

Oracle Database 11g: PL/SQL-Grundlagen

Schulungsunterlagen

D49990DE10

Production 1.0

Oktober 2007

Bestellnummer: D52607

ORACLE®

Autor

Tulika Srivastava

Technischer Inhalt und Überarbeitung

Tom Best
Christoph Burandt

Yanti Chang
Ashita Dhir
Peter Driver

Gerlinde Frenzen
Nancy Greenberg
Chaitanya Kortamaddi
Tim Leblanc
Wendy Lo
Bryan Roberts
Lauran Serhal
Abhishek X Singh
Puja Singh
Lex.Van.Der Werff

Editoren

Atanu Raychaudhuri
Richard Wallis

Grafische Gestaltung

Satish Bettegowda
Rajiv Chandrabhanu

Herausgeber

Sujatha Nagendra
Michael Sebastian
Giri Venugopal

Copyright © 2007, Oracle. Alle Rechte vorbehalten.

Dieser Kurs liefert einen Überblick über die für Release 11g geplanten Funktionen und Weiterentwicklungen. Er soll Sie lediglich dabei unterstützen, die Vorteile zu beurteilen, die sich durch ein Upgrade auf 11g ergeben, und Ihnen die Planung Ihrer IT-Projekte erleichtern.

Dieser Kurs enthält in all seinen Formen, einschließlich der Kursübungen und der gedruckten Materialien, Informationen, die ausschließlich Eigentum von Oracle sind. Dieser Kurs und die darin enthaltenen Informationen dürfen ohne vorherige schriftliche Genehmigung von Oracle weder veröffentlicht, vervielfältigt, reproduziert oder an Personen weitergegeben werden, die nicht Mitarbeiter von Oracle sind. Dieser Kurs und sein Inhalt sind nicht Bestandteil Ihres Lizenzvertrags und werden auch nicht Bestandteil einer vertraglichen Vereinbarung mit Oracle oder dessen Tochtergesellschaften oder verbundenen Unternehmen.

Dieser Kurs dient ausschließlich informativen Zwecken und soll Sie lediglich dabei unterstützen, die Implementierung und ein Upgrade der beschriebenen Funktionen zu planen. Er stellt keine Verpflichtung zur Bereitstellung von Materialien, Code oder Funktionalität dar und darf nicht als Grundlage einer Kaufentscheidung herangezogen werden. Die Entwicklung, Freigabe und zeitliche Herausgabe der in diesem Dokument beschriebenen Funktionen oder Funktionalität liegt im alleinigen Ermessen von Oracle.

Diese Unterlagen enthalten Informationen, die gesetzlich durch Urheberrechtsgesetze und andere Immaterialgütergesetze geschützt sind. Das Anfertigen von Kopien und das Ausdrucken dieser Unterlagen ist ausschließlich für den eigenen Gebrauch in einem Oracle Trainingskurs gestattet. Die Unterlagen in irgendeiner Weise zu verändern oder umzugestalten, ist untersagt. Des Weiteren ist es ohne ausdrückliche Genehmigung durch Oracle nicht erlaubt, diese Unterlagen oder Teile davon zu nutzen, weiterzugeben, herunterzuladen, hochzuladen, zu vervielfältigen, auszudrucken, anzuzeigen, vorzuführen, zu reproduzieren, zu veröffentlichen, zu lizenziieren, zu versenden, zu übermitteln oder zu vertreiben, es sei denn das Urhebergesetz lässt eine derartige Nutzung ausdrücklich zu.

Oracle behält sich das Recht jederzeitiger Änderung dieser Unterlagen ohne vorherige Mitteilung vor. Falls Sie Unstimmigkeiten in diesen Unterlagen finden, benachrichtigen Sie uns bitte schriftlich unter folgender Adresse: Oracle University, 500 Oracle Parkway, Redwood Shores, California 94065 USA. Oracle übernimmt keine Garantie dafür, dass diese Unterlagen fehlerfrei sind.

Wenn diese Unterlagen an die Regierung der Vereinigten Staaten von Amerika oder an jemanden geliefert werden, der diese Unterlagen im Auftrag oder im Namen der Regierung der Vereinigten Staaten von Amerika verwendet, gelten die folgenden Beschränkungen:

U.S. GOVERNMENT RIGHTS

The U.S. Government's rights to use, modify, reproduce, release, perform, display, or disclose these training materials are restricted by the terms of the applicable Oracle license agreement and/or the applicable U.S. Government contract.

Oracle ist eine eingetragene Marke der Oracle Corporation und/oder ihrer verbundenen Unternehmen. Andere Namen und Bezeichnungen können Marken der jeweiligen Inhaber sein.

Inhalt

I Einführung

- Ziele I-2
- Kursziele I-3
- Kursagenda I-4
- Human Resources-(HR-)Schema für diesen Kurs I-6
- PL/SQL-Entwicklungsumgebungen I-7
- PL/SQL in Oracle SQL Developer codieren I-8
- PL/SQL in SQL*Plus codieren I-9
- PL/SQL in Oracle JDeveloper codieren I-10
- SQL Developer starten und Datenbankverbindung erstellen I-11
- Schemaobjekte erstellen I-12
- SQL Worksheet I-13
- SQL-Anweisungen ausführen I-14
- Zusammenfassung I-15

1 Einführung in PL/SQL

- Ziele 1-2
- PL/SQL 1-3
- PL/SQL-Umgebung 1-5
- PL/SQL – Vorteile 1-6
- PL/SQL-Blockstruktur 1-9
- Blocktypen 1-11
- Programmkonstrukte 1-13
- Anonyme Blöcke erstellen 1-15
- Anonyme Blöcke ausführen 1-16
- Ausgabe von PL/SQL-Blöcken testen 1-17
- Zusammenfassung 1-19
- Übungen zu Lektion 1 – Überblick 1-20

2 PL/SQL-Variablen deklarieren

- Ziele 2-2
- Variablen verwenden 2-3
- Anforderungen für Variablennamen 2-4
- Variablen in PL/SQL verwenden 2-5
- PL/SQL-Variablen deklarieren und initialisieren 2-6
- Begrenzungszeichen in Zeichenfolgenliteralen 2-8
- Variabtentypen 2-9

PL/SQL-Variablen deklarieren und initialisieren – Richtlinien 2-12
PL/SQL-Variablen deklarieren – Richtlinien 2-13
Skalare Datentypen 2-14
Skalare Basisdatentypen 2-15
Skalare Variablen deklarieren 2-19
%TYPE-Attribut 2-20
Variablen mit dem %TYPE-Attribut deklarieren 2-22
Boolesche Variablen deklarieren 2-23
Bind-Variablen 2-24
Bind-Variablenwerte ausgeben 2-26
Variablen mit LOB-Datentypen 2-28
Zusammengesetzte Datentypen 2-29
Zusammenfassung 2-30
Übungen zu Lektion 2 – Überblick 2-31

3 Ausführbare Anweisungen erstellen

Ziele 3-2
Lexikalische Einheiten in PL/SQL-Blöcken 3-3
PL/SQL-Blocksyntax – Richtlinien 3-5
Code kommentieren 3-6
SQL-Funktionen in PL/SQL 3-7
SQL-Funktionen in PL/SQL – Beispiele 3-8
Sequences in PL/SQL-Ausdrücken verwenden 3-9
Datentypen konvertieren 3-10
Verschachtelte Blöcke 3-13
Gültigkeitsbereich und Sichtbarkeit von Variablen 3-15
Identifier kennzeichnen 3-17
Quiz – Gültigkeitsbereich von Variablen bestimmen 3-18
Operatoren in PL/SQL 3-20
Operatoren in PL/SQL – Beispiele 3-21
Richtlinien für die Programmierung 3-22
Code einrücken 3-23
Zusammenfassung 3-24
Übungen zu Lektion 3 – Überblick 3-25

4 Mit dem Oracle-Datenbank-Server interagieren

Ziele 4-2
SQL-Anweisungen in PL/SQL 4-3
SELECT-Anweisungen in PL/SQL 4-4
Daten in PL/SQL abrufen 4-8
Benennungskonventionen 4-10

Daten mit PL/SQL bearbeiten 4-12
Daten einfügen 4-13
Daten aktualisieren 4-14
Daten löschen 4-15
Zeilen zusammenführen 4-16
SQL-Cursor 4-18
SQL-Cursor-Attribute für implizite Cursor 4-20
Zusammenfassung 4-22
Übungen zu Lektion 4 – Überblick 4-23

5 Kontrollstrukturen erstellen

Ziele 5-2
Ablauf der Ausführung steuern 5-3
IF-Anweisungen 5-4
Einfache IF-Anweisungen 5-6
Anweisung IF THEN ELSE 5-7
Klausel IF ELSIF ELSE 5-8
NULL-Wert in IF-Anweisung 5-9
CASE-Ausdrücke 5-10
CASE-Ausdrücke – Beispiel 5-11
Searched CASE-Ausdrücke 5-12
CASE-Anweisungen 5-13
NULL-Werte verwenden 5-14
Logiktabellen 5-15
Boolesche Bedingungen 5-16
Kontrollierte Iteration – LOOP-Anweisungen 5-17
Basisschleifen 5-18
WHILE-Schleifen 5-20
FOR-Schleifen 5-22
Richtlinien für Schleifen 5-26
Verschachtelte Schleifen und Labels 5-27
PL/SQL-Anweisung CONTINUE 5-29
PL/SQL-Anweisung CONTINUE – Beispiel 5-30
Zusammenfassung 5-32
Übungen zu Lektion 5 – Überblick 5-33

6 Zusammengesetzte Datentypen verwenden

Ziele 6-2
Zusammengesetzte Datentypen 6-3
PL/SQL-Records 6-6

PL/SQL-Records erstellen 6-7
Struktur von PL/SQL-Records 6-9
%ROWTYPE-Attribut 6-10
%ROWTYPE – Vorteile 6-12
%ROWTYPE-Attribut – Beispiel 6-13
Records mit %ROWTYPE einfügen 6-14
Zeilen in einer Tabelle mit einem Record aktualisieren 6-15
INDEX BY-Tabellen oder assoziative Arrays 6-16
INDEX BY-Tabellen erstellen 6-17
Struktur von INDEX BY-Tabellen 6-19
INDEX BY-Tabellen erstellen 6-20
INDEX BY-Tabellenmethoden 6-21
INDEX BY-Tabelle mit Records 6-22
INDEX BY-Tabelle mit Records – Beispiel 6-24
Nested Tables 6-25
VARRAY 6-27
Zusammenfassung 6-28
Übungen zu Lektion 6 – Überblick 6-29

7 Explizite Cursor

Ziele 7-2
Cursor 7-3
Explizite Cursor-Operationen 7-4
Explizite Cursor kontrollieren 7-5
Cursor deklarieren 7-7
Cursor öffnen 7-9
Daten aus Cursorn lesen 7-10
Cursor schließen 7-13
Cursor und Records 7-14
Cursor FOR-Schleifen 7-15
Explizite Cursor-Attribute 7-17
%ISOPEN-Attribut 7-18
%ROWCOUNT und %NOTFOUND – Beispiel 7-19
Cursor FOR-Schleifen mit Unterabfragen 7-20
Cursor mit Parametern 7-21
FOR UPDATE-Klausel 7-23
Klausel WHERE CURRENT OF 7-25
Cursor mit Unterabfragen 7-26
Zusammenfassung 7-27
Übungen zu Lektion 7 – Überblick 7-28

8 Exceptions behandeln

- Ziele 8-2
- Exceptions – Beispiel 8-3
- Exceptions mit PL/SQL behandeln 8-5
- Exceptions behandeln 8-6
- Exception-Typen 8-7
- Exceptions abfangen 8-8
- Exceptions abfangen – Richtlinien 8-10
- Vordefinierte Oracle-Server-Fehler abfangen 8-11
- Nicht vordefinierte Oracle-Server-Fehler abfangen 8-14
- Nicht vordefinierte Fehler 8-15
- Funktionen zum Abfangen von Exceptions 8-16
- Benutzerdefinierte Exceptions abfangen 8-18
- Exceptions in Unterblöcken propagieren 8-20
- Prozedur RAISE_APPLICATION_ERROR 8-21
- Zusammenfassung 8-24
- Übungen zu Lektion 8 – Überblick 8-25

9 Stored Procedures und Stored Functions erstellen

- Ziele 9-2
- Prozeduren und Funktionen 9-3
- Anonyme Blöcke und Unterprogramme – Unterschiede 9-4
- Prozedur – Syntax 9-5
- Prozeduren – Beispiel 9-6
- Prozeduren aufrufen 9-8
- Funktion – Syntax 9-9
- Funktionen – Beispiel 9-10
- Funktionen aufrufen 9-11
- Parameter an Funktionen übergeben 9-12
- Funktionen mit einem Parameter aufrufen 9-13
- Zusammenfassung 9-14
- Übungen zu Lektion 9 – Überblick 9-15

Anhang A: Übungen und Lösungen

Anhang B: Tabellenbeschreibungen und -daten

Anhang C: SQL Developer

Anhang D: SQL*Plus

Anhang E: Oracle JDeveloper

Anhang F: REF-Cursor

Zusätzliche Übungen

Zusätzliche Übungen – Lösungen

I

Einführung

ORACLE®

Copyright © 2007, Oracle. Alle Rechte vorbehalten.

Oracle Internal & Oracle Academy
Use Only

Ziele

Nach Ablauf dieser Lektion haben Sie folgende Ziele erreicht:

- Kursaufbau beschreiben
- HR-Beispielschema beschreiben
- Verfügbare Entwicklungsumgebungen auflisten
- Grundlegende Funktionen von SQL Developer prüfen



Copyright © 2007, Oracle. Alle Rechte vorbehalten.

Lernziel

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. Darüber hinaus erhalten Sie eine Einführung in die verschiedenen Produkte der Grid-Infrastruktur von Oracle 10g.

Kursziele

Nach Ablauf dieses Kurses haben Sie folgende Ziele erreicht:

- **Programmiererweiterungen angeben, die PL/SQL für SQL zur Verfügung stellt**
- **PL/SQL-Code erstellen, der eine Schnittstelle zur Datenbank bildet**
- **Anonyme PL/SQL-Blöcke entwerfen, die effizient ausgeführt werden**
- **PL/SQL-Programmkonstrukte und Bedingungsanweisungen verwenden**
- **Laufzeitfehler behandeln**
- **Stored Procedures und Stored Functions beschreiben**

ORACLE

Copyright © 2007, Oracle. Alle Rechte vorbehalten.

Kursziele

In diesem Kurs werden die Grundlagen von PL/SQL vorgestellt. Die PL/SQL-Syntax, -Blöcke und -Programmkonstrukte werden beschrieben. Darüber hinaus werden die Vorteile der Integration von SQL mit diesen Konstrukten herausgestellt. Sie lernen, 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.

Kursagenda

Lektionen des ersten Tages:

- I. Einführung
- 1. Einführung in PL/SQL
- 2. PL/SQL-Variablen deklarieren
- 3. Ausführbare Anweisungen erstellen
- 4. Mit dem Oracle-Datenbank-Server interagieren
- 5. Kontrollstrukturen erstellen

ORACLE®

Copyright © 2007, Oracle. Alle Rechte vorbehalten.

Kursagenda

Lektionen des zweiten Tages:

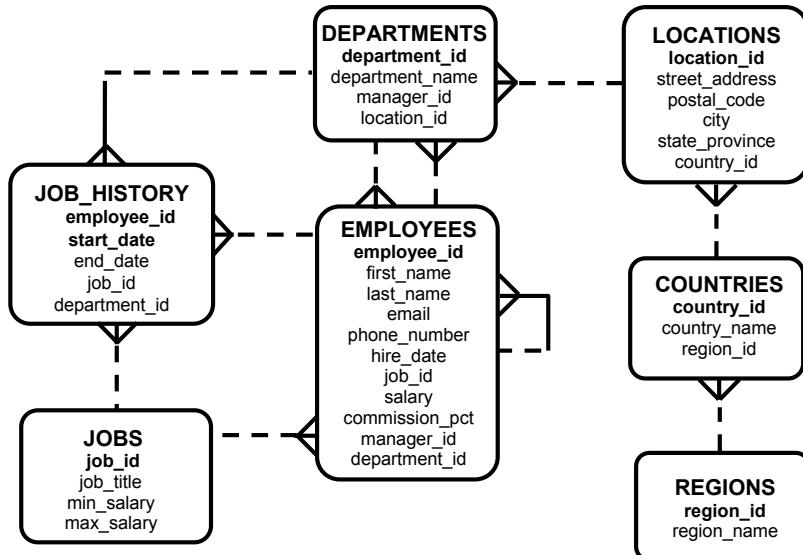
- 6. Zusammengesetzte Datentypen**
- 7. Explizite Cursor**
- 8. Exceptions behandeln**
- 9. Stored Procedures und Stored Functions erstellen**

ORACLE®

Copyright © 2007, Oracle. Alle Rechte vorbehalten.

Oracle Internal & Oracle Academy
Use Only

Human Resources-(HR-)Schema für diesen Kurs



ORACLE

Copyright © 2007, Oracle. Alle Rechte vorbehalten.

Human Resources-(HR-)Schema beschreiben

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

Tabellenbeschreibungen

- REGIONS enthält Zeilen, die für eine Region wie Amerika, Asien usw. stehen.
- COUNTRIES enthält Zeilen für Länder, die jeweils einer Region zugeordnet sind.
- LOCATIONS enthält die eindeutigen Adressen der spezifischen Büros, Lager oder Produktionsstandorte 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 EMPLOYEES-Tabelle aufweisen.
- EMPLOYEES enthält Details zu den einzelnen zu einer Abteilung gehörenden Mitarbeitern. Nicht alle Mitarbeiter müssen einer Abteilung zugeordnet sein.
- JOBS enthält die Berufsarten, 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 in diese Tabelle eine neue Zeile mit den alten Informationen des Mitarbeiters eingefügt.

