reserved. Alle Rechte vorbehalten.

Ziele

Nach Ablauf dieses Anhangs haben Sie folgende Ziele erreicht:

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

ORACLE

Copyright © 2014, Oracle und/oder verbundene Unternehmen. All rights reserved. Alle Rechte vorbehalten.

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

Einfache SELECT-Anweisungen

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

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

ORACLE

Copyright © 2014, Oracle und/oder verbundene Unternehmen. All rights reserved. Alle Rechte vorbehalten.

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

- Eine SELECT-Klausel zur Angabe der anzuzeigenden Spalten
- Eine FROM-Klausel zur Angabe der Tabelle mit den in der SELECT-Klausel aufgeführten Spalten

Für die Syntax gilt:

SELECT	Liste aus einer oder mehreren Spalten
*	Wählt alle Spalten
DISTINCT	Unterdrückt doppelte Spalten
<i>column / expression</i>	Wählt die angegebene Spalte oder den Ausdruck
<i>alias</i>	Gibt den gewählten Spalten verschiedene Überschriften
FROM <i>table</i>	Gibt an, welche Tabelle die Spalten enthält

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

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

SELECT-Anweisungen

- Alle Spalten wählen:

```
SELECT *  
FROM job_history;
```

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

- Bestimmte Spalten wählen:

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

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

...

ORACLE

Copyright © 2014, Oracle und/oder verbundene Unternehmen. All rights reserved. Alle Rechte vorbehalten.

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

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

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

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

WHERE-Klauseln

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

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

- Beispiel:

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

ORACLE

Copyright © 2014, Oracle und/oder verbundene Unternehmen. All rights reserved. Alle Rechte vorbehalten.

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

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

Beispiel:

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

ORDER BY-Klauseln

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

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

- Beispiel:

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

ORACLE

Copyright © 2014, Oracle und/oder verbundene Unternehmen. All rights reserved. Alle Rechte vorbehalten.

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

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

GROUP BY-Klauseln

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

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

- Beispiel:

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

ORACLE

Copyright © 2014, Oracle und/oder verbundene Unternehmen. All rights reserved. Alle Rechte vorbehalten.

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

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

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

Data Definition Language (DDL)

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

ORACLE

Copyright © 2014, Oracle und/oder verbundene Unternehmen. All rights reserved. Alle Rechte vorbehalten.

Mithilfe von DDL-Anweisungen können Sie die Attribute eines Objekts ändern, ohne gleichzeitig auch die Anwendungen ändern zu müssen, die auf das Objekt zugreifen. Außerdem lässt sich mit DDL-Anweisungen die Struktur von Objekten ändern, während Benutzer in der Datenbank arbeiten. Häufige Anwendungsbereiche von DML-Anweisungen:

- Schemaobjekte und andere Datenbankstrukturen erstellen, ändern und löschen, einschließlich der Datenbank selbst und ihrer Benutzer
- Alle Daten in Schemaobjekten löschen, ohne die Struktur dieser Objekte zu entfernen
- Berechtigungen und Rollen erteilen und entziehen

Oracle Database schreibt die aktuelle Transaktion vor und nach jeder DDL-Anweisung implizit fest.

CREATE TABLE-Anweisungen

- Mit der Anweisung CREATE TABLE erstellen Sie Tabellen in der Datenbank.
- Syntax:

```
CREATE TABLE tablename (
  {column-definition | Table-level constraint}
  [ , {column-definition | Table-level constraint} ] * )
```

- Beispiel:

```
CREATE TABLE teach_dept (
  department_id NUMBER(3) PRIMARY KEY,
  department_name VARCHAR2(10));
```

ORACLE

Copyright © 2014, Oracle und/oder verbundene Unternehmen. All rights reserved. Alle Rechte vorbehalten.

Mit der Anweisung CREATE TABLE erstellen Sie Tabellen in der Datenbank. Dazu müssen Sie über die Berechtigung CREATE TABLE und einen Speicherbereich verfügen, in dem Sie Objekte erstellen können.

Die Eigentümer der Tabelle und der Datenbank erhalten nach der Erstellung einer Tabelle automatisch folgende Berechtigungen für die Tabelle:

- INSERT
- SELECT
- REFERENCES
- ALTER
- UPDATE

Die Eigentümer der Tabelle und der Datenbank können anderen Benutzern die vorstehenden Berechtigungen erteilen.

ALTER TABLE-Anweisungen

- Mit der Anweisung ALTER TABLE ändern Sie die Definition einer vorhandenen Tabelle in der Datenbank.
- 1. Beispiel:

```
ALTER TABLE teach_dept  
ADD location_id NUMBER NOT NULL;
```

- 2. Beispiel:

```
ALTER TABLE teach_dept  
MODIFY department_name VARCHAR2(30) NOT NULL;
```

ORACLE

Copyright © 2014, Oracle und/oder verbundene Unternehmen. All rights reserved. Alle Rechte vorbehalten.

Mithilfe der Anweisung ALTER TABLE können Sie Änderungen an einer vorhandenen Tabelle vornehmen.

Sie können:

- Spalten zu Tabellen hinzufügen
- Constraints zu Tabellen hinzufügen
- vorhandene Spaltendefinitionen ändern
- Spalten aus Tabellen löschen
- vorhandene Constraints aus Tabellen löschen
- die Breite von VARCHAR- und CHAR-Spalten vergrößern
- Tabellen in den schreibgeschützten Status versetzen

Im 1. Beispiel auf der Folie wird die Tabelle TEACH_DEPT um die neue Spalte LOCATION_ID erweitert.

Im 2. Beispiel wird die vorhandene Spalte DEPARTMENT_NAME von VARCHAR2(10) in VARCHAR2(30) geändert, und es wird ein NOT NULL Constraint hinzugefügt.

DROP TABLE-Anweisungen

- Mit der Anweisung `DROP TABLE` löschen Sie die Tabelle mit sämtlichen darin enthaltenen Daten aus der Datenbank.
- Beispiel:

```
DROP TABLE teach_dept;
```

- `DROP TABLE` mit der `PURGE`-Klausel löscht die Tabelle und gibt den damit verbundenen Speicherplatz frei.

```
DROP TABLE teach_dept PURGE;
```

- Bei Verwendung der Klausel `CASCADE CONSTRAINTS` werden alle Constraints zur referenziellen Integrität aus der Tabelle gelöscht.

```
DROP TABLE teach_dept CASCADE CONSTRAINTS;
```

ORACLE

Copyright © 2014, Oracle und/oder verbundene Unternehmen. All rights reserved. Alle Rechte vorbehalten.

Mithilfe der Anweisung `DROP TABLE` können Sie eine Tabelle einschließlich ihres Inhalts aus der Datenbank löschen und in den Papierkorb verschieben. Beim Löschen einer Tabelle werden die abhängigen Objekte invalidiert und Objektberechtigungen für die Tabelle entfernt.