PL/SQL-Entwicklungsumgebungen

In diesem Kurs werden die folgenden Tools zur Entwicklung von PL/SQL-Code bereitgestellt:

- **Oracle SQL Developer (in diesem Kurs verwendet)**
- **Oracle SQL*Plus**
- **Integrierte Entwicklungsumgebung von Oracle JDeveloper**

ORACLE

Copyright © 2007, Oracle. Alle Rechte vorbehalten.

PL/SQL-Entwicklungsumgebungen

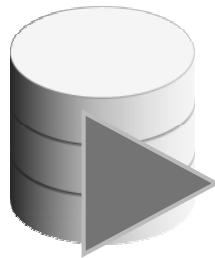
Es gibt viele Tools, die eine Umgebung zur Entwicklung von PL/SQL-Code bereitstellen. Oracle bietet mehrere Tools an, mit denen Sie PL/SQL-Code erstellen können. Die folgenden Entwicklungs-Tools können in diesem Kurs u. a. verwendet werden:

- **Oracle SQL Developer:** Ein grafisches Tool
- **Oracle SQL*Plus:** Eine fensterbasierte Anwendung oder Befehlszeilenanwendung
- **Oracle JDeveloper:** Eine fensterbasierte integrierte Entwicklungsumgebung (IDE)

Hinweis: Die im Kursmaterial vorgestellten Code- und Bildschirmbeispiele wurden von Ausgaben in der SQL Developer-Umgebung erzeugt.

PL/SQL in Oracle SQL Developer codieren

- Oracle SQL Developer ist ein kostenloses grafisches Tool zur Erhöhung Ihrer Produktivität und Vereinfachung der Aufgaben bei der Datenbankentwicklung.
- Sie können sich mit der Standardauthentifizierung für die Oracle-Datenbank bei jedem Oracle-Zieldatenbankschema anmelden.
- In diesem Kurs verwenden Sie SQL Developer.



SQL Developer

ORACLE®

Copyright © 2007, Oracle. Alle Rechte vorbehalten.

Was ist Oracle SQL Developer?

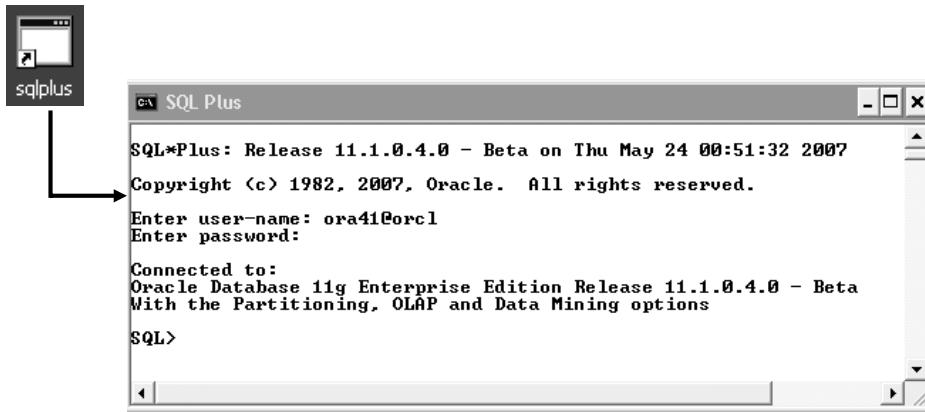
Oracle SQL Developer ist ein kostenloses grafisches Tool, das Ihre Produktivität erhöht und Routineaufgaben der Datenbankentwicklung vereinfacht. Mit wenigen Mausklicks können Sie problemlos Stored Procedures erstellen und debuggen, SQL-Anweisungen testen und Optimizer-Pläne anzeigen.

Mit SQL Developer, dem grafischen Tool zur Datenbankentwicklung, werden folgende Aufgaben vereinfacht:

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

Sie können sich mit der Standardauthentifizierung für Oracle-Datenbanken bei jedem Oracle-Zieldatenbankschema anmelden. Nach der Anmeldung können Sie Operationen ausführen, die sich auf die Objekte in der Datenbank beziehen.

PL/SQL in SQL*Plus codieren



ORACLE®

Copyright © 2007, Oracle. Alle Rechte vorbehalten.

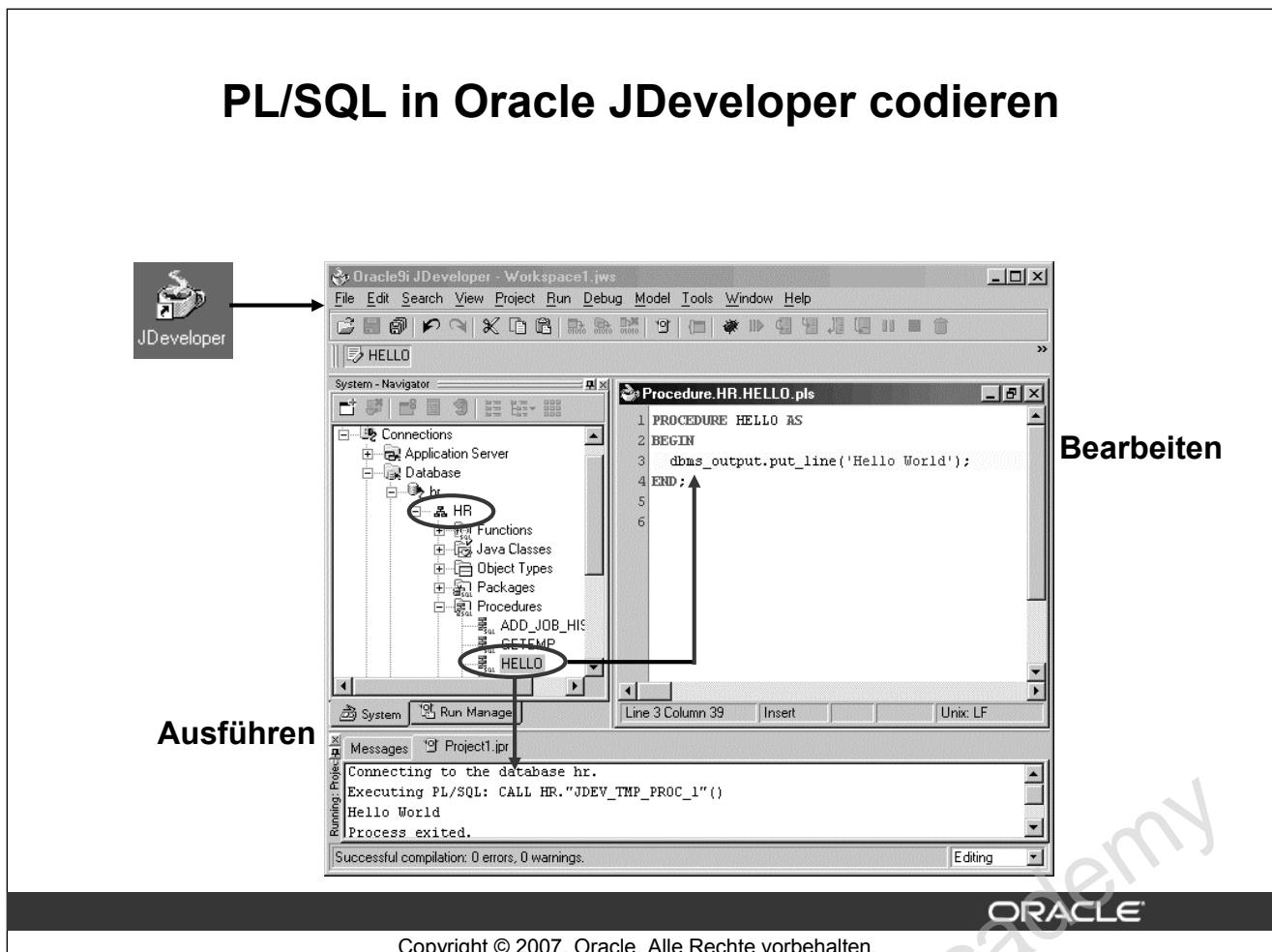
PL/SQL in SQL*Plus codieren

Oracle SQL*Plus ist eine Befehlszeilenanwendung, 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 auf dem Datenbank-Server-System installiert
- wird über ein Symbol oder an der Befehlszeile aufgerufen

PL/SQL in Oracle JDeveloper codieren



PL/SQL in Oracle JDeveloper codieren

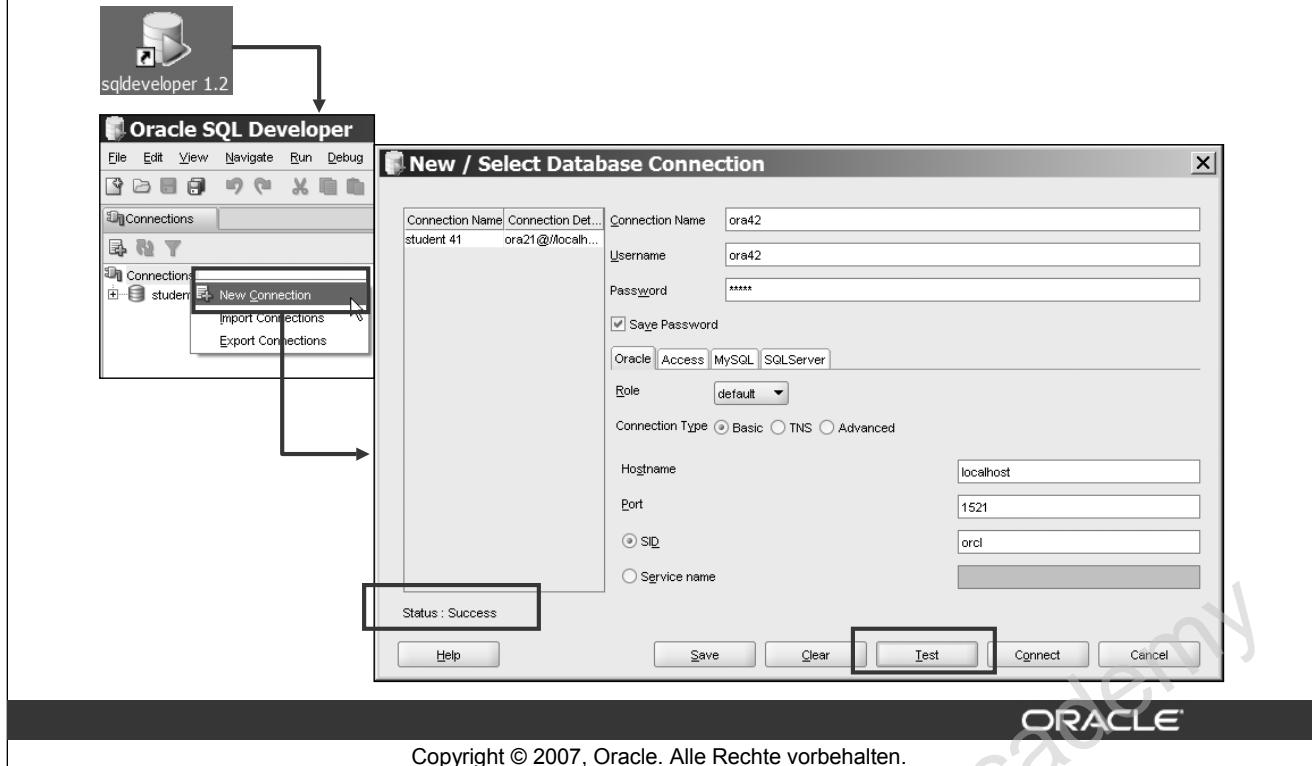
Mit Oracle JDeveloper können Entwickler PL/SQL-Code in einer anspruchsvollen grafischen Benutzeroberfläche (GUI) erstellen, bearbeiten, testen und debuggen. Oracle JDeveloper ist ein Teil der Oracle Developer Suite und auch als separates Produkt erhältlich.

Beachten Sie die folgenden Aspekte, wenn Sie PL/SQL in JDeveloper codieren:

- Sie erstellen zunächst eine Datenbankanmeldung, damit JDeveloper auf einen Datenbankschemaeigentümer für die Unterprogramme zugreifen kann.
- Bei der Datenbankanmeldung können Sie über die Kontextmenüs von JDeveloper Code Editor ein neues Unterprogrammkonstrukt erstellen. JDeveloper Code Editor stellt eine exzellente Umgebung zur PL/SQL-Entwicklung bereit, die die folgenden Features enthält:
 - Verschiedene Farben für die syntaktischen Komponenten der PL/SQL-Sprache
 - Code Insight zur schnellen Suche nach Prozeduren und Funktionen in bereitgestellten Packages
- Mit einem Run-Befehl im Kontextmenü für das benannte Unterprogramm können Sie ein Unterprogramm aufrufen. Die Ausgabe wird im Protokollfenster von JDeveloper angezeigt, wie im unteren Bereich des Screenshots zu sehen.

Hinweis: JDeveloper stellt im JDeveloper Code Editor farbcodierte Syntax zur Verfügung und ist geeignet für PL/SQL-Konstrukte und -Anweisungen.

SQL Developer starten und Datenbankverbindung erstellen



Datenbankverbindung erstellen

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

1. Doppelklicken Sie auf <your_path>\sqldeveloper\sqldeveloper.exe.
2. Klicken Sie in der Registerkarte **Connections** mit der rechten Maustaste auf **Connections**, und wählen Sie **New Connection**.
3. Geben Sie den Verbindungsnamen, den Benutzernamen, das Passwort, den Host-Namen und die SID für die Anmeldung bei der gewünschten Datenbank ein.
4. Um sicherzustellen, dass die Verbindung richtig eingerichtet wurde, klicken Sie auf **Test**.
5. Klicken Sie auf **Connect**.

Geben Sie in der im unteren Bereich des Fensters angezeigten Registerkarte die folgenden Optionen ein:

- **Hostname:** Host-System für die Oracle-Datenbank
- **Port:** Listener Port
- **SID:** Datenbankname
- **Service name:** Netzwerk-Service-Name für eine Remote-Datenbankverbindung

Wenn Sie das Kontrollkästchen **Save Password** aktivieren, wird das Passwort in einer XML-Datei gespeichert. Wenn Sie die SQL Developer-Verbindung schließen und erneut öffnen, müssen Sie das Passwort nicht erneut eingeben.

Schemaobjekte erstellen

- **Mit den folgenden Methoden können Sie in SQL Developer beliebige Schemaobjekte erstellen:**
 - SQL-Anweisungen in SQL Worksheet ausführen
 - Kontextmenü verwenden
- **Bearbeiten Sie Objekte in einem Bearbeitungsdialogfeld oder über eines der vielen Kontextmenüs.**
- **Zeigen Sie die DDL (Data Definition Language, Datendefinitionssprache) für Anpassungen an, z. B. das Erstellen neuer Objekte oder das Bearbeiten vorhandener Schemaobjekte.**

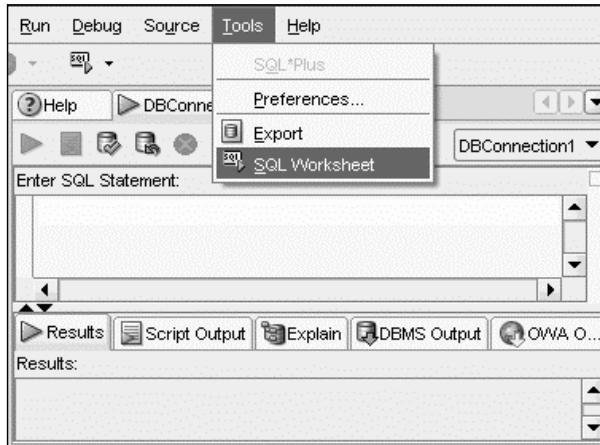
ORACLE®

Copyright © 2007, Oracle. Alle Rechte vorbehalten.

Oracle Internal & Oracle Academy
Use Only

SQL Worksheet

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



Copyright © 2007, Oracle. Alle Rechte vorbehalten.

SQL Worksheet

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

In einem Worksheet können Sie verschiedene Aktionen zur Bearbeitung der Datenbank und Definition der Anmeldung angeben:

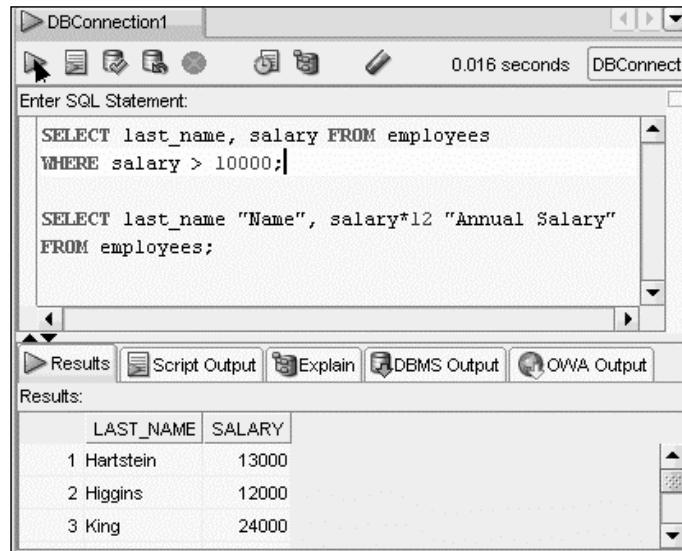
- Tabellen erstellen
- Daten einfügen
- Trigger erstellen und bearbeiten
- Daten aus einer Tabelle wählen
- Gewählte Daten in einer Datei speichern

Sie haben folgende Möglichkeiten, SQL Worksheet anzuzeigen:

- Wählen Sie **Tools > SQL Worksheet**.
- Klicken Sie auf das Symbol **Open SQL Worksheet**.

SQL-Anweisungen ausführen

Im Bereich "Enter SQL Statement" können Sie einzelne oder mehrere SQL-Anweisungen eingeben.