Um den zugewiesenen Speicherplatz für die Tabelle wieder für den Tablespace freizugeben, verwenden Sie die Anweisung `DROP TABLE` mit der `PURGE`-Klausel. Sie können `DROP TABLE`-Anweisungen mit der `PURGE`-Klausel nicht zurücksetzen und auch die Tabelle nicht wiederherstellen, wenn sie mit der `PURGE`-Klausel gelöscht wurde.

Mit der Klausel `CASCADE CONSTRAINTS` können Sie die Referenz auf den Primärschlüssel und die eindeutigen Schlüssel in der gelöschten Tabelle aufheben.

GRANT-Anweisungen

- Mit GRANT-Anweisungen werden Berechtigungen zur Ausführung folgender Vorgänge erteilt:
 - Daten einfügen oder löschen
 - Fremdschlüsselreferenz auf die benannte Tabelle oder eine Teilmenge von Spalten aus einer Tabelle erstellen
 - Daten, Views oder eine Teilmenge von Spalten aus einer Tabelle wählen
 - Trigger für Tabellen erstellen
 - Angegebene Funktionen oder Prozeduren ausführen
- Beispiel:

```
GRANT SELECT any table to PUBLIC;
```

ORACLE

Copyright © 2014, Oracle und/oder verbundene Unternehmen. All rights reserved. Alle Rechte vorbehalten.

Mithilfe von GRANT-Anweisungen können Sie:

- bestimmten Benutzern oder Rollen (beziehungsweise allen Benutzern) Berechtigungen zur Ausführung von Aktionen an Datenbankobjekten erteilen
- Benutzern, PUBLIC oder anderen Rollen eine Rolle erteilen

Vergewissern Sie sich vor dem Absetzen einer GRANT-Anweisung, dass für die Autorisierungseigenschaft derby.database.sql der Wert True eingestellt ist. Mit dieser Eigenschaft wird der SQL-Autorisierungsmodus aktiviert. Sie können Berechtigungen für ein Objekt erteilen, wenn Sie der Eigentümer der Datenbank sind.

Um allen Benutzern Berechtigungen zu erteilen, verwenden Sie das Schlüsselwort PUBLIC. Ist PUBLIC angegeben, gelten die Berechtigungen oder Rollen für alle aktuellen und künftigen Benutzer.

Typen von Berechtigungen

Mit GRANT-Anweisungen werden folgende Berechtigungen erteilt:

- ALL PRIVILEGES
- DELETE
- INSERT
- REFERENCES
- SELECT
- UPDATE



Copyright © 2014, Oracle und/oder verbundene Unternehmen. All rights reserved. Alle Rechte vorbehalten.

Oracle Database bietet verschiedene Berechtigungstypen, um Benutzern und Rollen Berechtigungen zu erteilen:

- Mit dem Berechtigungstyp ALL PRIVILEGES erteilen Sie Benutzern oder Rollen sämtliche Berechtigungen für die angegebene Tabelle.
- Mit dem Berechtigungstyp DELETE erteilen Sie die Berechtigung, Zeilen aus der angegebenen Tabelle zu löschen.
- Mit dem Berechtigungstyp INSERT erteilen Sie die Berechtigung, Zeilen in die angegebene Tabelle einzufügen.
- Mit dem Berechtigungstyp REFERENCES erteilen Sie die Berechtigung zur Erstellung einer Fremdschlüsselreferenz auf die angegebene Tabelle.
- Mit dem Berechtigungstyp SELECT erteilen Sie die Berechtigung zur Ausführung von SELECT-Anweisungen für eine Tabelle oder View.
- Mit dem Berechtigungstyp UPDATE erteilen Sie die Berechtigung zur Verwendung von UPDATE-Anweisungen für die angegebene Tabelle.

REVOKE-Anweisungen

- Mit REVOKE-Anweisungen entziehen Sie Benutzern Berechtigungen zur Ausführung von Aktionen an Datenbankobjekten.
- Einem Benutzer eine *Systemberechtigung* entziehen:

```
REVOKE DROP ANY TABLE  
FROM hr;
```

- Einem Benutzer eine *Rolle* entziehen:

```
REVOKE dw_manager  
FROM sh;
```

ORACLE

Copyright © 2014, Oracle und/oder verbundene Unternehmen. All rights reserved. Alle Rechte vorbehalten.

Mit REVOKE-Anweisungen entziehen Sie einem bestimmten Benutzer (oder mehreren Benutzern) beziehungsweise einer Rolle die Berechtigungen zur Ausführung von Aktionen an Datenbankobjekten. Folgende Vorgänge sind möglich:

- Eine Rolle einem Benutzer, PUBLIC oder einer anderen Rolle entziehen
- Berechtigungen für ein Objekt entziehen, sofern Sie der Eigentümer des Objekts oder der Datenbank sind

Hinweis: Um eine Rolle oder eine Systemberechtigung entziehen zu können, benötigen Sie die Berechtigung mit der ADMIN OPTION.

TRUNCATE TABLE-Anweisungen

- Mithilfe von TRUNCATE TABLE-Anweisungen löschen Sie alle Zeilen einer Tabelle.
- Beispiel:

```
TRUNCATE TABLE employees_demo;
```

- Oracle Database führt standardmäßig folgende Aufgaben durch:
 - Durch die gelöschten Zeilen belegten Speicherplatz freigeben
 - Speicherparameter NEXT auf die Größe des letzten Extents einstellen, das durch den Leerungsprozess aus dem Segment entfernt wurde

ORACLE

Copyright © 2014, Oracle und/oder verbundene Unternehmen. All rights reserved. Alle Rechte vorbehalten.

Mit TRUNCATE TABLE-Anweisungen löschen Sie alle Zeilen aus einer angegebenen Tabelle. Zeilen mit einer TRUNCATE TABLE-Anweisung zu entfernen, kann effizienter sein als die Tabelle zu löschen und neu zu erstellen. Beim Löschen und erneuten Erstellen einer Tabelle:

- werden die abhängigen Objekte der Tabelle invalidiert
- müssen Sie Objektberechtigungen neu erteilen
- müssen Sie Indizes, Integritäts-Constraints und Trigger neu erstellen
- müssen Sie Speicherparameter neu festlegen

Bei Verwendung von TRUNCATE TABLE-Anweisungen können Sie auf diese Arbeitsschritte verzichten.

Hinweis: TRUNCATE TABLE-Anweisungen können nicht zurückgesetzt werden.

Data Manipulation Language (DML)

- Mit DML-Anweisungen fragen Sie Daten in vorhandenen Schemaobjekten ab oder bearbeiten diese Daten.
- Eine DML-Anweisung wird ausgeführt, wenn Sie:
 - einer Tabelle mit einer `INSERT`-Anweisung neue Zeilen hinzufügen
 - mit einer `UPDATE`-Anweisung vorhandene Zeilen in einer Tabelle ändern
 - mit einer `DELETE`-Anweisung vorhandene Zeilen aus einer Tabelle löschen
- Eine *Transaktion* besteht aus einer Gruppe von DML-Anweisungen, die eine logische Arbeitseinheit bilden.

ORACLE

Copyright © 2014, Oracle und/oder verbundene Unternehmen. All rights reserved. Alle Rechte vorbehalten.

Mithilfe von Data Manipulation Language-(DML-)Anweisungen können Sie den Inhalt vorhandener Schemaobjekte abfragen oder ändern. Häufige Anwendungsbereiche von DML-Anweisungen:

- Einer Tabelle oder View neue Datenzeilen hinzufügen (durch Angabe einer Liste von Spaltenwerten oder durch Auswahl und Bearbeitung vorhandener Daten mithilfe einer Unterabfrage)
- Spaltenwerte in den vorhandenen Zeilen einer Tabelle oder View ändern
- Zeilen aus Tabellen oder Views entfernen

Eine Gruppe von DML-Anweisungen, die eine logische Arbeitseinheit bilden, wird als Transaktion bezeichnet. Im Gegensatz zu DDL-Anweisungen schreiben DML-Anweisungen die aktuelle Transaktion nicht implizit fest.

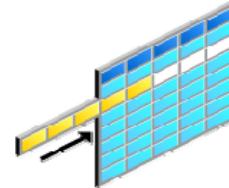
INSERT-Anweisungen

- Mithilfe von INSERT-Anweisungen fügen Sie einer Tabelle neue Zeilen hinzu.
- Syntax:

```
INSERT INTO table [(column [, column...])]  
VALUES      (value [, value...]);
```

- Beispiel:

```
INSERT INTO departments  
VALUES      (200, 'Development', 104, 1400);  
1 rows inserted.
```



ORACLE

Copyright © 2014, Oracle und/oder verbundene Unternehmen. All rights reserved. Alle Rechte vorbehalten.

Mit einer INSERT-Anweisung fügen Sie neue Tabellenzeilen hinzu. Achten Sie darauf, neue Zeilen mit Werten für alle Spalten einzufügen und die Werte in der standardmäßigen Reihenfolge der Spalten in der Tabelle aufzulisten. Optional können Sie die Spalten auch in der INSERT-Anweisung auflisten.

Beispiel:

```
INSERT INTO job_history (employee_id, start_date, end_date, job_id)  
VALUES (120, '25-JUL-06', '12-FEB-08', 'AC_ACCOUNT');
```

Mit der auf der Folie gezeigten Syntax können Sie jeweils eine einzelne Zeile einfügen. Das Schlüsselwort VALUES weist den entsprechenden Spalten in der Spaltenliste die Werte von Ausdrücken zu.

UPDATE-Anweisungen – Syntax

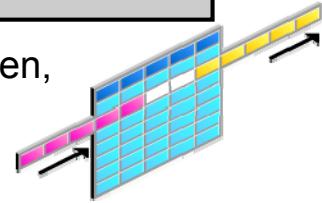
- Mit einer UPDATE-Anweisung ändern Sie die vorhandenen Zeilen in einer Tabelle.
- Falls erforderlich, können mehrere Zeilen gleichzeitig aktualisiert werden.

```
UPDATE      table
SET         column = value [, column = value, ...]
[WHERE      condition];
```

- Beispiel:

```
UPDATE      copy_emp
SET
22 rows updated
```

- Um einen Spaltenwert mit NULL zu aktualisieren, geben Sie SET *column_name*= NULL an.



ORACLE

Copyright © 2014, Oracle und/oder verbundene Unternehmen. All rights reserved. Alle Rechte vorbehalten.

Mit einer UPDATE-Anweisung ändern Sie die vorhandenen Werte in einer Tabelle. Um den Aktualisierungsvorgang zu prüfen, zeigen Sie über eine Tabellenabfrage die aktualisierten Zeilen an. Um nur bestimmte Zeilen zu ändern, geben Sie die WHERE-Klausel an.

Beispiel:

```
UPDATE employees
SET      salary = 17500
WHERE    employee_id = 102;
```

In der Regel identifizieren Sie die zu aktualisierende Zeile in der WHERE-Klausel mithilfe der Primärschlüsselspalte. Beispiel: Sie möchten eine bestimmte Zeile in der Tabelle EMPLOYEES aktualisieren. Um die Zeile anzugeben, verwenden Sie *employee_id* anstelle von *employee_name*, da eventuell mehrere Mitarbeiter denselben Namen haben können.

Hinweis: Das Schlüsselwort *condition* besteht in der Regel aus Spaltennamen, Ausdrücken, Konstanten, Unterabfragen und Vergleichsoperatoren.

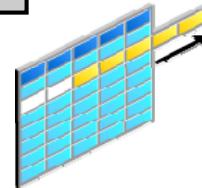
DELETE-Anweisungen

- Mit einer DELETE-Anweisung löschen Sie vorhandene Zeilen aus einer Tabelle.
- Syntax:

```
DELETE      [ FROM ]      table  
[ WHERE      condition ] ;
```

- Um bestimmte Zeilen aus einer Tabelle zu löschen, geben Sie die DELETE-Anweisung mit der WHERE-Klausel an.

```
DELETE FROM departments  
WHERE department_name = 'Finance';  
1 rows deleted
```



ORACLE

Copyright © 2014, Oracle und/oder verbundene Unternehmen. All rights reserved. Alle Rechte vorbehalten.

Mit einer DELETE-Anweisung entfernen Sie vorhandene Zeilen aus einer Tabelle. Um bestimmte Zeilen auf Basis einer vorgegebenen Bedingung zu löschen, geben Sie die WHERE-Klausel an. Die zu löschen Zeilen werden über das Schlüsselwort `condition` bestimmt, das Spaltennamen, Ausdrücke, Konstanten, Unterabfragen und Vergleichsoperatoren enthalten kann.

Im ersten Beispiel auf der Folie wird die Finanzabteilung aus der Tabelle DEPARTMENTS gelöscht. Um den Löschvorgang zu prüfen, fragen Sie die Tabelle mit einer SELECT-Anweisung ab.

```
SELECT *  
FROM   departments  
WHERE  department_name = 'Finance';
```

Wenn Sie die WHERE-Klausel weglassen, werden alle Zeilen aus der Tabelle gelöscht. Beispiel:

```
DELETE FROM copy_emp;
```

Im vorstehenden Beispiel werden alle Zeilen aus der Tabelle COPY_EMP gelöscht.

Anweisungen zur Transaktionskontrolle

- Anweisungen zur Transaktionskontrolle dienen zur Verwaltung der über DML-Anweisungen vorgenommenen Änderungen.
- Die DML-Anweisungen werden zu Transaktionen zusammengefasst.
- Anweisungen zur Transaktionskontrolle:
 - COMMIT
 - ROLLBACK
 - SAVEPOINT

ORACLE

Copyright © 2014, Oracle und/oder verbundene Unternehmen. All rights reserved. Alle Rechte vorbehalten.