The screenshot shows the Oracle SQL Worksheet window titled "DBConnection1". In the "Enter SQL Statement" pane, two SQL queries are entered:

```
SELECT last_name, salary FROM employees
WHERE salary > 10000;

SELECT last_name "Name", salary*12 "Annual Salary"
FROM employees;
```

Below the statements, the "Results" tab is selected, displaying the output:

LAST_NAME	SALARY
1 Hartstein	13000
2 Higgins	12000
3 King	24000

ORACLE®

Copyright © 2007, Oracle. Alle Rechte vorbehalten.

SQL-Anweisungen ausführen

In SQL Worksheet können Sie im Bereich **Enter SQL Statement** eine einzelne oder mehrere SQL-Anweisungen eingeben. Wenn Sie nur eine Anweisung eingeben, ist das abschließende Semikolon optional.

Beim Eingeben der Anweisung werden die SQL-Schlüsselwörter automatisch hervorgehoben. Um eine SQL-Anweisung auszuführen, stellen Sie sicher, dass sich der Cursor innerhalb der Anweisung befindet. Klicken Sie auf das Symbol **Execute Statement**. Alternativ können Sie die Taste F9 drücken.

Um mehrere SQL-Anweisungen auszuführen und die Ergebnisse anzuzeigen, klicken Sie auf die Schaltfläche **Run Script**. Alternativ können Sie die Taste F5 drücken.

Da im Beispiel auf der Folie mehrere SQL-Anweisungen eingegeben wurden, ist die erste Anweisung mit einem Semikolon abgeschlossen. Der Cursor befindet sich in der ersten Anweisung. Daher werden bei Ausführung der Anweisung im Feld **Results** nur die Ergebnisse der ersten Anweisung angezeigt.

Zusammenfassung

In dieser Lektion haben Sie Folgendes gelernt:

- Kursaufbau beschreiben
- HR-Beispielschema beschreiben
- Verfügbare Entwicklungsumgebungen auflisten
- Grundlegende Funktionen von SQL Developer prüfen

ORACLE®

Copyright © 2007, Oracle. Alle Rechte vorbehalten.

Oracle Internal & Oracle Academy
Use Only

Kursübungen

In den Übungen zu diesem Kurs entwickeln Sie eine einfache Anwendung. Sie verwenden anonyme Blöcke, die folgende Themen behandeln:

- Deklarativen Bereich erstellen
- Variablen skalarer Datentypen deklarieren
- Variablen mit dem %TYPE-Attribut deklarieren
- Ausführbaren Bereich schreiben
- Benutzereingaben für Variablen akzeptieren
- Werte aus der Datenbank abrufen und mit Hilfe der INTO-Klausel in den Variablen speichern
- Verschachtelten Block im ausführbaren Bereich erstellen
- Mit den Kontrollstrukturen im ausführbaren Bereich Geschäftslogik ausführen
- Werte in der INDEX BY-Tabelle speichern und ausgeben
- Exceptions behandeln

1

Einführung in PL/SQL

ORACLE®

Copyright © 2007, Oracle. Alle Rechte vorbehalten.

Oracle Internal & Oracle Academy
Use Only

Ziele

Nach Ablauf dieser Lektion haben Sie folgende Ziele erreicht:

- **Notwendigkeit für PL/SQL beschreiben**
- **Vorteile von PL/SQL beschreiben**
- **Verschiedene Typen von PL/SQL-Blöcken angeben**
- **Ausgabemeldungen in PL/SQL**

ORACLE®

Copyright © 2007, Oracle. Alle Rechte vorbehalten.

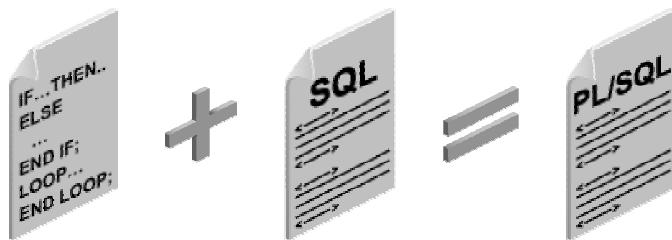
Lernziel

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

PL/SQL

PL/SQL:

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



ORACLE

Copyright © 2007, Oracle. Alle Rechte vorbehalten.

PL/SQL

Structured Query Language (SQL) ist die primäre Sprache für den Zugriff auf und die Bearbeitung von Daten in relationalen Datenbanken. Es gibt nur wenige SQL-Befehle. Sie 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 mit Hilfe von Bedingungen abgerufen werden, stoßen Sie rasch an die Grenzen von SQL. Dazu die folgende leicht abgewandelte Aufgabenstellung: Für jeden abgerufenen Mitarbeiter soll 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 gezeigte SQL-Anweisung ausführen, die Daten sammeln und Logik auf die Daten anwenden. Sie haben die Möglichkeit, 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. Bedingungsanweisungen würden die Aufgabe vereinfachen. PL/SQL ist für Anforderungen dieser Art konzipiert. Die Sprache bietet eine Programmiererweiterung für bereits vorhandenes SQL.

PL/SQL

PL/SQL:

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

ORACLE®

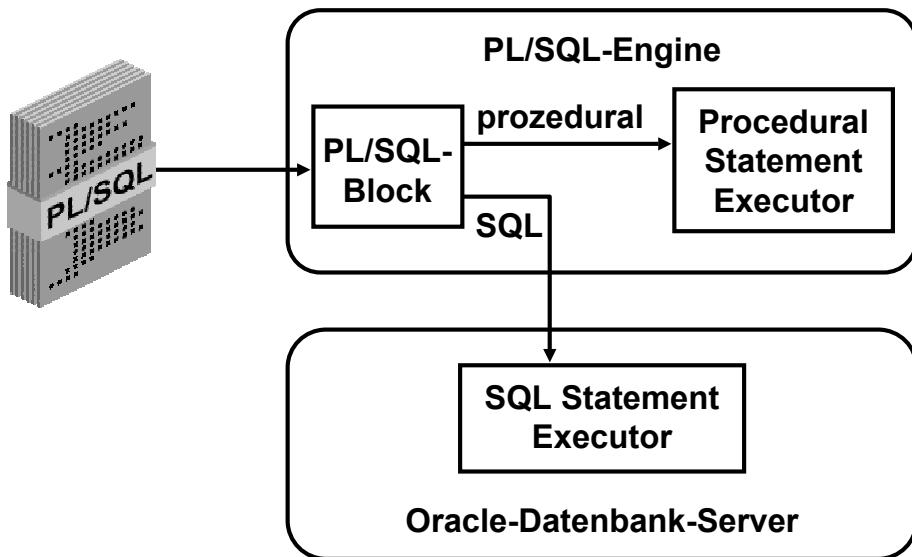
Copyright © 2007, Oracle. Alle Rechte vorbehalten.

PL/SQL (Fortsetzung)

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

PL/SQL stellt Features des modernen Software Engineerings zur Verfügung, zum Beispiel Datenkapselung, Exception-Behandlung, Information Hiding sowie objektorientiertes Programmieren. 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-Umgebung



ORACLE

Copyright © 2007, Oracle. Alle Rechte vorbehalten.

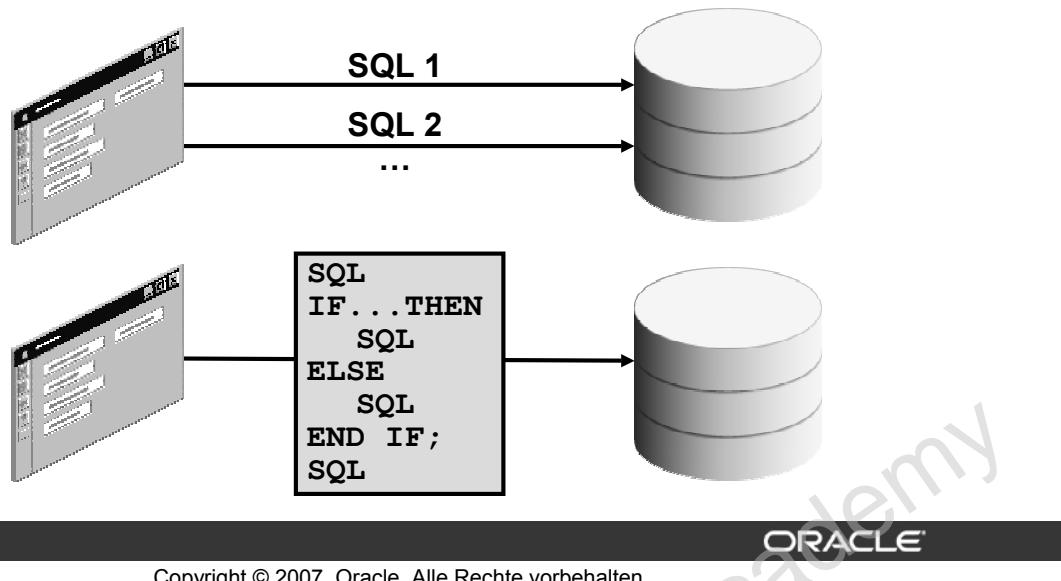
PL/SQL-Umgebung

Auf der Folie wird die PL/SQL-Ausführungsumgebung im Oracle-Datenbank-Server dargestellt. Ein PL/SQL-Block enthält prozedurale Anweisungen und SQL-Anweisungen. Wenn Sie den PL/SQL-Block an den Server weiterleiten, parst die PL/SQL-Engine zunächst den Block. Die PL/SQL-Engine identifiziert die prozeduralen Anweisungen und die SQL-Anweisungen. Sie übergibt die prozeduralen Anweisungen an den Procedural Statement Executor und die SQL-Anweisungen jeweils einzeln an den SQL Statement Executor.

Das Diagramm auf der Folie zeigt die PL/SQL-Engine innerhalb des Datenbank-Servers. Die Oracle-Tools für die Anwendungsentwicklung können ebenfalls eine PL/SQL-Engine enthalten. Das Tool übergibt die Blöcke an die eigene lokale PL/SQL-Engine. Daher werden alle prozeduralen Anweisungen lokal und nur die SQL-Anweisungen in der Datenbank ausgeführt. Welche Engine jeweils verwendet wird, hängt davon ab, von wo aus der PL/SQL-Block aufgerufen wurde.

PL/SQL – Vorteile

- **Integration prozeduraler Konstrukte in SQL**
- **Bessere Performance**



Copyright © 2007, Oracle. Alle Rechte vorbehalten.

ORACLE

PL/SQL – Vorteile

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 Datenbank-Server mit, *was* er tun soll. Sie können jedoch nicht angeben, *wie* er die Aufgabe ausführen soll. PL/SQL integriert Kontrollanweisungen und Bedingungsanweisungen in SQL. Sie haben eine bessere Kontrolle über Ihre SQL-Anweisungen und ihre 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 die Daten weiterleitet, müssen Sie möglicherweise eine Reihe von SQL-Anweisungen ausführen. 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 einem einzigen Programmblöck 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 mit Hilfe von PL/SQL-Blöcken gruppieren, bevor sie zur Ausführung an den Oracle-Datenbank-Server gesendet werden.

PL/SQL – Vorteile

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

ORACLE

Copyright © 2007, Oracle. Alle Rechte vorbehalten.

PL/SQL – Vorteile (Fortsetzung)

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 mit Hilfe von Prozeduren, Funktionen und Packages implementiert. Sie werden in der Lektion "Stored Procedures und Stored Functions erstellen" erläutert.

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

PL/SQL – Vorteile (Fortsetzung)

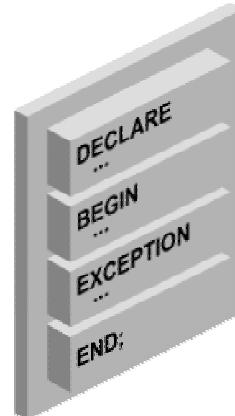
Portierbarkeit: PL/SQL-Programme lassen sich überall dort ausführen, wo auch der Oracle-Server ausgeführt wird, ungeachtet des Betriebssystems oder der Plattform. Sie müssen die Programme nicht an jede neue Umgebung anpassen. Sie können portierbare Programm-Packages und Librarys erstellen, die in unterschiedlichen 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 (erforderlich)**
 - SQL-Anweisungen
 - PL/SQL-Anweisungen
- **EXCEPTION (optional)**
 - Aktionen, die durchgeführt werden sollen, wenn Fehler auftreten
- **END; (erforderlich)**



ORACLE®

Copyright © 2007, Oracle. Alle Rechte vorbehalten.

PL/SQL-Blockstruktur

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

- **Deklarativer Bereich (optional):** Der deklarative Bereich beginnt mit dem DECLARE-Schlüsselwort und endet mit Beginn des ausführbaren Bereichs.
- **Ausführbarer Bereich (obligatorisch):** Der ausführbare Bereich beginnt mit dem BEGIN-Schlüsselwort und endet mit END. Der 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.
- **Abschnitt für Exception-Behandlung (optional):** Der Exception-Abschnitt ist im ausführbaren Bereich verschachtelt. Er beginnt mit dem EXCEPTION-Schlüsselwort.

PL/SQL-Blockstruktur (Fortsetzung)

In einem PL/SQL-Block werden die Schlüsselwörter DECLARE, BEGIN und EXCEPTION nicht mit einem Semikolon abgeschlossen. Allerdings müssen Sie das END-Schlüsselwort und alle SQL- sowie PL/SQL-Anweisungen mit einem Semikolon abschließen.

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

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 © 2007, Oracle. Alle Rechte vorbehalten.

Blocktypen

Ein PL/SQL-Programm besteht aus einem oder mehreren Blöcken. Die Blöcke können klar voneinander getrennt oder ineinander verschachtelt sein. Drei Typen von Blöcken bilden ein PL/SQL-Programm:

- 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, um zur Laufzeit ausgeführt zu werden. Die Trigger in Oracle Developer-Komponenten bestehen aus solchen Blöcken. Die anonymen Blöcke werden zur Laufzeit ausgeführt, da sie inline angegeben sind. 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.

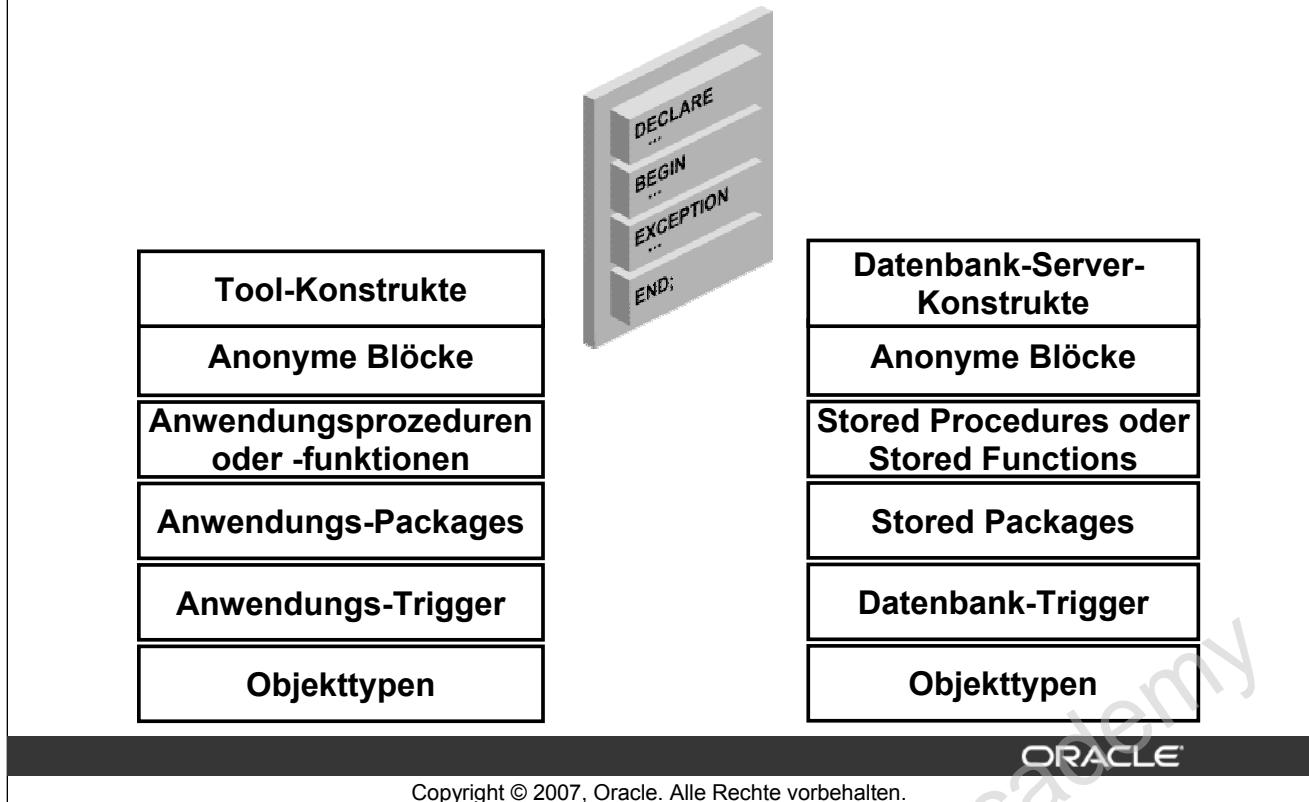
Blocktypen (Fortsetzung)

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, um Aktionen durchzuführen und Funktionen, um Werte zu berechnen und zurückzugeben.

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 Trigger innerhalb derselben Anwendung aufrufen, wann immer erforderlich.

Hinweis: Funktionen sind mit Prozeduren vergleichbar. Sie *müssen* jedoch einen Wert zurückgeben.

Programmkonstrukte



Copyright © 2007, Oracle. Alle Rechte vorbehalten.