Eine Transaktion besteht aus einer Folge von SQL-Anweisungen, die von Oracle Database als einzelne Einheit behandelt werden. Anweisungen zur Transaktionskontrolle werden in einer Datenbank verwendet, um die über DML-Anweisungen vorgenommenen Änderungen zu verwalten und diese Anweisungen zu Transaktionen zusammenzufassen.

Jeder Transaktion wird eine eindeutige `transaction_id` zugewiesen. Die in der Transaktion enthaltenen SQL-Anweisungen werden entweder allesamt festgeschrieben (und in der Datenbank angewendet) oder allesamt zurückgesetzt (und in der Datenbank rückgängig gemacht).

COMMIT-Anweisungen

- Mit COMMIT-Anweisungen können Sie:
 - die während der aktuellen Transaktion vorgenommenen Änderungen endgültig speichern
 - alle Savepoints in der Transaktion löschen
 - Transaktionssperren aufheben
- Beispiel:

```
INSERT INTO departments
VALUES      (201, 'Engineering', 106, 1400);
COMMIT;

1 rows inserted.
committed.
```

ORACLE

Copyright © 2014, Oracle und/oder verbundene Unternehmen. All rights reserved. Alle Rechte vorbehalten.

Wenn Sie die aktuelle Transaktion mit einer COMMIT-Anweisung beenden, werden alle noch nicht gespeicherten Datenänderungen dauerhaft festgeschrieben. Dabei werden alle Zeilen- und Tabellensperren aufgehoben sowie alle seit dem letzten Commit- oder Rollback-Vorgang eventuell festgelegten Savepoints gelöscht. Die mit COMMIT-Anweisungen vorgenommenen Änderungen sind für alle Benutzer sichtbar.

Oracle empfiehlt, jede Transaktion in Ihren Anwendungsprogrammen (einschließlich der letzten Transaktion) explizit mit COMMIT oder ROLLBACK zu beenden, bevor Sie sich bei Oracle Database abmelden. Wenn Sie die Transaktion nicht explizit festschreiben und das Programm anschließend nicht ordnungsgemäß beendet werden kann, wird andernfalls die letzte nicht festgeschriebene Transaktion automatisch zurückgesetzt.

Hinweis: Oracle Database setzt vor und nach DDL-Anweisungen implizite COMMIT-Anweisungen ab.

ROLLBACK-Anweisungen

- Mit einer ROLLBACK-Anweisung machen Sie die während der aktuellen Transaktion an der Datenbank vorgenommenen Änderungen rückgängig.
- Um den Teil der Transaktion nach dem Savepoint rückgängig zu machen, verwenden Sie die Klausel TO SAVEPOINT.
- Beispiel:

```
UPDATE      employees
SET          salary = 7000
WHERE        last_name = 'Ernst';
SAVEPOINT   Ernst_sal;

UPDATE      employees
SET          salary = 12000
WHERE        last_name = 'Mourgos';

ROLLBACK TO SAVEPOINT Ernst_sal;
```

ORACLE

Copyright © 2014, Oracle und/oder verbundene Unternehmen. All rights reserved. Alle Rechte vorbehalten.

Mit einer ROLLBACK-Anweisung werden die in der aktuellen Transaktion durchgeführten Aktionen zurückgesetzt. Um die aktuelle Transaktion zurückzusetzen, sind keine Berechtigungen erforderlich.

Mit ROLLBACK und der Klausel TO SAVEPOINT führen Sie folgende Vorgänge durch:

- Nur den Teil der Transaktion zurücksetzen, der nach dem Savepoint liegt
- Alle nach diesem Savepoint erstellten Savepoints löschen. Der benannte Savepoint bleibt erhalten, sodass Sie mehrere Rollbacks bis zu diesem Savepoint vornehmen können.

Mit ROLLBACK ohne Angabe der Klausel TO SAVEPOINT führen Sie folgende Vorgänge durch:

- Transaktion beenden
- Alle während der aktuellen Transaktion vorgenommenen Änderungen rückgängig machen
- Alle Savepoints in der Transaktion löschen

SAVEPOINT-Anweisungen

- Mit einer SAVEPOINT-Anweisung benennen und markieren Sie den aktuellen Verarbeitungspunkt einer Transaktion.
- Namen für jeden Savepoint angeben
- Savepoints innerhalb einer Transaktion eindeutig benennen, um Überschreibungen zu vermeiden
- Syntax:

```
SAVEPOINT savepoint;
```

ORACLE

Copyright © 2014, Oracle und/oder verbundene Unternehmen. All rights reserved. Alle Rechte vorbehalten.

Mit einer SAVEPOINT-Anweisung kennzeichnen Sie einen Punkt in der Transaktion, bis zu dem Sie spätere Rollbacks vornehmen können. Jeder Savepoint muss über einen eindeutigen Namen verfügen. Wenn Sie im Anschluss einen zweiten Savepoint mit demselben Namen erstellen, wird der frühere Savepoint gelöscht.

Nach der Erstellung eines Savepoints können Sie mit der Verarbeitung fortfahren, Ihre Arbeit festschreiben, die gesamte Transaktion zurückschreiben oder alle Aktionen nach dem Savepoint zurücksetzen.

Bei einem einfachen Rollback- oder Commit-Vorgang werden alle Savepoints gelöscht. Bei einem Rollback bis zu einem Savepoint werden alle späteren Savepoints gelöscht. Der Savepoint, der für den Rollback-Vorgang verwendet wurde, bleibt erhalten.

Wenn Savepoint-Namen innerhalb einer Transaktion mehrfach verwendet werden, verschiebt Oracle Database den Savepoint von der bisherigen an die aktuelle Position in der Transaktion und setzt damit den älteren Savepoint außer Kraft.

Joins

Mithilfe von Joins fragen Sie Daten aus mehreren Tabellen ab:

```
SELECT      table1.column, table2.column  
FROM        table1, table2  
WHERE       table1.column1 = table2.column2;
```

- Join-Bedingungen werden in der WHERE-Klausel angegeben.
- Setzen Sie den Tabellennamen vor den Spaltennamen, wenn derselbe Spaltenname in mehreren Tabellen vorkommt.



Copyright © 2014, Oracle und/oder verbundene Unternehmen. All rights reserved. Alle Rechte vorbehalten.

Wenn Sie Daten aus mehreren Tabellen einer Datenbank benötigen, werden *Join*-Bedingungen verwendet. Zeilen in einer Tabelle können mit Zeilen einer anderen Tabelle über gemeinsame Werte in den entsprechenden Spalten verknüpft werden. (In der Regel geschieht dies über Primär- und Fremdschlüsselspalten.)

Um Daten aus zwei oder mehr verknüpften Tabellen anzuzeigen, geben Sie in der WHERE-Klausel eine einfache Join-Bedingung an.

Für die Syntax gilt:

<i>table1.column</i>	Tabelle und Spalte, aus der Daten abgerufen werden
<i>table1.column1</i> =	Bedingung, unter der Tabellen verknüpft (oder miteinander in
<i>table2.column2</i>	Verbindung gesetzt) werden

Richtlinien

- Setzen Sie bei der Erstellung von SELECT-Anweisungen, die Tabellen verknüpfen, den Tabellennamen vor den Spaltennamen, um den Code besser verständlich zu machen und den Datenbankzugriff zu optimieren.
- Wenn derselbe Spaltenname in mehreren Tabellen verwendet wird, müssen Sie dem Spaltennamen den Tabellennamen voranstellen.
- Um n Tabellen miteinander zu verknüpfen, benötigen Sie mindestens $n-1$ Join-Bedingungen. Beispiel: Um vier Tabellen zu verknüpfen, benötigen Sie mindestens drei Joins. Diese Regel gilt möglicherweise nicht, wenn Ihre Tabelle über einen verketteten Primärschlüssel verfügt, da in diesem Fall mehrere Spalten benötigt werden, um jede Zeile eindeutig zu identifizieren.

Typen von Joins

- Natural Join
- Equi Join
- Non-Equi Join
- Outer Join
- Self Join
- Cross Join

ORACLE

Copyright © 2014, Oracle und/oder verbundene Unternehmen. All rights reserved. Alle Rechte vorbehalten.

Mit der Join-Syntax von Oracle können Sie Tabellen verknüpfen.

Hinweis: Vor Release Oracle9*i* galt für Joins eine Oracle-spezifische Syntax. Die SQL:1999-konforme Join-Syntax bietet gegenüber der Oracle-spezifischen Join-Syntax keine Performance-vorteile.

Mehrdeutige Spaltennamen eindeutig kennzeichnen

- Spaltennamen, die in mehreren Tabellen vorkommen, müssen durch das Tabellenpräfix eindeutig gekennzeichnet werden.
- Tabellenpräfixe verbessern die Performance.
- Verwenden Sie als Präfix Tabellenaliasnamen anstelle von vollständigen Tabellennamen.
- Tabellenaliasnamen verkürzen die Namen von Tabellen:
 - Kürzerer SQL-Code, weniger Speicherbedarf
- Spaltenaliasnamen dienen der Unterscheidung von Spalten, die identische Namen haben, aber in verschiedenen Tabellen stehen.

ORACLE®

Copyright © 2014, Oracle und/oder verbundene Unternehmen. All rights reserved. Alle Rechte vorbehalten.

Wenn Sie zwei oder mehr Tabellen miteinander verknüpfen, kennzeichnen Sie die Spaltennamen durch den jeweiligen Tabellennamen, um Mehrdeutigkeit zu vermeiden. Ohne das Tabellenpräfix könnte sich der Spaltenname `DEPARTMENT_ID` in der Liste `SELECT` sowohl auf die entsprechende Spalte in der Tabelle `DEPARTMENTS` als auch in der Tabelle `EMPLOYEES` beziehen. Daher müssen Sie das Tabellenpräfix hinzufügen, um die Abfrage auszuführen. Wenn ein Spaltenname lediglich in einer Tabelle existiert, besteht keine Notwendigkeit, die Spalte eindeutig zu kennzeichnen. Tabellenpräfixe verbessern jedoch die Performance, weil sie dem Oracle-Server genau angeben, wo sich die Spalten befinden.

Die eindeutige Kennzeichnung von Spaltennamen durch Tabellennamen kann insbesondere bei langen Tabellennamen sehr zeitaufwändig sein. Sie können daher anstelle von Tabellennamen *Tabellenaliasnamen* verwenden. Genau wie ein Spaltenalias einer Spalte einen anderen Namen zuweist, legt ein Tabellenalias einen anderen Namen für eine Tabelle fest. Tabellenaliasnamen dienen dazu, den SQL-Code kürzer und damit weniger speicherintensiv zu halten.

Der Tabellename wird komplett angegeben. Danach folgt ein Leerzeichen und dann der Tabellenalias. Beispiel: Der Tabelle `EMPLOYEES` kann der Alias `e` und der Tabelle `DEPARTMENTS` der Alias `d` zugewiesen werden.

Richtlinien

- Tabellenaliasnamen können bis zu 30 Zeichen lang sein, kürzere Aliasnamen sind allerdings besser.
- Wenn für einen bestimmten Tabellennamen in der `FROM`-Klausel ein Tabellenalias verwendet wird, muss dieser Tabellenalias in der gesamten `SELECT`-Anweisung anstelle des Tabellennamens verwendet werden.
- Wählen Sie möglichst aussagekräftige Tabellenaliasnamen.
- Ein Tabellenalias gilt nur für die aktuelle `SELECT`-Anweisung.

Natural Joins

- Die Klausel NATURAL JOIN basiert auf allen Spalten, die in beiden Tabellen denselben Namen haben.
- Die Klausel wählt Zeilen aus Tabellen, deren Spalten identische Namen und Datenwerte aufweisen.
- Beispiel:

```
SELECT country_id, location_id, country_name, city  
FROM countries NATURAL JOIN locations;
```

COUNTRY_ID	LOCATION_ID	COUNTRY_NAME	CITY
1 US	1400	United States of America	Southlake
2 US	1500	United States of America	South San Francisco
3 US	1700	United States of America	Seattle
4 CA	1800	Canada	Toronto
5 UK	2500	United Kingdom	Oxford

ORACLE

Copyright © 2014, Oracle und/oder verbundene Unternehmen. All rights reserved. Alle Rechte vorbehalten.

Sie können Tabellen auf der Basis von Spalten mit gleichem Namen und gleichem Datentyp automatisch verknüpfen. Hierzu verwenden Sie die Schlüsselwörter NATURAL JOIN

Hinweis: Der Join kann nur für Spalten erfolgen, die in beiden Tabellen denselben Namen und denselben Datentyp aufweisen. Wenn die Spalten denselben Namen haben, ihre Datentypen sich jedoch unterscheiden, verursacht die Syntax NATURAL JOIN einen Fehler.

Im Beispiel auf der Folie werden die Tabellen COUNTRIES und LOCATIONS über die Spalte COUNTRY_ID verknüpft, da dies der einzige Spaltenname ist, der in beiden Tabellen vorkommt. Wären weitere übereinstimmende Spalten vorhanden, würde der Join alle diese Spalten berücksichtigen.

Equi Joins

EMPLOYEES

	EMPLOYEE_ID	DEPARTMENT_ID
1	200	10
2	201	20
3	202	20
4	205	110
5	206	110
6	100	90
7	101	90
8	102	90
9	103	60
10	104	60

DEPARTMENTS

	DEPARTMENT_ID	DEPARTMENT_NAME
1	10	Administration
2	20	Marketing
3	50	Shipping
4	60	IT
5	80	Sales
6	90	Executive
7	110	Accounting
8	190	Contracting

...