Programmkonstrukte

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

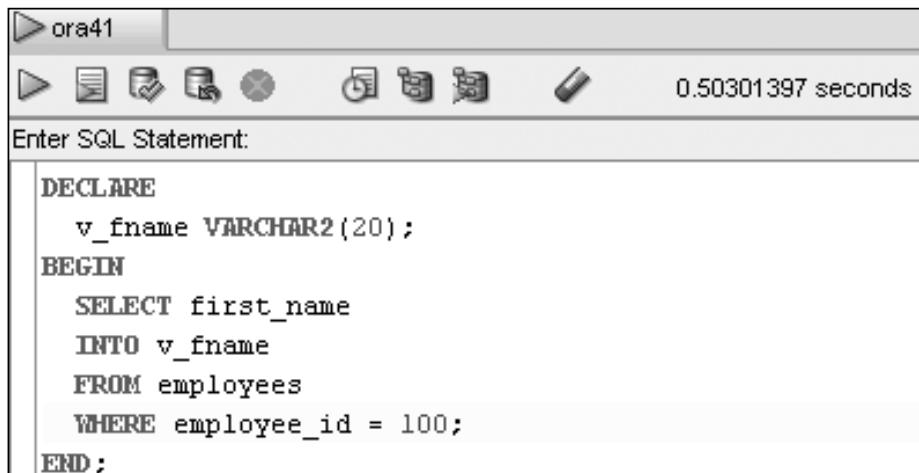
Programmkonstrukt	Beschreibung	Verfügbarkeit
Anonyme Blöcke	Unbenannte PL/SQL-Blöcke, die in eine Anwendung eingebettet sind oder interaktiv ausgegeben werden.	In allen PL/SQL-Umgebungen
Anwendungsprozeduren oder -funktionen	Benannte PL/SQL-Blöcke, die in einer Oracle Forms Developer-Anwendung oder Shared Library gespeichert sind. Sie können Parameter annehmen und wiederholt namentlich 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 wiederholt namentlich 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 Identifier gruppiert sind.	Im Oracle-Server und in Komponenten von Oracle Developer-Tools, beispielsweise Oracle Forms Developer

Programmkonstrukte (Fortsetzung)

Programmkonstrukt	Beschreibung	Verfügbarkeit
Datenbank-Trigger	PL/SQL-Blöcke, die einer Datenbanktabelle zugeordnet sind und durch verschiedene Ereignisse automatisch ausgelöst werden.	Im Oracle-Server oder in jedem Oracle-Tool, das die DML absetzt
Anwendungs-Trigger	PL/SQL-Blöcke, die entweder einer Datenbanktabelle oder Systemereignissen zugeordnet sind. Sie werden automatisch durch ein DML- beziehungsweise Systemereignis ausgelöst.	In Oracle Developer-Tools, beispielsweise Oracle Forms Developer
Objekttypen	Benutzerdefinierte, zusammengesetzte Datentypen, die eine Datenstruktur mit den für die Datenmanipulation notwendigen Funktionen und Prozeduren kapseln.	Im Oracle-Server und in Oracle Developer-Tools

Anonyme Blöcke erstellen

Anonymen Block im SQL Developer Workspace eingeben:



The screenshot shows the Oracle SQL Developer interface. The title bar says "ora41". Below it is a toolbar with various icons. The main area is titled "Enter SQL Statement:" and contains the following PL/SQL code:

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

ORACLE

Copyright © 2007, Oracle. Alle Rechte vorbehalten.

Anonyme Blöcke erstellen

Um einen anonymen Block mit SQL Developer zu erstellen, geben Sie den Block im Workspace ein, wie auf der Folie gezeigt. Der Block hat den deklarativen und ausführbaren Bereich. 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 einer Variablen namens `f_name`.

Anonyme Blöcke ausführen

Anonymen Block durch Klicken auf die Schaltfläche "Run Script" ausführen:

The screenshot shows the Oracle SQL Developer interface. In the top bar, there is a progress indicator showing "0.50301397 seconds". Below the bar, the "Enter SQL Statement:" field contains the following PL/SQL code:

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

A callout box points to the "Run Script" button in the toolbar, which is highlighted. The "Script Output" tab is selected in the bottom navigation bar, and it displays the message "anonymous block completed".

ORACLE

Copyright © 2007, Oracle. Alle Rechte vorbehalten.

Anonyme Blöcke ausführen

Klicken Sie auf die Schaltfläche **Run Script**, um den anonymen Block im Workspace auszuführen. Beachten Sie, dass die Meldung **anonymous block completed** im Fenster **Script Output** angezeigt wird, nachdem der Block ausgeführt wurde.

Ausgabe von PL/SQL-Blöcken testen

- Ausgabe in SQL Developer durch Klicken auf die Schaltfläche "Enable DBMS Output" in der Registerkarte "DBMS Output" aktivieren:



- Vordefiniertes Oracle-Package und seine Prozeduren verwenden:

- DBMS_OUTPUT.PUT_LINE

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

ORACLE®

Copyright © 2007, Oracle. Alle Rechte vorbehalten.

Ausgabe von PL/SQL-Blöcken testen

Im Beispiel auf der vorherigen Folie wurde ein Wert in der f_name-Variablen gespeichert. Der Wert wurde jedoch nicht ausgegeben. In diesem Abschnitt wird erläutert, wie Sie den Wert ausgeben. PL/SQL verfügt über keine integrierte Eingabe- oder Ausgabefunktionalität. Daher werden vordefinierte Oracle-Packages für die Ein- und Ausgabe verwendet. Um eine Ausgabe zu generieren, gehen Sie wie folgt vor:

- Aktivieren Sie die Ausgabe in SQL Developer. Klicken Sie hierfür auf die Schaltfläche **Enable DBMS Output** in der Registerkarte **DBMS Output**. Dadurch wird der Befehl SET SERVEROUTPUT ON ausgeführt, der im Fenster angezeigt wird. Um die Ausgabe in SQL*Plus zu aktivieren, müssen Sie den Befehl SET SERVEROUTPUT ON explizit absetzen.
- Zeigen Sie die Ausgabe mit der PUT_LINE-Prozedur des DBMS_OUTPUT-Packages an. Übergeben Sie den auszugebenden Wert als Argument an diese Prozedur, wie auf der Folie gezeigt. Die Prozedur gibt dann die Argumente aus.

Ausgabe von PL/SQL-Blöcken testen

The screenshot shows the Oracle SQL Developer interface. In the center, there's a large text area labeled "Enter SQL Statement:" containing the following PL/SQL code:

```
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');
END;
```

Below the code, the "Results" tab is selected in the toolbar, and the output window displays the results:

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

The status bar at the bottom indicates "Script Finished".

ORACLE

Copyright © 2007, Oracle. Alle Rechte vorbehalten.

Ausgabe von PL/SQL-Blöcken testen (Fortsetzung)

Die Folie zeigt die Ausgabe des PL/SQL-Blockes, nachdem Code zur Generierung der Ausgabe hinzugefügt wurde.

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**

ORACLE®

Copyright © 2007, Oracle. Alle Rechte vorbehalten.

Zusammenfassung

PL/SQL ist eine Programmiersprache mit Features, die die Funktionalität von SQL erweitern. Die nicht prozedurale Sprache SQL wird durch PL/SQL-Programmkonstrukte 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 1 – Überblick

Die Übungen behandeln folgende Themen:

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

ORACLE

Copyright © 2007, Oracle. Alle Rechte vorbehalten.

Übungen zu Lektion 1 – Überblick

Die Übungen vertiefen die in dieser Lektion erlernten Grundlagen von PL/SQL.

- In der nicht am Rechner auszuführenden 1. Übung geben Sie PL/SQL-Blöcke an, die erfolgreich ausgeführt werden.
- In der 2. Übung erstellen Sie einen einfachen PL/SQL-Block und führen ihn aus.

Übungen zu Lektion 1

Ihr Arbeitsverzeichnis ist der Ordner `labs`. Sie können Ihre Skripts in den Ordner `labs` speichern. Bitten Sie Ihren Dozenten, Ihnen den Pfad zum Ordner `labs` für diesen Kurs mitzuteilen. Die Lösungen der Übungen befinden sich im Ordner `soln`.

1. Welche der folgenden PL/SQL-Blöcke werden erfolgreich ausgeführt?

a. BEGIN

END;

b. DECLARE

amount INTEGER(10);

END;

c. DECLARE

BEGIN

END;

d. DECLARE

amount INTEGER(10);

BEGIN

DBMS_OUTPUT.PUT_LINE(amount);

END;

2. Erstellen Sie einen einfachen anonymen Block, der den Text "Hello World" ausgibt, und führen Sie ihn aus. Führen Sie das Skript aus, und speichern Sie es unter dem Namen `lab_01_02_soln.sql`.

PL/SQL-Variablen deklarieren

2

ORACLE®

Copyright © 2007, Oracle. Alle Rechte vorbehalten.

Oracle Internal & Oracle Academy
Use Only

Ziele

Nach Ablauf dieser Lektion haben Sie folgende Ziele erreicht:

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

ORACLE®

Copyright © 2007, Oracle. Alle Rechte vorbehalten.

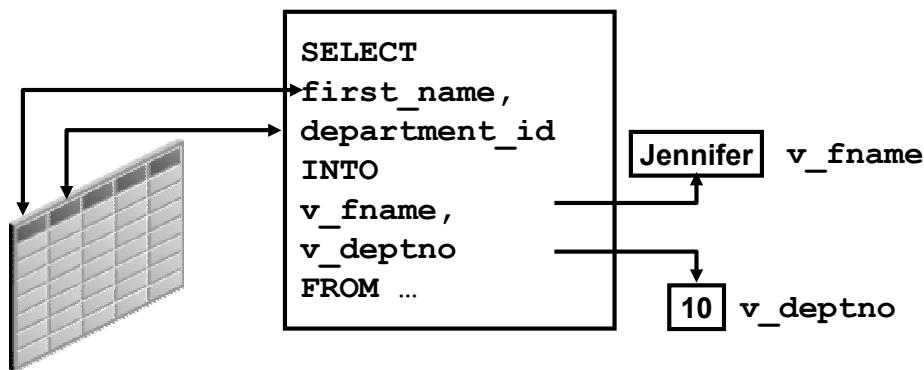
Lernziel

Sie haben bereits einfache PL/SQL-Blöcke und deren Bereiche kennen gelernt. In dieser Lektion werden gültige und ungültige Identifier 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 %TYPE-Attribut und dessen Vorteile kennen.

Variablen verwenden

Variablen können für folgende Zwecke verwendet werden:

- Daten temporär speichern
- Gespeicherte Werte bearbeiten
- Wiederverwendbarkeit



ORACLE

Copyright © 2007, Oracle. Alle Rechte vorbehalten.

Variablen verwenden

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 den Vornamen oder die Abteilungsnummer bearbeiten möchten, müssen Sie den abgerufenen Wert speichern. Um den Wert temporär zu speichern, werden Variablen verwendet. Mit dem in diesen Variablen gespeicherten Wert können Sie die Daten ver- und bearbeiten. Variablen können jedes beliebige PL/SQL-Objekt speichern, zum Beispiel Variablen, Typen, Cursor und Unterprogramme.

Wiederverwendbarkeit ist ein weiterer Vorteil des Deklarierens von Variablen. Nachdem Sie die Variablen deklariert haben, können Sie sie mehrfach in einer Anwendung verwenden, indem Sie in verschiedenen Anweisungen wiederholt auf sie verweisen.

Anforderungen für Variablennamen

Variablennamen:

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



ORACLE®

Copyright © 2007, Oracle. Alle Rechte vorbehalten.

Anforderungen für Variablennamen

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

Variablen in PL/SQL verwenden

Variablen werden:

- im deklarativen Bereich deklariert und 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 © 2007, Oracle. Alle Rechte vorbehalten.

Variablen in PL/SQL verwenden

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 Speicherplatz, Datentyp und Speicherort für einen Wert an, damit Sie auf ihn verweisen 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 in anderen Anweisungen (auch anderen deklarativen Anweisungen) darauf verweisen können.

Im ausführbaren Bereich verwenden und neue Werte zuweisen: Sie können den vorhandenen Wert der Variablen im ausführbaren Bereich durch den 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_deptno        NUMBER(2) NOT NULL := 10;
    v_location       VARCHAR2(13) := 'Atlanta';
    c_comm           CONSTANT NUMBER := 1400;
```

ORACLE

Copyright © 2007, Oracle. Alle Rechte vorbehalten.

PL/SQL-Variablen deklarieren und initialisieren

Sie müssen alle PL/SQL-Identifier im deklarativen Bereich deklarieren, um sie im PL/SQL-Block referenzieren zu können. Sie haben die Möglichkeit, einer Variablen einen Ausgangswert zuzuweisen (wie auf der Folie gezeigt). Sie müssen Variablen jedoch keinen Wert zuweisen, um sie deklarieren zu können. 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
CONSTANT	Legt die Variable als Konstante fest, so dass ihr Wert nicht geändert werden kann. (Konstanten müssen initialisiert werden.)
<i>data type</i>	Steht für einen skalaren, zusammengesetzten, Referenz- oder LOB-Datentyp. (In diesem Kurs werden nur skalare, zusammengesetzte und LOB-Datentypen behandelt.)
NOT NULL	Legt fest, dass die Variable einen Wert enthalten muss. (NOT NULL-Variablen müssen initialisiert werden.)
<i>expr</i>	Steht für einen PL/SQL-Ausdruck und 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

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

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

ORACLE

Copyright © 2007, Oracle. Alle Rechte vorbehalten.

PL/SQL-Variablen deklarieren und initialisieren (Fortsetzung)

Untersuchen Sie die beiden Code-Blöcke auf der Folie.

1. Die v_myName-Variable ist im deklarativen Bereich des Blockes deklariert. Sie können auf diese Variable im ausführbaren Bereich desselben Blockes zugreifen. Im ausführbaren Bereich wird der Variablen der Wert John zugewiesen. Zeichenfolgenliterale müssen in einfache Anführungszeichen gesetzt werden. Wenn die Zeichenfolge ein einfaches Anführungszeichen enthält, wie in "Today's Date", muss sie in der Form "Today"s Date" (zwei einfache Anführungszeichen zwischen "y" und "s") angegeben werden. ":" ist der Zuweisungsoperator. Die PUT_LINE-Prozedur rufen Sie auf, indem Sie die v_myName-Variable übergeben. Der Wert der Variablen ist mit der Zeichenfolge 'My name is:' verknüpft. Die Ausgabe dieses anonymen Blockes lautet:

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

2. Im zweiten Block wird die v_myName-Variable im deklarativen Bereich deklariert und initialisiert. Nach der Initialisierung enthält v_myName den Wert John. Sie können diesen Wert im ausführbaren Bereich des Blockes bearbeiten. Die Ausgabe dieses anonymen Blockes lautet:

```
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;
/
```

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

ORACLE

Copyright © 2007, Oracle. Alle Rechte vorbehalten.

Begrenzungszeichen in Zeichenfolgenliteralen

Wenn die Zeichenfolge einen Apostroph enthält (mit einem einfachen Anführungszeichen identisch), 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 führt zu einer komplexen Zeichenfolge, insbesondere, wenn Sie SQL-Anweisungen als Zeichenfolgen verwenden. Sie können daher jedes Zeichen, das nicht in der Zeichenfolge enthalten ist, als Begrenzungszeichen angeben. Die Folie zeigt, wie Sie das Begrenzungszeichen mit Hilfe der q'-Notation angeben. Im Beispiel werden als Begrenzungszeichen Ausrufezeichen (!) und eckige Klammern ([]) verwendet. Betrachten Sie folgendes Beispiel:

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

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

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

Variablenarten

- **PL/SQL-Variablen:**
 - Skalar
 - Zusammengesetzt
 - Referenz
 - Large Object (LOB)
- **Nicht-PL/SQL-Variablen: Bind-Variablen**

ORACLE

Copyright © 2007, Oracle. Alle Rechte vorbehalten.

Variablenarten

Jede PL/SQL-Variable gehört einem Datentyp an, der Speicherformat, Constraints und den gültigen Wertebereich festlegt. PL/SQL unterstützt fünf Kategorien von Datentypen, mit denen Sie Variablen, Konstanten und Zeiger deklarieren können: skalar, zusammengesetzt, Referenz, Large Objekt (LOB) und Objekt.

- **Skalare Datentypen:** Skalare Datentypen speichern einen einzelnen Wert. Der Wert hängt vom Datentyp der Variablen ab. Beispiel: Die `v_myName`-Variable, die im Abschnitt "PL/SQL-Variablen deklarieren und initialisieren" dieser Lektion als Beispiel verwendet wird, hat den Typ `VARCHAR2`. Sie kann daher einen Zeichenfolgenwert speichern. PL/SQL unterstützt auch boolesche Variablen.
- **Zusammengesetzte Datentypen:** Zusammengesetzte Datentypen enthalten interne skalare oder zusammengesetzte Elemente. Beispiele für zusammengesetzte Datentypen sind `RECORD` und `TABLE`.
- **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 (z. B. Grafiken) an, die außerhalb der Tabelle gespeichert sind.

Variablenarten (Fortsetzung)

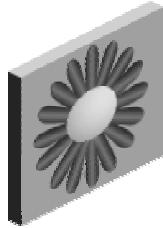
Zu Variablen, die keine PL/SQL-Variablen sind, gehören in Precompiler-Programmen deklarierte Variablen der Host-Sprache, Maskenfelder in Forms-Anwendungen sowie Host-Variablen. Im weiteren Verlauf dieser Lektion werden die Host-Variablen näher erläutert.

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

Oracle Internal & Oracle Academy
Use Only

Variablenarten

TRUE



25-JAN-01

Schneewittchen

Es war einmal
in einem fernen Land,
da lebte eine Prinzessin
namens Schneewittchen...

256120.08



ORACLE

Copyright © 2007, Oracle. Alle Rechte vorbehalten.

Variablenarten (Fortsetzung)

Die Folie enthält Beispiele für die folgenden Datentypen:

- TRUE steht für einen booleschen Wert.
- 25-JAN-01 steht für den DATE-Datentyp.
- Das Bild steht für ein BLOB.
- Der Text in der Sprechblase steht für den VARCHAR2-Datentyp oder ein CLOB.
- 256120.08 steht für den NUMBER-Datentyp mit Dezimalstellen.
- Die Filmrolle steht für ein BFILE.
- Der Ortsname *Atlanta* steht für den VARCHAR2-Datentyp.

PL/SQL-Variablen deklarieren und initialisieren – Richtlinien

- Beachten Sie die Benennungskonventionen.
- Verwenden Sie aussagekräftige Identifier für Variablen.
- Initialisieren Sie als NOT NULL und CONSTANT definierte Variablen.
- Initialisieren Sie Variablen mit dem Zuweisungsoperator (:=) oder dem DEFAULT-Schlüsselwort:

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

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

- Deklarieren Sie zur besseren Lesbarkeit und Code-Verwaltung nur einen Identifier pro Zeile.

ORACLE

Copyright © 2007, Oracle. Alle Rechte vorbehalten.

PL/SQL-Variablen deklarieren und initialisieren – Richtlinien

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

- Halten Sie die Benennungskonventionen ein. Verwenden Sie z. B. name für Variablen und c_name für Konstanten. Entsprechend können Sie Variablen mit v_fname benennen.
- Verwenden Sie aussagekräftige und geeignete Identifier für Variablen. Beispiel: Sie können anstelle von salary1 und salary2 auch salary und sal_with_commission verwenden.
- Wenn Sie das NOT NULL-Constraint verwenden, müssen Sie beim Deklarieren der Variablen einen Wert zuweisen.
- Beim Deklarieren von Konstanten muss der Typspezifikation das CONSTANT-Schlüsselwort vorangestellt werden. Die folgende Deklaration benennt eine Konstante des NUMBER-Typs 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

- Verwenden Sie keine Spaltennamen als Identifier.

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



- Verwenden Sie das NOT NULL-Constraint, wenn die Variable einen Wert speichern muss.

ORACLE

Copyright © 2007, Oracle. Alle Rechte vorbehalten.

PL/SQL-Variablen deklarieren – Richtlinien

- Initialisieren Sie die Variable mit Hilfe des Zuweisungsoperators (:=) oder des reservierten Wortes DEFAULT mit einem Ausdruck. Wenn Sie keinen Ausgangswert zuweisen, enthält die neue Variable standardmäßig NULL, bis Sie einen Wert zuweisen. Um einer Variablen einen neuen oder bereits zugewiesenen Wert zuzuweisen, erstellen Sie eine PL/SQL-Zuweisungsanweisung. Das Initialisieren aller Variablen ist eine gängige und sinnvolle Vorgehensweise bei der Programmierung.
- Gleiche Objektnamen sind zulässig, vorausgesetzt, die Objekte sind in unterschiedlichen Blöcken definiert. Objekte mit gleichen Objektnamen können Sie mit Labels kennzeichnen.
- Verwenden Sie keine Spaltennamen als Identifier. Der Oracle-Server sieht PL/SQL-Variablen in SQL-Anweisungen, die denselben Namen wie eine Spalte haben, als die referenzierte Spalte an. Obwohl das Code-Beispiel 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 als NOT NULL definierten Variablen keine NULL-Werte zuweisen. Dem NOT NULL-Constraint muss eine Initialisierungsklausel folgen.

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

Skalare Datentypen

- Speichern einen einzelnen Wert
- Haben keine internen Komponenten

TRUE

25-JAN-01

256120.08

Atlanta

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

ORACLE®

Copyright © 2007, Oracle. Alle Rechte vorbehalten.

Skalare Datentypen

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 Basistypen beschrieben, die häufig in PL/SQL-Programmen verwendet werden.

Skalare Datentypen speichern einen einzelnen Wert und haben 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 *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 © 2007, Oracle. Alle Rechte vorbehalten.

Skalare Basisdatentypen

Datentyp	Beschreibung
CHAR [(maximum_length)]	Basistyp für Daten mit Zeichenfolgen fester Länge bis 32.767 Byte. Wenn Sie keine maximale Länge angeben, wird als Default-Länge 1 festgelegt.
VARCHAR2 (maximum_length)	Basistyp für Daten mit Zeichenfolgen variabler Länge bis 32.767 Byte. Für VARCHAR2-Variablen und -Konstanten gibt es keine Default-Größe.
NUMBER [(precision, scale)]	Ziffern mit einer festgelegten Anzahl von Nachkommastellen (<i>scale</i>) und Gesamtstellen (<i>precision</i>). 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.

Skalare Basisdatentypen (Fortsetzung)

Datentyp	Beschreibung
PLS_INTEGER	Basistyp für Ganzzahlen mit Vorzeichen zwischen -2.147.483.647 und 2.147.483.647. PLS_INTEGER-Werte erfordern weniger Speicherplatz und lassen sich schneller verarbeiten als Werte des NUMBER-Typs. In Oracle Database 10g sind die Datentypen PLS_INTEGER und BINARY_INTEGER identisch. Die arithmetischen Operationen mit Werten für PLS_INTEGER und BINARY_INTEGER lassen sich schneller verarbeiten als 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 © 2007, Oracle. Alle Rechte vorbehalten.

Skalare Basisdatentypen (Fortsetzung)

Datentyp	Beschreibung
DATE	Basistyp für Datum und Uhrzeit. DATE-Werte beinhalten die Tageszeit ab Mitternacht in Sekunden. Der Datumsbereich liegt zwischen 4712 v. Chr. und 9999 n. Chr.
TIMESTAMP	Der TIMESTAMP-Datentyp, eine Erweiterung von DATE, speichert Jahr, Monat, Tag, Stunde, Minute, Sekunde und Sekundenbruchteil. Die Syntax lautet <code>TIMESTAMP [(precision)]</code> , wobei der optionale <code>precision</code> -Parameter die Anzahl der Stellen im Bruchteilbereich des Sekundenfeldes angibt. Für die Gesamtstelligenzahl müssen Sie eine Ganzzahl zwischen 0 und 9 verwenden. Der Default ist 6.
TIMESTAMP WITH TIME ZONE	Der Datentyp TIMESTAMP WITH TIME ZONE, eine Erweiterung von TIMESTAMP, beinhaltet eine Zeitzonendifferenz. Die Zeitzonendifferenz 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 TIME ZONE</code> , wobei der optionale <code>precision</code> -Parameter die Anzahl der Stellen im Bruchteilbereich des Sekundenfeldes angibt. Für die Gesamtstelligenzahl müssen Sie eine Ganzzahl zwischen 0 und 9 verwenden. Der Default ist 6.

Skalare Basisdatentypen (Fortsetzung)

Datentyp	Beschreibung
TIMESTAMP WITH LOCAL TIME ZONE	<p>Der Datentyp TIMESTAMP WITH LOCAL TIME ZONE, eine Erweiterung von TIMESTAMP, beinhaltet eine Zeitzonendifferenz. Die Zeitzonendifferenz 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 <code>precision</code>-Parameter die Anzahl der Stellen im Bruchteilbereich des Sekundenfeldes angibt. Sie können die Gesamtstelligenzahl nicht mit einer symbolischen Konstanten oder Variablen angeben, sondern müssen eine Ganzzahl zwischen 0 und 9 verwenden. Der Default ist 6.</p> <p>Der Unterschied zum Datentyp TIMESTAMP WITH TIME ZONE liegt darin, dass in Datenbankspalten eingegebene Zeitzonewerte 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 Anzahl der Stellen im Feld für das Jahr angibt. Sie können die Gesamtstelligenzahl nicht mit einer symbolischen Konstanten oder Variablen angeben, sondern müssen eine Ganzzahl zwischen 0 und 4 verwenden. Der Default ist 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> die Anzahl der Stellen im Feld für das Jahr angibt. Sie können in beiden Fällen die Gesamtstelligenzahl nicht mit einer symbolischen Konstanten oder Variablen angeben, sondern müssen ein Ganzzahlliteral zwischen 0 und 9 verwenden. Die Default-Werte sind 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 © 2007, Oracle. Alle Rechte vorbehalten.

Skalare Variablen deklarieren

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

- **v_emp_job:** Variable, die die Job-Kennung eines Mitarbeiters speichert
- **v_count_loop:** Variable, die die Iterationen einer Schleife zählt und auf 0 initialisiert wird
- **v_dept_total_sal:** Variable, die die Gesamtsumme der Gehälter einer Abteilung berechnet und auf 0 initialisiert wird
- **v_orderdate:** Variable, die das Lieferdatum eines Auftrags speichert und auf eine Woche nach dem aktuellen Datum initialisiert wird
- **c_tax_rate:** Konstante für den Steuersatz, der sich im gesamten PL/SQL-Block nie ändert und auf 8.25 festgelegt ist.
- **v_valid:** Flag, das angibt, ob eine Dateneingabe gültig oder ungültig ist, und auf TRUE initialisiert wird

%TYPE-Attribut

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

ORACLE

Copyright © 2007, Oracle. Alle Rechte vorbehalten.

%TYPE-Attribut

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 Gesamtstellenzahl haben. 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 Gesamtstellenzahl von Variablen hartzudcodieren, können Sie Variablen mithilfe des %TYPE-Attributs entsprechend einer zuvor deklarierten Variablen oder Datenbankspalte deklarieren. Das %TYPE-Attribut wird am häufigsten verwendet, wenn der in der Variablen gespeicherte Wert von einer Tabelle in der Datenbank abgeleitet wird. Wenn Sie das %TYPE-Attribut verwenden, um eine Variable zu deklarieren, müssen Sie ihr 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.

%TYPE-Attribut (Fortsetzung)

Vorteile des %TYPE-Attributs

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

Variablen mit dem %TYPE-Attribut deklarieren

Syntax

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

Beispiele

```
...
emp_lname      employees.last_name%TYPE;
...
```

```
...
balance        NUMBER(7,2);
min_balance    balance%TYPE := 1000;
...
```

ORACLE

Copyright © 2007, Oracle. Alle Rechte vorbehalten.

Variablen mit dem %TYPE-Attribut deklarieren

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

Deklarieren Sie Variablen so, dass sie den Kontostand eines Bankkontos sowie das Mindestguthaben von 1.000 speichern. Die min_balance-Variable wird so definiert, dass sie denselben Datentyp wie die balance-Variable hat. Das %TYPE-Attribut 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 %TYPE-Attribut deklariert haben und die eine als NOT NULL definierte Datenbankspalte verwendet, den NULL-Wert zuordnen.

Boolesche Variablen deklarieren

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

ORACLE®

Copyright © 2007, Oracle. Alle Rechte vorbehalten.

Boolesche Variablen deklarieren

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 mit Hilfe 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
```

Deklarieren und initialisieren Sie eine boolesche Variable:

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

Bind-Variablen

Bind-Variablen werden:

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

ORACLE®

Copyright © 2007, Oracle. Alle Rechte vorbehalten.

Bind-Variablen

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

Bind-Variablen verwenden

Bind-Variablen werden in der Umgebung und nicht im deklarativen Bereich eines PL/SQL-Blockes erstellt. In einem PL/SQL-Block deklarierte Variablen sind nur verfügbar, wenn Sie den Block ausführen. Nach Ausführung des Blockes wird der von der Variablen belegte Speicher freigegeben. Sie können auf Bind-Variablen jedoch auch zugreifen, nachdem der Block ausgeführt wurde. Aus diesem Grund können Bind-Variablen 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.

Bind-Variablen erstellen

Um eine Bind-Variable in SQL Developer zu erstellen, verwenden Sie den VARIABLE-Befehl. Deklarieren Sie beispielsweise Variablen des Typs NUMBER und VARCHAR2 wie folgt:

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

SQL Developer kann die Bind-Variable referenzieren und ihren Wert mit dem PRINT-Befehl anzeigen.

Bind-Variablen (Fortsetzung)

Beispiel

Sie können eine Bind-Variable in einem PL/SQL-Programm referenzieren, indem Sie ihr einen Doppelpunkt voranstellen:

```
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

b_result
-----
30000
```

Hinweis: Wenn Sie eine Bind-Variable des NUMBER-Typs erstellen, können Sie die Gesamtstelligenzahl und die Anzahl der Nachkommastellen nicht angeben. Sie können jedoch die Länge von Zeichenfolgen angeben. Oracle-Variablen des NUMBER-Typs 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-Variablenwerte ausgeben

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;
```

ORACLE

Copyright © 2007, Oracle. Alle Rechte vorbehalten.

Bind-Variablenwerte ausgeben

In SQL*Plus können Sie den Wert einer Bind-Variablen mit dem PRINT-Befehl anzeigen. Wenn Sie den auf der Folie gezeigten PL/SQL-Block ausführen, wird bei Ausführung des PRINT-Befehls die folgende Ausgabe angezeigt:

b_emp_salary

7000

b_emp_salary ist eine Bind-Variable. Sie können diese Variable jetzt in einer beliebigen SQL-Anweisung oder einem beliebigen PL/SQL-Programm verwenden. Beachten Sie die SQL-Anweisung, die die Bind-Variable verwendet. Die Ausgabe der SQL-Anweisung lautet:

FIRST_NAME LAST_NAME

----- -----

Oliver	Tuvault
Sarath	Sewall
Kimberely	Grant

Hinweis: Um alle Bind-Variablen anzuzeigen, verwenden Sie den PRINT-Befehl ohne Variablen.

Bind-Variablenwerte ausgeben

Beispiel:

```
VARIABLE b_emp_salary NUMBER
SET AUTOPRINT ON
DECLARE
    v_empno NUMBER(6) :=&empno;
BEGIN
    SELECT salary INTO :b_emp_salary
    FROM employees WHERE employee_id = v_empno;
END;
```

Ausgabe:

```
7000
```

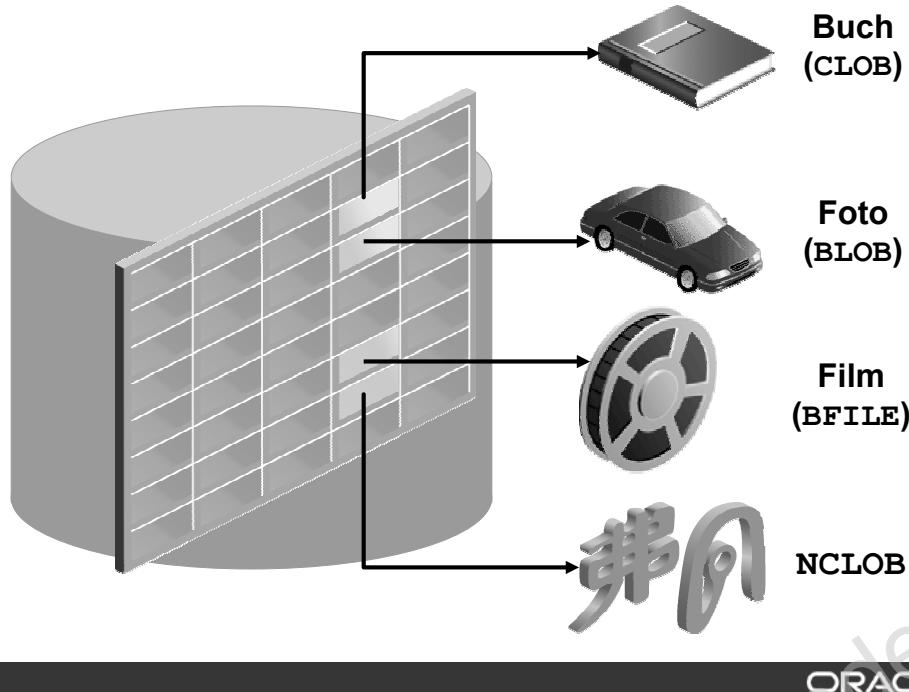
ORACLE

Copyright © 2007, Oracle. Alle Rechte vorbehalten.

Bind-Variablen ausgeben (Fortsetzung)

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

Variablen mit LOB-Datentypen



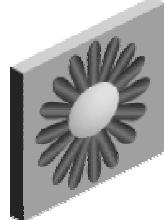
Copyright © 2007, Oracle. Alle Rechte vorbehalten.

Variablen mit LOB-Datentypen

Large Objects (LOBs) dienen zum Speichern großer Datenmengen. Datenbankspalten können der LOB-Kategorie angehören. Mit der LOB-Kategorie der Datentypen (BLOB, CLOB usw.) können Sie je nach Größe des Datenbankblockes Blöcke mit unstrukturierten Daten (z. B. Text, Grafiken, Videoclips und Sound-Dateien) mit bis zu 128 Terabyte speichern. LOB-Datentypen ermöglichen den effizienten, direkten und schrittweisen Zugriff auf die Daten und können Attribute eines Objekttyps sein.

- Mit dem CLOB-Datentyp (Character Large Object) können Sie große Blöcke von Zeichendaten in der Datenbank speichern.
- Mit dem BLOB-Datentyp (Binary Large Object) können Sie große unstrukturierte und strukturierte binäre Objekte in der Datenbank speichern. Wenn Sie die Daten in die Datenbank einfügen oder daraus abrufen, interpretiert die Datenbank die Daten nicht. Externe Anwendungen, die die Daten verwenden, müssen sie interpretieren.
- Mit dem BFILE-Datentyp (Binärdatei) können Sie große Binärdateien speichern. Im Gegensatz zu anderen LOBs werden BFILES nicht in der Datenbank gespeichert. Sie werden außerhalb der Datenbank gespeichert. BFILES können Betriebssystemdateien sein. In der Datenbank wird nur ein Zeiger zum BFILE gespeichert.
- Mit dem NCLOB-Datentyp (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

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

Struktur einer PL/SQL-Tabelle Struktur einer PL/SQL-Tabelle

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

PLS_INTEGER VARCHAR2
PLS_INTEGER NUMBER

ORACLE

Copyright © 2007, Oracle. Alle Rechte vorbehalten.

Zusammengesetzte Datentypen

Skalare Datentypen haben keine internen Komponenten. Zusammengesetzte Datentypen haben interne Komponenten, die individuell bearbeitet werden können. Zusammengesetzte Datentypen (auch als *Collections* bezeichnet) sind: TABLE, RECORD, NESTED TABLE und VARRAY.

Mit dem TABLE-Datentyp können Sie Datensammlungen als vollständiges Objekt referenzieren und bearbeiten. Mit dem RECORD-Datentyp können Sie verwandte, aber unterschiedliche Daten als logische Einheit behandeln.

Die Datentypen NESTED TABLE und VARRAY werden im Kurs *Oracle Database 11g: PL/SQL-Datenbankprogrammierung* behandelt.

Zusammenfassung

In dieser Lektion haben Sie Folgendes gelernt:

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

ORACLE

Copyright © 2007, Oracle. Alle Rechte vorbehalten.

Zusammenfassung

Anonyme PL/SQL-Blöcke sind grundlegende, unbenannte Einheiten von PL/SQL-Programmen. Sie bestehen aus einer Gruppe von SQL- oder PL/SQL-Anweisungen zum Ausführen einer logischen Funktion. Der deklarative Teil ist der erste Teil eines PL/SQL-Blockes. Er wird zum Deklarieren von Objekten wie Variablen, Konstanten, Cursorn und Definitionen von Fehlersituationen verwendet, die als *Exceptions* bezeichnet werden.

In dieser Lektion wurde erläutert, wie Sie Variablen im deklarativen Bereich deklarieren. Sie haben einige der Richtlinien zum Deklarieren von Variablen kennen gelernt. Sie haben gelernt, 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 2 – Überblick

Diese Übungen behandeln folgende Themen:

- Gültige Identifier bestimmen
- Gültige Variablen Deklarationen bestimmen
- Variablen in anonymen Blöcken deklarieren
- Variablen mit dem %TYPE-Attribut deklarieren
- Bind-Variablen deklarieren und ausgeben
- Einen PL/SQL-Block ausführen

ORACLE®

Copyright © 2007, Oracle. Alle Rechte vorbehalten.

Übungen zu Lektion 2 – Überblick

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

Übungen zu Lektion 2

1. Identifizieren Sie gültige und ungültige Identifier-Namen:
 - a. today
 - b. last_name
 - c. today's_date
 - d. Number_of_days_in_February_this_year
 - e. Isleap\$year
 - f. #number
 - g. NUMBER#
 - h. number1to7
2. Identifizieren Sie gültige und ungültige Variablen-deklarationen und -initialisierungen:
 - a. number_of_copies PLS_INTEGER;
 - b. printer_name constant VARCHAR2(10);
 - c. deliver_to VARCHAR2(10) :=Johnson;
 - d. by_when DATE:= CURRENT_DATE+1;

3. Prüfen Sie den folgenden anonymen Block, und wählen Sie die richtige Aussage.