Fremdschlüssel

Primärschlüssel

ORACLE

Copyright © 2014, Oracle und/oder verbundene Unternehmen. All rights reserved. Alle Rechte vorbehalten.

Ein **Equi Join** ist ein Join mit einer Join-Bedingung, die einen Gleichheitsoperator enthält. Equi Joins kombinieren Zeilen, die äquivalente Werte in den angegebenen Spalten enthalten. Um die Abteilung eines Mitarbeiters zu bestimmen, vergleichen Sie die Werte in der Spalte DEPARTMENT_ID der Tabelle EMPLOYEES mit den Werten für DEPARTMENT_ID in der Tabelle DEPARTMENTS. Die Beziehung zwischen den Tabellen EMPLOYEES und DEPARTMENTS ist ein **Equi Join**, das heißt, die Werte in der Spalte DEPARTMENT_ID müssen in beiden Tabellen gleich sein. Häufig sind bei dieser Art von Join Primärschlüssel- und Fremdschlüsselkomponenten beteiligt.

Hinweis: Equi Joins werden auch als *einfache Joins* bezeichnet.

Records mithilfe von Equi Joins abrufen

```
SELECT e.employee_id, e.last_name, e.department_id,
       d.department_id, d.location_id
  FROM employees e JOIN departments d
 WHERE e.department_id = d.department_id;
```

EMPLOYEE_ID	LAST_NAME	DEPARTMENT_ID	DEPARTMENT_ID_1	LOCATION_ID
1	200 Whalen	10	10	1700
2	201 Hartstein	20	20	1800
3	202 Fay	20	20	1800
4	144 Vargas	50	50	1500
5	143 Matos	50	50	1500
6	142 Davies	50	50	1500
7	141 Rajs	50	50	1500
8	124 Mourgos	50	50	1500
9	103 Hunold	60	60	1400
10	104 Ernst	60	60	1400
11	107 Lorentz	60	60	1400
...				

ORACLE

Copyright © 2014, Oracle und/oder verbundene Unternehmen. All rights reserved. Alle Rechte vorbehalten.

Im Beispiel auf der Folie gilt:

- Die **SELECT-Klausel** gibt die abzurufenden Spaltennamen an:
 - Nachname, Personalnummer und Abteilungsnummer des Mitarbeiters aus den entsprechenden Spalten der Tabelle EMPLOYEES
 - Abteilungs-ID und Standort-ID aus den entsprechenden Spalten der Tabelle DEPARTMENTS
- Die **FROM-Klausel** gibt die beiden Tabellen an, auf die die Datenbank zugreifen muss:
 - Tabelle EMPLOYEES
 - Tabelle DEPARTMENTS
- Die **WHERE-Klausel** gibt an, wie die Tabellen verknüpft werden sollen:
`e.department_id = d.department_id`

Da die Spalte DEPARTMENT_ID in beiden Tabellen enthalten ist, muss dem Spaltennamen der Tabellenalias vorangestellt werden, um Mehrdeutigkeit zu vermeiden. Eine Kennzeichnung anderer Spalten, die nicht in beiden Tabellen vorkommen, durch einen Tabellenalias ist nicht erforderlich, empfiehlt sich jedoch aus Performancegründen.

Hinweis: Wenn Sie die Abfrage mit dem Symbol **Execute Statement** ausführen, hängt SQL Developer zur Unterscheidung der beiden DEPARTMENT_IDS "_1" an.

Zusätzliche Suchbedingungen mit den Operatoren AND und WHERE

```
SELECT d.department_id, d.department_name, l.city
FROM departments d JOIN locations l
ON d.location_id = l.location_id
AND d.department_id IN (20, 50);
```

	DEPARTMENT_ID	DEPARTMENT_NAME	CITY
1	20	Marketing	Toronto
2	50	Shipping	South San Francisco

```
SELECT d.department_id, d.department_name, l.city
FROM departments d JOIN locations l
ON d.location_id = l.location_id
WHERE d.department_id IN (20, 50);
```

ORACLE

Copyright © 2014, Oracle und/oder verbundene Unternehmen. All rights reserved. Alle Rechte vorbehalten.

Neben dem Join können Sie auch Kriterien für die WHERE-Klausel angeben, um die Zeilen aus einer oder mehreren Tabellen des Joins einzuschränken. Im Beispiel auf der Folie werden die Tabellen DEPARTMENTS und LOCATIONS verknüpft und darüber hinaus nur die Abteilungen mit der ID 20 oder 50 angezeigt. Um weitere Bedingungen für die ON-Klausel aufzunehmen, können Sie AND-Klauseln hinzufügen. Alternativ können Sie zusätzliche Bedingungen über eine WHERE-Klausel anwenden.

Beide Abfragen führen zum selben Ergebnis.

LAST_NAME	DEPARTMENT_ID	DEPARTMENT_NAME
Matos	50	Shipping

Records mithilfe von Non-Equi Joins abrufen

```
SELECT e.last_name, e.salary, j.grade_level
FROM   employees e JOIN job_grades j
ON    e.salary
      BETWEEN j.lowest_sal AND j.highest_sal;
```

	LAST_NAME	SALARY	GRADE_LEVEL
1	Vargas	2500	A
2	Matos	2600	A
3	Davies	3100	B
4	Rajs	3500	B
5	Lorentz	4200	B
6	Whalen	4400	B
7	Fay	6000	C

...

ORACLE

Copyright © 2014, Oracle und/oder verbundene Unternehmen. All rights reserved. Alle Rechte vorbehalten.

Im Beispiel auf der Folie wird ein Non-Equi Join erstellt, um die Gehaltsstufe eines Mitarbeiters auszuwerten. Das Gehalt muss *zwischen* dem Mindest- und Höchstwert für die einzelnen Gehaltsbereiche liegen.

Beachten Sie, dass jeder Mitarbeiter beim Ausführen dieser Abfrage genau einmal angezeigt wird. Kein Mitarbeiter wird in der Liste mehrfach angezeigt. Dafür gibt es zwei Gründe:

- Keine der Zeilen in der Tabelle `job_grades` enthält Gehaltsstufen, die sich überschneiden. Das bedeutet, der Gehaltswert für einen Mitarbeiter kann nur zwischen dem Mindest- und Höchstwert aus einer der Zeilen der Gehaltsgruppentabelle liegen.
- Alle Mitarbeitergehälter liegen innerhalb der durch die Tabelle `job_grades` festgelegten Grenzwerte. Kein Mitarbeiter verdient also weniger als das niedrigste Gehalt aus der Spalte `LOWEST_SAL` oder mehr als das höchste Gehalt aus der Spalte `HIGHEST_SAL`.