```
DECLARE
    v_fname VARCHAR2(20);
    v_lname VARCHAR2(15) DEFAULT 'fernandez';
BEGIN
    DBMS_OUTPUT.PUT_LINE(v_fname || ' ' || v_lname);
END;
/
```

- a. Der Block wird erfolgreich ausgeführt. Er gibt "fernandez" aus.
 - b. Der Block gibt einen Fehler zurück, da die fname-Variablen ohne Initialisierung verwendet wird.
 - c. Der Block wird erfolgreich ausgeführt. Er gibt "null fernandez" aus.
 - d. Der Block gibt einen Fehler zurück, da Sie Variablen des Typs VARCHAR2 nicht mit dem DEFAULT-Schlüsselwort initialisieren können.
 - e. Der Block gibt einen Fehler zurück, da die v_fname-Variablen nicht deklariert ist.
4. Erstellen Sie einen anonymen Block. Laden Sie in SQL Developer das Skript lab_01_02_soln.sql, das Sie in der zweiten Aufgabe der Übungen zu Lektion 1 erstellt haben.
 - a. Fügen Sie diesem PL/SQL-Block einen deklarativen Bereich hinzu. Deklarieren Sie im deklarativen Bereich die folgenden Variablen:
 1. v_today-Variablen des DATE-Typs. Initialisieren Sie today mit SYSDATE.
 2. v_tomorrow-Variablen des today-Typs. Deklarieren Sie diese Variable mit dem %TYPE-Attribut.
 - b. Initialisieren Sie im ausführbaren Bereich die tomorrow-Variablen mit einem Ausdruck, der das Datum von morgen berechnet (erhöhen Sie den Wert in today um eins). Zeigen Sie die Werte von today und tomorrow im Anschluss an "Hello World" an.

Übungen zu Lektion 2 (Fortsetzung)

- c. Führen Sie das Skript aus, und speichern Sie es unter dem Namen lab_02_04_soln.sql. Hier die Ausgabe:

```
anonymous block completed
```

```
Hello World
```

```
TODAY IS : 16-MAY-07
```

```
TOMORROW IS : 17-MAY-07
```

5. Bearbeiten Sie das Skript lab_02_04_soln.sql.

- a. Fügen Sie Code hinzu, um zwei Bind-Variablen zu erstellen.

Erstellen Sie die Bind-Variablen b_basic_percent und b_pf_percent des NUMBER-Typs.

- b. Weisen Sie den Bind-Variablen b_basic_percent und b_pf_percent im ausführbaren Bereich des PL/SQL-Blockes die Werte 45 bzw. 12 zu.

- c. Beenden Sie den PL/SQL-Block mit einem Schrägstrich (/), und zeigen Sie den Wert der Bind-Variablen mit dem PRINT-Befehl an.

- d. Führen Sie die Skriptdatei aus, und speichern Sie sie unter dem Namen lab_02_05_soln.sql. Ein Beispiel der Ausgabe lautet wie folgt:

```
anonymous block completed
```

```
Hello World
```

```
TODAY IS : 16-MAY-07
```

```
TOMORROW IS : 17-MAY-07
```

```
basic_percent
```

```
--
```

```
45
```

```
pf_percent
```

```
--
```

```
12
```


Ausführbare Anweisungen erstellen

3

ORACLE®

Copyright © 2007, Oracle. Alle Rechte vorbehalten.

Oracle Internal & Oracle Academy
Use Only

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

ORACLE®

Copyright © 2007, Oracle. Alle Rechte vorbehalten.

Lernziel

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.

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
- können wie folgt klassifiziert werden:
 - Identifier: `v_fname, c_percent`
 - Begrenzungszeichen: `; , +, -`
 - Literale: `John, 428, True`
 - Kommentare: `--, /* */`

ORACLE

Copyright © 2007, Oracle. Alle Rechte vorbehalten.

Lexikalische Einheiten in PL/SQL-Blöcken

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

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

Identifier in Anführungszeichen:

- erfordern die Berücksichtigung der Groß-/Kleinschreibung
- 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. Die Verwendung von Identifiern in Anführungszeichen wird jedoch nicht empfohlen.

- **Begrenzungszeichen:** Begrenzungszeichen sind Symbole mit spezieller Bedeutung. Sie haben bereits gelernt, dass das Semikolon (`;`) zum Beenden von SQL- oder PL/SQL-Anweisungen verwendet wird. Aus diesem Grund ist das Semikolon ein Beispiel für ein Begrenzungszeichen. Weitere Informationen finden Sie im *PL/SQL User's Guide and Reference*.

Lexikalische Einheiten in PL/SQL-Blöcken (Fortsetzung)

- **Begrenzungszeichen (Fortsetzung)**

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

Einfache Symbole

Symbol	Bedeutung
+	Additionsoperator
-	Subtraktions-/Verneinungsoperator
*	Multiplikationsoperator
/	Divisionsoperator
=	Gleich-Operator
@	Indikator für Remote-Zugriff
;	Abschlusszeichen für eine Anweisung

Zusammengesetzte Symbole

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

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

- **Literale:** Jeder Wert, der einer Variablen zugewiesen wird, ist ein Literal. Alle Zeichen, Zahlen, booleschen Werte oder Datumswerte, die keine Identifier sind, sind Literale. Literale werden wie folgt klassifiziert:
 - **Zeichenliterale:** Zeichenfolgenliterale haben generell den Datentyp CHAR oder VARCHAR2 und werden daher als Zeichenliterale bezeichnet (z. B. John und 12C).
 - **Numerische Literale:** Numerische Literale stehen für eine Ganzzahl oder einen realen Wert (z. B. 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 in einen PL/SQL-Block aufnehmen, kann der Compiler die Anleitungen nicht interpretieren. Sie müssen angeben können, dass die Anleitungen nicht kompiliert werden müssen. 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.

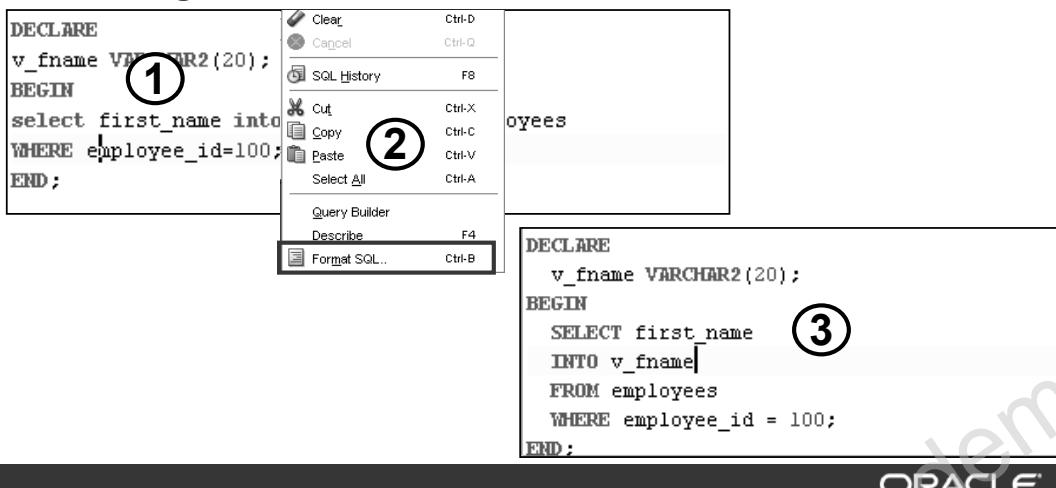
PL/SQL-Blocksyntax – Richtlinien

- **Literale**

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

```
name := 'Henderson';
```

- Anweisungen können sich über mehrere Zeilen erstrecken.



Copyright © 2007, Oracle. Alle Rechte vorbehalten.

PL/SQL-Blocksyntax – Richtlinien

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

- Zeichenliterale beinhalten alle druckbaren Zeichen des PL/SQL-Zeichensatzes: Buchstaben, Zahlen, Leer- und Sonderzeichen.
- Numerische Literale können entweder durch einen einfachen Wert (z. B. -32.5) oder durch wissenschaftliche Notation (z. B. 2E5, d. h. 2 mal $10^5 = 200.000$) dargestellt werden.

Anweisungen können mehrere Zeilen umfassen (siehe 3. Beispiel auf der Folie).

Unformatierte SQL-Anweisungen (siehe 1. Beispiel auf der Folie) können Sie im Kontextmenü von SQL Developer mit der Option **Format SQL** formatieren. Klicken Sie mit der rechten Maustaste auf das aktive SQL-Worksheet, und wählen Sie im daraufhin angezeigten Kontextmenü **Format SQL** (siehe 2. Beispiel).

Code kommentieren

- Stellen Sie einzeiligen Kommentaren zwei Bindestriche (--) voran.
- Schließen Sie mehrzeilige Kommentare in die Symbole /* und */ ein.

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 © 2007, Oracle. Alle Rechte vorbehalten.

Code kommentieren

Sie sollten Code kommentieren, um die einzelnen Phasen zu dokumentieren und das Debugging zu erleichtern. Kommentieren Sie den PL/SQL-Code mit zwei Bindestrichen (--), wenn der Kommentar eine Zeile lang ist. Erstreckt sich der Kommentar über mehrere Zeilen, setzen Sie ihn zwischen die Symbole /* und */.

Kommentare dienen rein zu Informationszwecken und erzwingen keinerlei Bedingungen oder Verhalten hinsichtlich der Logik oder Daten. Richtig platzierte Kommentare verbessern die Lesbarkeit von Code und erleichtern die spätere Code-Verwaltung. Im Beispiel auf der Folie gibt die in die Symbole /* und */ eingebettete Zeile einen Kommentar an, der den nachfolgenden Code erläutert.

SQL-Funktionen in PL/SQL

- **In prozeduralen Anweisungen verfügbar:**
 - Single-Row-Funktionen
- **Nicht in prozeduralen Anweisungen verfügbar:**
 - DECODE
 - Gruppenfunktionen

ORACLE®

Copyright © 2007, Oracle. Alle Rechte vorbehalten.

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 (z. B. Single Row-Zahlen- und Zeichenfunktionen, Datentyp-Konvertierungsfunktionen 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(prod_description);
```

- Anzahl der Beschäftigungsmonate eines Mitarbeiters abrufen:

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

ORACLE®

Copyright © 2007, Oracle. Alle Rechte vorbehalten.

SQL-Funktionen in PL/SQL – Beispiele

Sie können mit Hilfe von SQL-Funktionen Daten bearbeiten. Die Funktionen werden in folgende Kategorien gruppiert:

- Numerische Funktionen
- Zeichenfunktionen
- Konvertierungsfunktionen
- Datumsfunktionen
- Verschiedene Funktionen

Sequences in PL/SQL-Ausdrücken verwenden

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;
/
```



Copyright © 2007, Oracle. Alle Rechte vorbehalten.



Sequence-Werte aufrufen

In Oracle Database 11g können Sie die Pseudospalten NEXTVAL und CURRVAL in jedem beliebigen PL/SQL-Kontext verwenden, in dem Ausdrücke des NUMBER-Datentyps 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 CURRVAL mit Hilfe einer SELECT-Anweisung referenziert, um eine Sequence-Nummer zu ermitteln. Diese Methode führte zu Problemen bei der Verwendbarkeit von Sequences.

In Oracle Database 11g ist es nicht mehr erforderlich, eine SQL-Anweisung zu erstellen, um einen Sequence-Wert abzurufen. Das erweiterte Sequence Feature bietet folgende Vorteile:

- Verbesserte Sequence-Verwendbarkeit
- Geringerer Schreibaufwand für 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 © 2007, Oracle. Alle Rechte vorbehalten.

Datentypen konvertieren

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 folgendes 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;
END;
/
```

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

Implizite Konvertierungen sind möglich zwischen:

- Zeichen und Zahlen
- Zeichen und Datumsangaben

Datentypen konvertieren (Fortsetzung)

Explizite Konvertierungen: Sie können Werte mit Hilfe integrierter 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 bzw. TO_NUMBER.

Oracle Internal & Oracle Academy
Use Only

Datentypen konvertieren

1

```
date_of_joining DATE := '02-Feb-2000' ;
```

2

```
date_of_joining DATE := 'February 02,2000' ;
```

3

```
date_of_joining DATE := TO_DATE('February  
02,2000','Month DD, YYYY');
```

ORACLE

Copyright © 2007, Oracle. Alle Rechte vorbehalten.

Datentypen konvertieren (Fortsetzung)

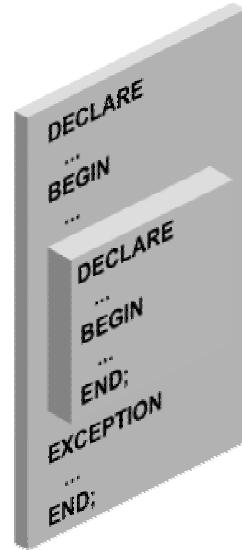
Betrachten Sie die drei Beispiele impliziter und expliziter Konvertierungen des DATE-Datentyps auf der Folie:

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

Verschachtelte Blöcke

PL/SQL-Blöcke können verschachtelt werden.

- **Ausführbare Bereiche (`BEGIN ... END`) können verschachtelte Blöcke enthalten.**
- **Der `EXCEPTION`-Abschnitt kann verschachtelte Blöcke enthalten.**



ORACLE®

Copyright © 2007, Oracle. Alle Rechte vorbehalten.

Verschachtelte Blöcke

Dank der prozeduralen Fähigkeiten von PL/SQL können Anweisungen verschachtelt werden. Sie können Blöcke überall verschachteln, wo ausführbare Anweisungen zulässig sind. Der verschachtelte Block wird 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`-Abschnitt 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;
```

ORACLE

Copyright © 2007, Oracle. Alle Rechte vorbehalten.

Verschachtelte Blöcke (Fortsetzung)

Im Beispiel auf der Folie wird ein äußerer (übergeordneter) Block und ein verschachtelter (untergeordneter) Block gezeigt. Die Variable `v_outer_variable` wird im äußeren Block deklariert, die Variable `v_inner_variable` im inneren Block.

`v_outer_variable` ist für den äußeren Block eine lokale Variable, für den inneren Block jedoch eine globale Variable. 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 gilt `v_outer_variable` als die globale Variable für alle übergeordneten Blöcke. Sie können auf diese Variable im inneren Block wie auf der Folie gezeigt zugreifen. In einem PL/SQL-Block deklarierte Variablen werden für diesen Block als lokal und für alle seine Unterblöcke als global betrachtet.

`v_inner_variable` ist für den inneren Block eine lokale Variable 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.

Gültigkeitsbereich und Sichtbarkeit von Variablen

```
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
        1-- 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);
    2-- END;
    DBMS_OUTPUT.PUT_LINE('Date of Birth: '||v_date_of_birth);
END;
/
```

ORACLE

Copyright © 2007, Oracle. Alle Rechte vorbehalten.

Gültigkeitsbereich und Sichtbarkeit von Variablen

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
```

Betrachten Sie das Geburtsdatum von Vater und Kind.

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 Bezeichner auf die Variable zugegriffen werden kann.

Gültigkeitsbereich

- Die Variablen `v_father_name` und `v_date_of_birth` werden im äußeren Block deklariert. Diese Variablen haben den Gültigkeitsbereich des Blockes, in dem sie deklariert und zugänglich sind. Daher ist der Gültigkeitsbereich dieser Variablen auf den äußeren Block begrenzt.

Gültigkeitsbereich und Sichtbarkeit von Variablen (Fortsetzung)

Gültigkeitsbereich (Fortsetzung)

- Die Variablen `v_child_name` und `v_date_of_birth` werden im inneren bzw. verschachtelten Block deklariert. Sie können auf diese Variablen nur innerhalb des verschachtelten Blockes, nicht aber im äußeren Block 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 in diesem Fall sichtbar und kann somit angezeigt werden.

Ein Block kann keine gleichnamigen Variablen enthalten. Sie können gleichnamige Variablen jedoch in zwei verschiedenen (verschachtelten) Blöcken deklarieren. Die beiden durch die Identifier dargestellten Elemente sind voneinander unabhängig. Änderungen an einem Element wirken sich nicht auf das andere Element aus.

Identifier kennzeichnen

```
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 © 2007, Oracle. Alle Rechte vorbehalten.

Identifier kennzeichnen

Ein Bezeichner ist ein Label, das einem Block zugewiesen wird. Mit Hilfe von Bezeichnern können Sie auf die Variablen zugreifen, die innerhalb des Gültigkeitsbereichs liegen, aber nicht sichtbar sind. Betrachten Sie den Code: Sie können jetzt das Geburtsdatum von Vater und Kind im inneren Block ausgeben. Der äußere Block ist mit dem Label `outer` gekennzeichnet. Mit diesem Label können Sie auf die im äußeren Block deklarierte Variable `v_date_of_birth` zugreifen.

Labels sind nicht auf den äußeren Block begrenzt. Sie können jeden beliebigen Block mit einem Label kennzeichnen. Der Code auf der Folie ergibt folgende Ausgabe:

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

Quiz – 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 © 2007, Oracle. Alle Rechte vorbehalten.

Quiz – Gültigkeitsbereich von Variablen bestimmen

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

1. Wert von MESSAGE an 1. Position
2. Wert von TOTAL_COMP an 2. Position
3. Wert von COMM an 1. Position
4. Wert von outer.COMM an 1. Position
5. Wert von COMM an 2. Position
6. Wert von MESSAGE an 2. Position

Antworten – Gültigkeitsbereich von Variablen bestimmen

1. Wert von MESSAGE an 1. Position: eligible for commission
2. Wert von TOTAL_COMP an 2. Position: Fehler, TOTAL_COMP hier nicht sichtbar, da im inneren Block definiert.
3. Wert von COMM an 1. Position: 0
4. Wert von outer.COMM an 1. Position: 15000
5. Wert von COMM an 2. Position: 12000
6. Wert von MESSAGE an 2. Position: eligible for commission

Oracle Internal & Oracle Academy
Use Only

Operatoren in PL/SQL

- Logische Operatoren
- Arithmetische Operatoren
- Verkettung
- Klammern zur Steuerung der Reihenfolge
- Exponential-Operator (**)

Wie in SQL

Copyright © 2007, Oracle. Alle Rechte vorbehalten.

ORACLE

Operatoren in PL/SQL

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

Operator	Operation
**	Exponentialoperator
+, -	Vorzeichen, Verneinung
*, /	Multiplikation, Division
+, -,	Addition, Subtraktion, Verkettung
=, <, >, <=, >=, <>, !=, ~=, ^=, IS NULL, LIKE, BETWEEN, IN	Vergleich
NOT	Logische Verneinung
AND	Logisches UND
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);
```

ORACLE

Copyright © 2007, Oracle. Alle Rechte vorbehalten.

Operatoren in PL/SQL (Fortsetzung)

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

- Vergleiche mit NULL-Werten ergeben stets NULL.
- Das Anwenden des logischen Operators NOT auf einen NULL-Wert ergibt stets NULL.
- Wenn die Bedingung bei bedingten Kontrollstrukturen NULL ergibt, wird die zugehörige Anweisungsfolge nicht ausgeführt.

Richtlinien für die Programmierung

Vereinfachen Sie die Code-Verwaltung, indem Sie:

- den Code mit Kommentaren dokumentieren**
- eine Konvention für Groß-/Kleinschreibung im Code festlegen**
- Benennungskonventionen für Identifier und andere Objekte festlegen**
- die Lesbarkeit durch Einrückungen verbessern**

ORACLE®

Copyright © 2007, Oracle. Alle Rechte vorbehalten.

Richtlinien für die Programmierung

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 durch Verwendung 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
Identifier und Parameter	Kleinbuchstaben	v_sal, emp_cursor, g_sal, p_empno
Datenbanktabellen und -spalten	Kleinbuchstaben	employees, employee_id, department_id

Code einrücken

Rücken Sie die einzelnen Code-Ebenen ein, um den Code übersichtlich zu strukturieren.

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

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

ORACLE

Copyright © 2007, Oracle. Alle Rechte vorbehalten.

Code einrücken

Rücken Sie die einzelnen Code-Ebenen ein, um den Code übersichtlich zu strukturieren und die Lesbarkeit zu verbessern. Um die Struktur zu zeigen, können Sie Zeilen durch Zeilenschaltungen trennen und mit Leerzeichen und Tabulatoren einrücken. Vergleichen Sie die folgenden IF-Anweisungen hinsichtlich der Lesbarkeit:

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

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

Zusammenfassung

In dieser Lektion haben Sie Folgendes gelernt:

- **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**

ORACLE®

Copyright © 2007, Oracle. Alle Rechte vorbehalten.

Zusammenfassung

Da PL/SQL eine Erweiterung von SQL ist, sind die allgemeinen Syntaxregeln von SQL auch auf PL/SQL anwendbar.

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-Abschnitt ebenfalls zum ausführbaren Bereich zählt, kann auch er verschachtelte Blöcke enthalten. Achten Sie bei der Verwendung verschachtelter Blöcke auf den richtigen Gültigkeitsbereich und die richtige Sichtbarkeit. Vermeiden Sie gleichnamige Identifier 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 3 – Überblick

Diese Übungen behandeln folgende Themen:

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

ORACLE®

Copyright © 2007, Oracle. Alle Rechte vorbehalten.

Übungen zu Lektion 3 – Überblick

Die erste und zweite Übung werden nicht am Rechner durchgeführt.

Übungen zu Lektion 3

PL/SQL-Block

```
DECLARE
    v_weight      NUMBER(3) := 600;
    v_message     VARCHAR2(255) := 'Product 10012';
BEGIN
    DECLARE
        v_weight      NUMBER(3) := 1;
        v_message     VARCHAR2(255) := 'Product 11001';
        v_new_locn   VARCHAR2(50) := 'Europe';
    BEGIN
        v_weight := v_weight + 1;
        v_new_locn := 'Western ' || v_new_locn;
    1    END;
    v_weight := v_weight + 1;
    v_message := v_message || ' is in stock';
    v_new_locn := 'Western ' || v_new_locn;
2    END;
/

```

1. Beurteilen Sie den vorangegangenen PL/SQL-Block, und bestimmen Sie den Datentyp und Wert jeder der folgenden Variablen entsprechend den Regeln für Gültigkeitsbereiche.
 - a. Wert von v_weight an 1. Position:
 - b. Wert von v_new_locn an 1. Position:
 - c. Wert von v_weight an 2. Position:
 - d. Wert von v_message an 2. Position:
 - e. Wert von v_new_locn an 2. Position:

Übungen zu Lektion 3 (Fortsetzung)

Gültigkeitsbereich – Beispiel

```
DECLARE
    v_customer      VARCHAR2(50) := 'Womansport';
    v_credit_rating VARCHAR2(50) := 'EXCELLENT';
BEGIN
    DECLARE
        v_customer      NUMBER(7) := 201;
        v_name          VARCHAR2(25) := 'Unisports';
    BEGIN
        v_credit_rating := 'GOOD';
        ...
    END;
    ...
END;
/
```

2. Bestimmen Sie im vorangegangenen PL/SQL-Block den Wert und Datentyp für jeden der folgenden Fälle.

- a. Wert von `v_customer` im verschachtelten Block:
- b. Wert von `name` im verschachtelten Block:
- c. Wert von `v_credit_rating` im verschachtelten Block:
- d. Wert von `v_customer` im Hauptblock:
- e. Wert von `name` im Hauptblock:
- f. Wert von `v_credit_rating` im Hauptblock:

Übungen zu Lektion 3 (Fortsetzung)

3. Bearbeiten Sie lab_02_05_soln.sql.
 - a. Kommentieren Sie die Zeilen zum Erstellen der Bind-Variablen mit einzeiligen Kommentaren.
 - b. Kommentieren Sie im ausführbaren Bereich die Zeilen zum Zuweisen von Werten zu Bind-Variablen mit mehrzeiligen Kommentaren.
 - c. Deklarieren Sie die Variablen v_basic_percent und v_pf_percent, und initialisieren Sie sie mit 45 bzw. 12. Deklarieren Sie außerdem zwei Variablen: v_fname des VARCHAR2-Datentyps mit der Größe 15 und v_emp_sal des NUMBER-Typs mit der Größe 10.
 - d. Fügen Sie die folgende SQL-Anweisung in den ausführbaren Bereich ein:

```
SELECT first_name, salary
INTO v_fname, v_emp_sal FROM employees
WHERE employee_id=110;
```
 - e. Ändern Sie die Zeile für die Anzeige von "Hello World" so, dass "Hello" und der Vorname angezeigt wird. Sie können wahlweise die Zeilen für die Datumsanzeige kommentieren und die Bind-Variablen anzeigen.
 - f. Berechnen Sie den Beitrag der Mitarbeiter zur Unterstützungskasse.
Der Anteil für die Unterstützungskasse liegt bei 12 % des Grundgehalts, und das Grundgehalt beträgt 45 % des Gehalts. Verwenden Sie zum Berechnen die Bind-Variablen. Verwenden Sie versuchsweise nur einen Ausdruck, um den Betrag für die Unterstützungskasse zu berechnen. Zeigen Sie das Gehalt des Mitarbeiters und seinen Beitrag zur Unterstützungskasse an.
 - g. Führen Sie das Skript aus, und speichern Sie es unter dem Namen lab_03_03_soln.sql. Die Ausgabe kann beispielsweise wie folgt lauten:

```
anonymous block completed
Hello John
YOUR SALARY IS : 8200
YOUR CONTRIBUTION TOWARDS PF:
442.8
```

Mit dem Oracle-Datenbank-Server interagieren

4

ORACLE®

Copyright © 2007, Oracle. Alle Rechte vorbehalten.

Oracle Internal & Oracle Academy
Use Only

Ziele

Nach Ablauf dieser Lektion haben Sie folgende Ziele erreicht:

- Die **SQL-Anweisungen bestimmen, die direkt in den ausführbaren Bereich eines PL/SQL-Blockes aufgenommen werden können**
- Daten in PL/SQL mit **DML-Anweisungen bearbeiten**
- Steueranweisungen für Transaktionen in PL/SQL verwenden
- Mit der **INTO-Klausel von SQL-Anweisungen zurückgegebene Werte speichern**
- Zwischen impliziten und expliziten Cursorn unterscheiden
- **SQL-Cursor-Attribute verwenden**

ORACLE®

Copyright © 2007, Oracle. Alle Rechte vorbehalten.

Lernziel

In dieser Lektion wird beschrieben, wie Sie die SQL-Standardanweisungen SELECT, INSERT, UPDATE, DELETE und MERGE in PL/SQL-Blöcke einbetten. Sie lernen, wie Sie DDL-(Data Definition Language-)Anweisungen und Steueranweisungen für Transaktionen in PL/SQL aufnehmen. Sie erfahren über die Notwendigkeit von Cursorn und wie Sie die beiden Cursor-Typen unterscheiden. Darüber hinaus werden in dieser Lektion die verschiedenen SQL-Cursor-Attribute erläutert, die Sie mit impliziten Cursorn verwenden können.

SQL-Anweisungen in PL/SQL

- Datenbankzeilen rufen Sie mit dem **SELECT-Befehl** ab.
- Änderungen in Datenbankzeilen nehmen Sie mit **DML-Befehlen** vor.
- Transaktionen steuern Sie mit den Befehlen **COMMIT**, **ROLLBACK** und **SAVEPOINT**.

ORACLE

Copyright © 2007, Oracle. Alle Rechte vorbehalten.

SQL-Anweisungen in PL/SQL

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

- Das END-Schlüsselwort gibt das Ende eines PL/SQL-Blockes und nicht das Ende einer Transaktion an. Ebenso wie ein Block mehrere Transaktionen umfassen kann, kann eine Transaktion mehrere Blöcke umfassen.
- PL/SQL bietet keine direkte Unterstützung für DDL-Anweisungen wie CREATE TABLE, ALTER TABLE oder DROP TABLE. PL/SQL unterstützt Early Binding, das nicht möglich ist, wenn Anwendungen während der Laufzeit Werte übergeben müssen, um Datenbankobjekte zu erstellen. Sie können diese DDL-Anweisungen nicht direkt ausführen, da es sich um dynamische SQL-Anweisungen handelt. Dynamische SQL-Anweisungen werden zur Laufzeit als Zeichenfolgen erstellt und können Platzhalter für Parameter enthalten. Daher können Sie DDL-Anweisungen in PL/SQL mit dynamischem SQL ausführen. Die Verwendung von dynamischem SQL wird im Kurs *Oracle Database 11g: PL/SQL-Datenbankprogrammierung* ausführlich behandelt.
- PL/SQL unterstützt DCL-(Data Control Language-)Anweisungen wie GRANT oder REVOKE nicht direkt. 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 © 2007, Oracle. Alle Rechte vorbehalten.

SELECT-Anweisungen in PL/SQL

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

<i>select_list</i>	Liste mit mindestens einer Spalte. Kann SQL-Ausdrücke, Zeilenfunktionen oder Gruppenfunktionen aufnehmen
<i>variable_name</i>	Skalare Variable, die den abgerufenen Wert aufnimmt
<i>record_name</i>	PL/SQL-Datensatz, der die abgerufenen Werte aufnimmt
<i>table</i>	Gibt den Namen der Datenbanktabelle an
<i>condition</i>	Besteht aus Spaltennamen, Ausdrücken, Konstanten und Vergleichsoperatoren einschließlich PL/SQL-Variablen und Konstanten

Richtlinien für den Datenabruf in PL/SQL

- Beenden Sie jede SQL-Anweisung mit einem Semikolon (;).
- Speichern Sie jeden abgerufenen Wert mit Hilfe 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-Anweisungen in PL/SQL (Fortsetzung)

- 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 erforderlich.
- Abfragen dürfen nur eine Zeile zurückgeben.

Beispiel:

```
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 ;
/
```

ORACLE

Copyright © 2007, Oracle. Alle Rechte vorbehalten.

SELECT-Anweisungen in PL/SQL (Fortsetzung)

INTO-Klausel

Die **INTO-Klausel** ist obligatorisch und steht zwischen der **SELECT-** und der **FROM-Klausel**. Sie gibt die Namen der Variablen an, welche 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- oder Host-Variablen mit Hilfe der **INTO-Klausel** auf.

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 behebt diese Fehler durch Auslösen von Standard-Exceptions, die Sie im Exception-Abschnitt des Blockes mit den Exceptions **NO_DATA_FOUND** und **TOO_MANY_ROWS** verarbeiten können. Nehmen Sie eine **WHERE-Bedingung** in die **SQL-Anweisung** auf, so dass die Anweisung nur eine Zeile zurückgibt. Auf die Exception-Behandlung wird im Verlauf dieses Kurses noch eingegangen.

SELECT-Anweisungen in PL/SQL (Fortsetzung)

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.

Oracle Internal & Oracle Academy
Use Only

Daten in PL/SQL abrufen

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

Beispiel:

```
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;
END ;
/
```

ORACLE

Copyright © 2007, Oracle. Alle Rechte vorbehalten.

Daten in PL/SQL abrufen

Im Beispiel auf der Folie werden die Variablen `emp_hiredate` und `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 `employees`-Tabelle die Werte der Spalten `hire_date` und `salary` abgerufen. Anschließend werden die Werte in der Variablen `emp_hiredate` bzw. `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. Beispiel: Wenn Sie im Beispiel auf der Folie `emp_hiredate` und `emp_salary` austauschen, generiert die Anweisung einen Fehler.

Daten in PL/SQL abrufen

Die Summe der Gehälter aller Mitarbeiter der angegebenen Abteilung abrufen

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;
```

ORACLE

Copyright © 2007, Oracle. Alle Rechte vorbehalten.

Daten in PL/SQL abrufen (Fortsetzung)

Im Beispiel auf der Folie werden die Variablen `sum_sal` und `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 mit Hilfe der SQL-Aggregatfunktion `SUM` berechnet. Das berechnete Gesamtgehalt wird der `sum_sal`-Variablen zugewiesen.

Hinweis: Sie können in der PL/SQL-Syntax keine Gruppenfunktionen verwenden. Diese werden in SQL-Anweisungen wie im Beispiel gezeigt in PL/SQL-Blöcke aufgenommen. Folgende Syntax ist nicht möglich:

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

Der PL/SQL-Block auf der Folie ergibt:

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

Benennungskonventionen

```
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 © 2007, Oracle. Alle Rechte vorbehalten.

Benennungskonventionen

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

Im Beispiel auf der Folie werden das Einstellungsdatum und das heutige Datum für `employee_id` 176 aus der `employees`-Tabelle abgerufen. 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 `employees`-Tabelle identisch sind.

Die folgende `DELETE`-Anweisung löscht neben dem Mitarbeiter namens "King" auch alle Mitarbeiter in der `employees`-Tabelle, 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

- **Verwenden Sie eine Benennungskonvention, um Mehrdeutigkeiten in der WHERE-Klausel zu vermeiden.**
- **Verwenden Sie keine Namen von Datenbankspalten als Identifier.**
- **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 Datenbanktabellen.**
- **Die Namen von Spalten in Datenbanktabellen haben Vorrang gegenüber den Namen lokaler Variablen.**

ORACLE®

Copyright © 2007, Oracle. Alle Rechte vorbehalten.