Hinweis: Es können auch andere Bedingungen verwendet werden, zum Beispiel `<=` und `>=`. `BETWEEN` ist jedoch die einfachste Möglichkeit. Geben Sie zuerst den Mindest- und dann den Höchstwert an, wenn Sie den Operator `BETWEEN` verwenden. Der Oracle-Server übersetzt den Operator `BETWEEN` in ein `AND`-Bedingungspaar. Daher bietet `BETWEEN` keine Performancevorteile und dient nur der logischen Einfachheit.

Die Tabellenaliasnamen wurden im Beispiel auf der Folie aus Performancegründen verwendet, nicht wegen möglicher Mehrdeutigkeit.

Records mithilfe von USING-Klauseln abrufen

- Um nur eine Spalte zu verwenden, wenn mehrere übereinstimmende Spalten existieren, verwenden Sie eine USING-Klausel.
- Sie können diese Klausel nicht mit einem NATURAL Join angeben.
- Kennzeichnen Sie den Spaltennamen nicht mit einem Tabellennamen oder -alias.
- Beispiel:

```
SELECT country_id, country_name, location_id, city
  FROM countries JOIN locations
     USING (country_id) ;
```

	COUNTRY_ID	COUNTRY_NAME	LOCATION_ID	CITY
1	US	United States of America	1400	Southlake
2	US	United States of America	1500	South San Francisco
3	US	United States of America	1700	Seattle
4	CA	Canada	1800	Toronto
5	UK	United Kingdom	2500	Oxford

ORACLE

Copyright © 2014, Oracle und/oder verbundene Unternehmen. All rights reserved. Alle Rechte vorbehalten.

Im Beispiel auf der Folie werden die Spalten COUNTRY_ID aus den Tabellen COUNTRIES und LOCATIONS verknüpft. Auf diese Weise wird die LOCATION_ID für den Standort angezeigt, an dem ein Mitarbeiter arbeitet.

Records mithilfe von ON-Klauseln abrufen

- Die Join-Bedingung für Natural Joins ist im Prinzip ein Equi Join aller Spalten mit gleichem Namen.
- Mit der ON-Klausel können Sie beliebige Bedingungen oder die zu verknüpfenden Spalten angeben.
- Die ON-Klausel macht den Code verständlicher.

```
SELECT e.employee_id, e.last_name, j.department_id,  
      FROM employees e JOIN job_history j  
      ON (e.employee_id = j.employee_id);
```

	EMPLOYEE_ID	LAST_NAME	DEPARTMENT_ID
1	101	Kochhar	110
2	101	Kochhar	110
3	102	De Haan	60
4	176	Taylor	80
5	176	Taylor	80
6	200	Whalen	90
7	200	Whalen	90
8	201	Hartstein	20

ORACLE

Copyright © 2014, Oracle und/oder verbundene Unternehmen. All rights reserved. Alle Rechte vorbehalten.

Geben Sie mit der ON-Klausel eine Join-Bedingung an. Auf diese Weise können Sie Join-Bedingungen in der WHERE-Klausel getrennt von Such- oder Filterkriterien angeben.

In diesem Beispiel werden die Spalten EMPLOYEE_ID aus den Tabellen EMPLOYEES und JOB_HISTORY mit der ON-Klausel verknüpft. Immer wenn eine Personalnummer in der Tabelle EMPLOYEES einer Personalnummer in der Tabelle JOB_HISTORY entspricht, wird die Zeile zurückgegeben. Der Tabellenalias ist erforderlich, um die übereinstimmenden Spaltennamen zu kennzeichnen.

Mit der ON-Klausel können Sie auch Spalten mit unterschiedlichen Namen verknüpfen. Die Klammersetzung um die verknüpften Spalten im Beispiel auf der Folie, (e.employee_id = j.employee_id), ist optional. Mit anderen Worten: ON e.employee_id = j.employee_id funktioniert genauso.

Hinweis: Wenn Sie die Abfrage mit dem Symbol **Execute Statement** ausführen, hängt SQL Developer zur Unterscheidung der beiden employee_ids "1" an.

Left Outer Joins

- Ein Join zwischen zwei Tabellen, der alle übereinstimmenden Zeilen sowie die nicht übereinstimmenden Zeilen aus der linken Tabelle zurückgibt, wird als LEFT OUTER JOIN bezeichnet.
- Beispiel:

```
SELECT c.country_id, c.country_name, l.location_id, l.city
FROM   countries c LEFT OUTER JOIN locations l
ON    (c.country_id = l.country_id) ;
```

	COUNTRY_ID	COUNTRY_NAME	LOCATION_ID	CITY
1	CA	Canada	1800	Toronto
2	DE	Germany	(null)	(null)
3	UK	United Kingdom	2500	Oxford
4	US	United States of America	1400	Southlake
5	US	United States of America	1500	South San Francisco
6	US	United States of America	1700	Seattle

ORACLE

Copyright © 2014, Oracle und/oder verbundene Unternehmen. All rights reserved. Alle Rechte vorbehalten.

Diese Abfrage ruft alle Zeilen aus der Tabelle COUNTRIES (linke Tabelle) ab, auch wenn keine Übereinstimmung in der Tabelle LOCATIONS gegeben ist.

Right Outer Joins

- Ein Join zwischen zwei Tabellen, der alle übereinstimmenden Zeilen sowie die nicht übereinstimmenden Zeilen aus der rechten Tabelle zurückgibt, wird als RIGHT OUTER JOIN bezeichnet.
- Beispiel:

```
SELECT d.department_id, d.department_name, l.location_id,  
l.city  
FROM departments d RIGHT OUTER JOIN locations l  
ON (d.location_id = l.location_id) ;
```

	DEPARTMENT_ID	DEPARTMENT_NAME	LOCATION_ID	CITY
1	10	Administration	1700	Seattle
2	20	Marketing	1800	Toronto
3	30	Purchasing	1700	Seattle
4	40	Human Resources	2400	London
...				
26	260	Recruiting	1700	Seattle
27	270	Payroll	1700	Seattle
28	(null)	(null)	2000	Beijing
29	(null)	(null)	3000	Bern
...				

ORACLE

Copyright © 2014, Oracle und/oder verbundene Unternehmen. All rights reserved. Alle Rechte vorbehalten.

Diese Abfrage ruft alle Zeilen aus der Tabelle LOCATIONS (rechte Tabelle) ab, auch wenn keine Übereinstimmung in der Tabelle COUNTRIES gegeben ist.

Full Outer Joins

- Ein Join zwischen zwei Tabellen, der neben allen übereinstimmenden Zeilen auch die nicht übereinstimmenden Zeilen aus beiden Tabellen zurückgibt, wird als FULL OUTER JOIN bezeichnet.
- Beispiel:

```
SELECT e.last_name, d.department_id, d.manager_id,
       d.department_name
  FROM   employees e FULL OUTER JOIN departments d
  ON    (e.manager_id = d.manager_id) ;
```

	LAST_NAME	DEPARTMENT_ID	MANAGER_ID	DEPARTMENT_NAME
1	King	(null)	(null)	(null)
2	Kochhar	90	100	Executive
3	De Haan	90	100	Executive
4	Hunold	(null)	(null)	(null)

...

106	Higgins	(null)	(null)	(null)
107	Gietz	110	205	Accounting
108	(null)	270	(null)	Payroll
109	(null)	260	(null)	Recruiting

...

ORACLE

Copyright © 2014, Oracle und/oder verbundene Unternehmen. All rights reserved. Alle Rechte vorbehalten.

Diese Abfrage ruft alle Zeilen aus der Tabelle EMPLOYEES ab, auch wenn es keine Übereinstimmung dazu in der Tabelle DEPARTMENTS gibt. Außerdem werden alle Zeilen aus der Tabelle DEPARTMENTS abgerufen, unabhängig davon, ob Übereinstimmungen in der Tabelle EMPLOYEES vorhanden sind.

Self-Joins – Beispiel

```
SELECT worker.last_name || ' works for '
    || manager.last_name
  FROM employees worker JOIN employees manager
 WHERE worker.manager_id = manager.employee_id
 ORDER BY worker.last_name;
```

WORKER_LAST_NAME 'WORKSFOR' MANAGER_LAST_NAME
1 Abel works for Zlotkey
2 Davies works for Mourgos
3 De Haan works for King
4 Ernst works for Hunold
5 Fay works for Hartstein
6 Gietz works for Higgins
7 Grant works for Zlotkey
8 Hartstein works for King
9 Higgins works for Kochhar

...

ORACLE

Copyright © 2014, Oracle und/oder verbundene Unternehmen. All rights reserved. Alle Rechte vorbehalten.

In bestimmten Situationen müssen Sie eine Tabelle mit sich selbst verknüpfen. Um für jeden Mitarbeiter den Namen des zuständigen Managers zu ermitteln, verknüpfen Sie die Tabelle EMPLOYEES mit sich selbst und führen damit einen Self Join durch. Im Beispiel auf der Folie wird die Tabelle EMPLOYEES mit sich selbst verknüpft. Um in der FROM-Klausel zwei Tabellen zu simulieren, werden für dieselbe Tabelle (EMPLOYEES) zwei Aliasnamen verwendet: `worker` und `manager`.

In diesem Beispiel enthält die WHERE-Klausel den Join, der wie folgt interpretiert wird: "wobei die Manager-ID für einen Mitarbeiter der Personalnummer des Managers entspricht".

Cross Joins

- Ein CROSS JOIN ist ein JOIN-Vorgang, der das kartesische Produkt aus zwei Tabellen zurückgibt.
- Beispiel:

```
SELECT department_name, city  
FROM department CROSS JOIN location;
```

DEPARTMENT_NAME	CITY
1 Administration	Oxford
2 Administration	Seattle
3 Administration	South San Francisco
4 Administration	Southlake
5 Administration	Toronto
6 Marketing	Oxford
7 Marketing	Seattle
8 Marketing	South San Francisco
9 Marketing	Southlake
10 Marketing	Toronto

...

ORACLE

Copyright © 2014, Oracle und/oder verbundene Unternehmen. All rights reserved. Alle Rechte vorbehalten.

Die Syntax für einen CROSS JOIN gibt das Kreuzprodukt an, das auch als kartesisches Produkt bezeichnet wird. Ein Cross Join gibt das Kreuzprodukt aus zwei Relationen zurück und entspricht im Prinzip einer durch Kommas getrennten Oracle Database-Notation.

Zwischen den beiden Tabellen im CROSS JOIN geben Sie keine WHERE-Bedingungen an.

Zusammenfassung

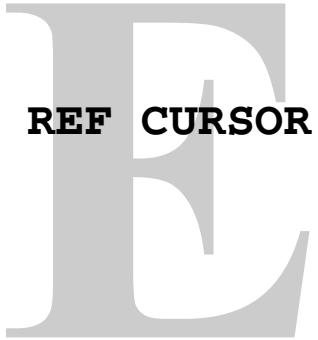
In diesem Anhang haben Sie Folgendes gelernt:

- Mit SELECT-Anweisungen Zeilen aus einer oder mehreren Tabelle abrufen
- Struktur von Objekten mit DDL-Anweisungen ändern
- Daten in vorhandenen Schemaobjekten mit DML-Anweisungen bearbeiten
- Mit Anweisungen zur Transaktionskontrolle Änderungen verwalten, die über DML-Anweisungen vorgenommen wurden
- Daten aus mehreren Tabellen mithilfe von Joins anzeigen

ORACLE

Copyright © 2014, Oracle und/oder verbundene Unternehmen. All rights reserved. Alle Rechte vorbehalten.

In SQL gibt es eine Vielzahl häufig eingesetzter Befehle und Anweisungen wie DDL-, DML- und Transaktionskontrollanweisungen sowie Joins.



ORACLE®

Copyright © 2014, Oracle und/oder verbundene Unternehmen. All rights reserved. Alle Rechte vorbehalten.

Cursorvariablen

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

ORACLE

Copyright © 2014, Oracle und/oder verbundene Unternehmen. All rights reserved. Alle Rechte vorbehalten.

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

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

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

Cursorvariablen – Einsatzmöglichkeiten

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

ORACLE

Copyright © 2014, Oracle und/oder verbundene Unternehmen. All rights reserved. Alle Rechte vorbehalten.

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

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

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

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

REF CURSOR-Typen definieren

REF CURSOR-Typ definieren:

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

Cursorvariable mit diesem Typ deklarieren:

```
ref_cv ref_type_name;
```

Beispiel:

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

ORACLE

Copyright © 2014, Oracle und/oder verbundene Unternehmen. All rights reserved. Alle Rechte vorbehalten.

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

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

Dabei gilt:

ref_type_name	Typspezifikation, die in nachfolgenden Deklarationen von Cursorvariablen verwendet wird
return_type	Stellt einen Record oder eine Zeile in einer Datenbanktabelle dar

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

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

```
DECLARE
```

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

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

Cursorvariablen deklarieren

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

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

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

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

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

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

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

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

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

Cursorvariablen als Parameter

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

```
DECLARE
```

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

OPEN-FOR-, FETCH- und CLOSE-Anweisungen

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

ORACLE®

Copyright © 2014, Oracle und/oder verbundene Unternehmen. All rights reserved. Alle Rechte vorbehalten.

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

Cursorvariablen öffnen

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

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

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

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

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

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

Daten aus einer Cursorvariablen lesen

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

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

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

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

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

Cursorvariablen schließen

Cursorvariablen werden mit CLOSE-Anweisungen deaktiviert. Die zugehörige Ergebnismenge ist dann undefiniert. Syntax:

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

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

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

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

Fetch-Vorgänge – Beispiel

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

ORACLE

Copyright © 2014, Oracle und/oder verbundene Unternehmen. All rights reserved. Alle Rechte vorbehalten.

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