Benennungskonventionen (Fortsetzung)

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

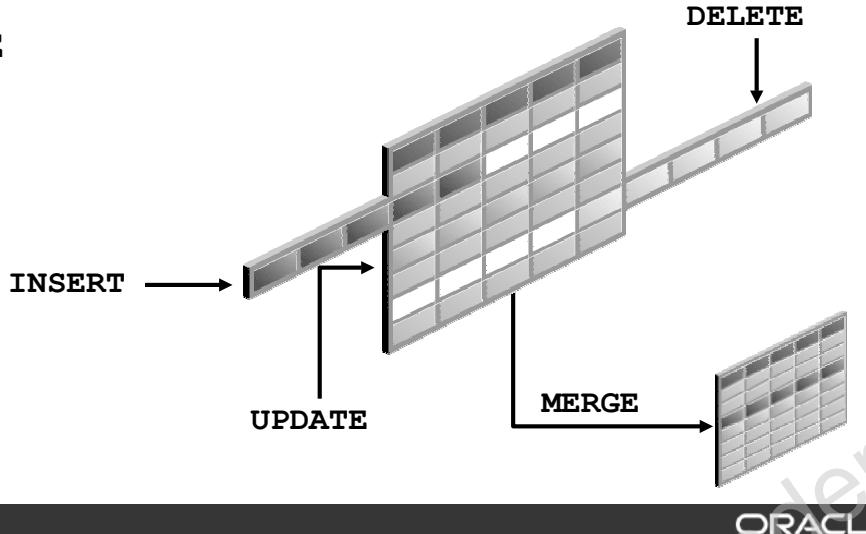
- Datenbankspalten und Identifier müssen unterschiedliche Namen haben.
- Syntaxfehler können auftreten, da PL/SQL zuerst nach einer Spalte in der Datenbanktabelle sucht.

Hinweis: In der SELECT-Klausel können keine Mehrdeutigkeiten auftreten, da jeder darin enthaltene Identifier der Name einer Datenbankspalte sein muss. In der INTO-Klausel können keine Mehrdeutigkeiten auftreten, da jeder darin enthaltene Identifier eine PL/SQL-Variable sein muss. Verwechslungen können nur in der WHERE-Klausel auftreten.

Daten mit PL/SQL bearbeiten

Änderungen in Datenbanktabellen mit DML-Befehlen durchführen:

- **INSERT**
- **UPDATE**
- **DELETE**
- **MERGE**



Copyright © 2007, Oracle. Alle Rechte vorbehalten.

Daten mit PL/SQL bearbeiten

Mit DML-Befehlen können Sie Daten in der Datenbank bearbeiten. Sie können die DML-Befehle **INSERT**, **UPDATE**, **DELETE** und **MERGE** in PL/SQL uneingeschränkt verwenden. Zeilensperren (und Tabellensperren) heben Sie auf, indem Sie die Anweisung **COMMIT** oder **ROLLBACK** in den PL/SQL-Code aufnehmen.

- **INSERT**-Anweisungen fügen neue Zeilen in die Tabelle ein.
- **UPDATE**-Anweisungen ändern vorhandene Tabellenzeilen.
- **DELETE**-Anweisungen entfernen Tabellenzeilen.
- **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 Objektprivilegien **INSERT** und **UPDATE** für die Zieltabelle und das **SELECT**-Privileg für die Quelltabelle.

Daten einfügen

Der EMPLOYEES-Tabelle neue Mitarbeiterinformationen hinzufügen

Beispiel:

```
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 © 2007, Oracle. Alle Rechte vorbehalten.

Daten einfügen

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

- SQL-Funktionen wie `USER` und `CURRENT_DATE` verwenden
- Primärschlüsselwerte mit vorhandenen Datenbank-Sequenzen generieren
- Werte im PL/SQL-Block ableiten

Hinweis: Die Daten in der `employees`-Tabelle müssen unverändert bleiben. Die `employees`-Tabelle ist zwar nicht schreibgeschützt, Sie dürfen jedoch keine Daten einfügen, aktualisieren oder löschen, um eine konsistente Ausgabe zu gewährleisten.

Daten aktualisieren

Das Gehalt aller Mitarbeiter mit der Job-Kennung "Stock Clerk" (ST_CLERK) erhöhen

Beispiel:

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

ORACLE®

Copyright © 2007, Oracle. Alle Rechte vorbehalten.

Daten aktualisieren

In der SET-Klausel der UPDATE-Anweisung können mehrdeutige Identifier auftreten. Während der Identifier auf der linken Seite des Zuweisungsoperators immer ein Spaltenname ist, kann der Identifier auf der rechten Seite ein Spaltenname oder eine PL/SQL-Variable sein. Wie bereits erläutert sucht der Oracle-Server bei identischen Spalten- und Identifier-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, wird (im Gegensatz zur SELECT-Anweisung in PL/SQL) kein Fehler ausgelöst.

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

Daten löschen

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

Beispiel:

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

ORACLE®

Copyright © 2007, Oracle. Alle Rechte vorbehalten.

Daten löschen

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 copy_emp-Tabelle einfügen oder entsprechend der employees-Tabelle 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 © 2007, Oracle. Alle Rechte vorbehalten.

Zeilen zusammenführen

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 employee_id in der copy_emp-Tabelle mit der employee_id in der employees-Tabelle abgeglichen. Wenn eine Übereinstimmung gefunden wird, wird die Zeile in der copy_emp-Tabelle an die Zeile in der employees-Tabelle angepasst. Falls die Zeile in der copy_emp-Tabelle nicht gefunden wird, wird sie eingefügt.

Das komplette Beispiel, wie die MERGE-Anweisung in einem PL/SQL-Block verwendet wird, finden Sie auf der nächsten Seite.

Zeilen zusammenführen (Fortsetzung)

```
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;  
/
```

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 Typen:
 - Implizite Cursor: Werden vom Oracle-Server erstellt und verwaltet, um SQL-Anweisungen zu verarbeiten
 - Explizite Cursor: Werden explizit vom Programmierer deklariert

ORACLE

Copyright © 2007, Oracle. Alle Rechte vorbehalten.

SQL-Cursor

Sie haben bereits gelernt, dass Sie SQL-Anweisungen, die eine einzelne Zeile zurückgeben, in einen PL/SQL-Block aufnehmen können. Nehmen Sie die von der SQL-Anweisung abgerufenen Daten mit der INTO-Klausel in Variablen auf.

Wo verarbeitet der Oracle-Server SQL-Anweisungen?

Der Oracle-Server weist einen privaten Speicherbereich zu, der als *Kontextbereich* bezeichnet wird. In diesem Bereich werden SQL-Anweisungen analysiert und verarbeitet. Sämtliche 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 verarbeitet wird. PL/SQL erstellt einen impliziten Cursor, wenn der ausführbare Bereich des Blockes eine SQL-Anweisung ausgibt.

Cursor-Typen

Es gibt zwei Typen:

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

SQL-Cursor (Fortsetzung)

Cursor-Typen (Fortsetzung)

- **Explizite Cursor:** Als Programmierer müssen Sie möglicherweise mehrere Zeilen aus einer Datenbanktabelle abrufen, benötigen einen Zeiger zu jeder abzurufenden Zeile und müssen 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-Cursor-Attribute für implizite Cursor

Mit SQL-Cursor-Attributen können Sie das Ergebnis von SQL-Anweisungen testen.

<code>SQL%FOUND</code>	Boolesches Attribut, das TRUE ist, wenn die zuletzt durchgeführte SQL-Anweisung mindestens eine Zeile zurückgegeben hat
<code>SQL%NOTFOUND</code>	Boolesches Attribut, das den Wert TRUE hat, wenn die zuletzt durchgeführte SQL-Anweisung keine Zeile zurückgegeben hat
<code>SQL%ROWCOUNT</code>	Eine Ganzzahl für die Anzahl der Zeilen, auf die sich die letzte SQL-Anweisung auswirkt

ORACLE

Copyright © 2007, Oracle. Alle Rechte vorbehalten.

SQL-Cursor-Attribute für implizite Cursor

Mit SQL-Cursor-Attributen 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 sich eine DML-Anweisung auf keine Zeilen der zu Grunde liegenden Tabelle auswirkt. Wenn jedoch eine SELECT-Anweisung keine Zeilen abruft, gibt PL/SQL eine Exception zurück.

Beachten Sie, dass den Attributen `SQL` vorangestellt ist. Diese Cursor-Attribute werden mit impliziten Cursorn verwendet, die PL/SQL automatisch erstellt und deren Namen Sie nicht kennen. Daher verwenden Sie an Stelle des Cursor-Namens `SQL`.

Das `SQL%NOTFOUND`-Attribut bewirkt das Gegenteil vom `SQL%FOUND`-Attribut. 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.

Auf explizite Cursor-Attribute wird im Verlauf dieses Kurses noch näher eingegangen.

SQL-Cursor-Attribute für implizite Cursor

Zeilen mit der angegebenen employee_ID aus der employees-Tabelle löschen

Anzahl der gelöschten Zeilen ausgeben

```
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;
```

ORACLE

Copyright © 2007, Oracle. Alle Rechte vorbehalten.

SQL-Cursor-Attribute für implizite Cursor (Fortsetzung)

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

Zusammenfassung

In dieser Lektion haben Sie Folgendes gelernt:

- **DML-Anweisungen, Steueranweisungen für Transaktionen und DLL-Anweisungen in PL/SQL einbetten**
- **Die INTO-Klausel verwenden, die für alle SELECT-Anweisungen in PL/SQL obligatorisch ist**
- **Zwischen impliziten und expliziten Cursorn unterscheiden**
- **Mit SQL-Cursor-Attributen das Ergebnis von SQL-Anweisungen bestimmen**

ORACLE®

Copyright © 2007, Oracle. Alle Rechte vorbehalten.

Zusammenfassung

Die DML-Befehle und Steueranweisungen für Transaktionen 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 Cursoren. Implizite Cursor werden vom Oracle-Server erstellt und verwaltet, um SQL-Anweisungen auszuführen. Sie können die Cursor gemeinsam mit SQL-Cursor-Attributen verwenden, um das Ergebnis der SQL-Anweisung zu bestimmen. Explizite Cursor werden von Programmierern deklariert.

Übungen zu Lektion 4 – Überblick

Diese Übungen behandeln folgende Themen:

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

ORACLE®

Copyright © 2007, Oracle. Alle Rechte vorbehalten.

Oracle Internal & Oracle Academy
Use Only

Übungen zu Lektion 4

1. Erstellen Sie einen PL/SQL-Block, der in der departments-Tabelle die höchste Abteilungsnummer wählt und in die Variable v_max_deptno speichert. Zeigen Sie die höchste Abteilungsnummer an.
 - a. Deklarieren Sie die Variable v_max_deptno des NUMBER-Typs im deklarativen Bereich.
 - b. Starten Sie den ausführbaren Bereich mit dem BEGIN-Schlüsselwort. Nehmen Sie eine SELECT-Anweisung auf, mit der Sie die höchste department_id aus der departments-Tabelle abrufen können.
 - c. Zeigen Sie v_max_deptno an, und beenden Sie den ausführbaren Bereich.
 - d. Führen Sie Ihr Skript aus, und speichern Sie es unter dem Namen lab_04_01_soln.sql. Hier die Ausgabe:

```
anonymous block completed
The maximum department_id is : 270
```

2. Ändern Sie den in der 1. Übung erstellten PL/SQL-Block, um eine neue Abteilung in die departments-Tabelle einzufügen.
 - a. Laden Sie das Skript lab_04_01_soln.sql. Deklarieren Sie zwei Variablen:
v_dept_name des Typs departments.department_name
v_dept_id des NUMBER-Typs
Weisen Sie der Variablen v_dept_name im deklarativen Bereich die Bezeichnung Education zu.
 - b. Sie haben die derzeit höchste Abteilungsnummer bereits aus der departments-Tabelle abgerufen. Erhöhen Sie die Zahl um 10, und weisen Sie das Ergebnis v_dept_id zu.
 - c. Nehmen Sie eine INSERT-Anweisung auf, um Daten in die Spalten department_name, department_id und location_id der departments-Tabelle einzufügen.
Verwenden Sie Werte in v_dept_name und v_dept_id für department_name bzw. department_id, und verwenden Sie NULL für location_id.
 - d. Mit dem SQL-Attribut SQL%ROWCOUNT können Sie die Anzahl der betroffenen Zeilen anzeigen.
 - e. Führen Sie eine SELECT-Anweisung aus, um zu prüfen, ob die neue Abteilung eingefügt wurde. Sie können den PL/SQL-Block mit "/" abschließen und die SELECT-Anweisung in Ihr Skript aufnehmen.
 - f. Führen Sie Ihr Skript aus, und speichern Sie es unter dem Namen lab_04_02_soln.sql. Hier die Ausgabe:

```
anonymous block completed
The maximum department_id is : 280
SQL%ROWCOUNT gives 1
```

DEPARTMENT_ID	DEPARTMENT_NAME	MANAGER_ID	LOCATION_ID
280	Education	(null)	(null)

Übungen zu Lektion 4 (Fortsetzung)

3. In der 2. Übung stellen Sie `location_id` auf `NULL` ein. Erstellen Sie einen PL/SQL-Block, der die `location_id` für die neue Abteilung auf `3000` aktualisiert. Mit der Bind-Variablen `dept_id` können Sie die Zeile aktualisieren.

Hinweis: Überspringen Sie Schritt (a), wenn Sie für diese Übung keine neue Session gestartet haben.

- a. Wenn Sie eine neue Session gestartet haben, löschen Sie die in die `departments`-Tabelle eingefügte Abteilung, und führen Sie das Skript `lab_04_02_soln.sql` aus.
- b. Starten Sie den ausführbaren Bereich des Blockes mit dem `BEGIN`-Schlüsselwort. Nehmen Sie die `UPDATE`-Anweisung auf, um die `location_id` für die neue Abteilung (`dept_id = 280`) auf `3000` einzustellen.
- c. Beenden Sie den ausführbaren Bereich des Blockes mit dem `END`-Schlüsselwort. Schließen Sie den PL/SQL-Block mit `"/"` ab. Nehmen Sie eine `SELECT`-Anweisung auf, um die aktualisierte Abteilung anzuzeigen.
- d. Nehmen Sie schließlich eine `DELETE`-Anweisung auf, um die hinzugefügte Abteilung zu löschen.
- e. Führen Sie Ihr Skript aus, und speichern Sie es unter dem Namen `lab_04_01_soln.sql`. Hier die Ausgabe:

```
anonymous block completed
DEPARTMENT_ID DEPARTMENT_NAME MANAGER_ID LOCATION_ID
-----
280          Education           3000
1 rows selected
1 rows deleted
```


Kontrollstrukturen erstellen

5

ORACLE®

Copyright © 2007, Oracle. Alle Rechte vorbehalten.

Oracle Internal & Oracle Academy
Use Only

Ziele

Nach Ablauf dieser Lektion haben Sie folgende Ziele erreicht:

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

ORACLE®

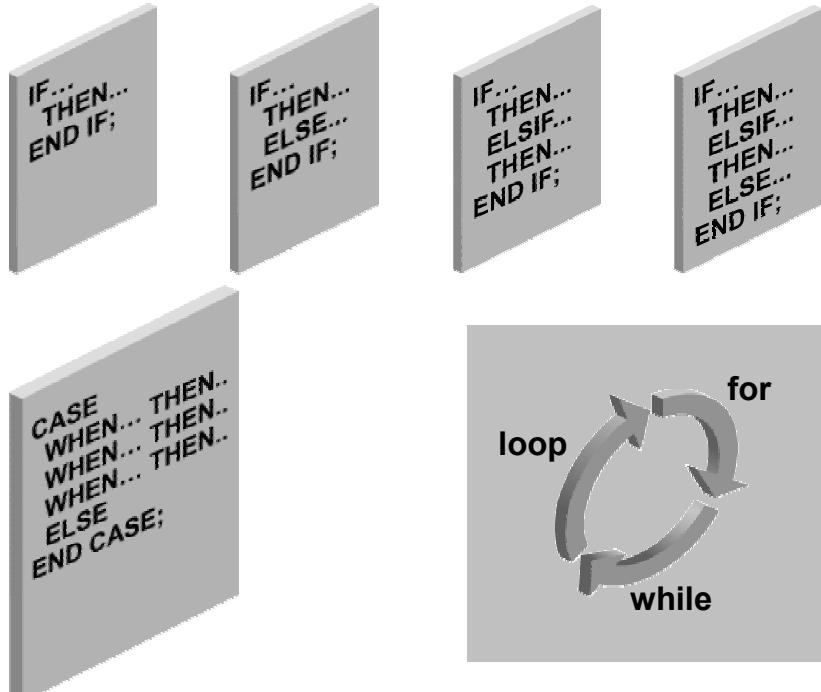
Copyright © 2007, Oracle. Alle Rechte vorbehalten.

Lernziel

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 Bereich von Blöcken 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 © 2007, Oracle. Alle Rechte vorbehalten.

Ablauf der Ausführung steuern

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

IF-Anweisungen

Syntax:

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

ORACLE

Copyright © 2007, Oracle. Alle Rechte vorbehalten.

IF-Anweisungen

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

Für die Syntax gilt:

<i>condition</i>	Eine boolesche Variable oder ein Ausdruck, die/der TRUE, FALSE oder NULL zurückgibt
THEN	Führt eine Klausel ein, die den booleschen Ausdruck mit der nachfolgenden Anweisungsfolge verknüpft
<i>statements</i>	Steht für eine oder mehrere PL/SQL- oder SQL-Anweisungen. (Diese können weitere IF-Anweisungen mit mehreren verschachtelten IF-, ELSE- und ELSIF-Anweisungen aufnehmen.) Die Anweisungen in der THEN-Klausel werden nur ausgeführt, wenn die Bedingung in der damit verknüpften IF-Klausel TRUE ist.

IF-Anweisung (Fortsetzung)

Für die Syntax gilt:

- ELSIF Ein Schlüsselwort, das einen booleschen Ausdruck einführt. (Wenn die erste Bedingung FALSE oder NULL ergibt, führt ELSIF zusätzliche Bedingungen ein.)
- ELSE Führt die Default-Klausel ein, die nur dann ausgeführt wird, wenn keines der zuvor mit IF und ELSIF eingeführten Prädikate TRUE ist. Die Tests werden der Reihe nach ausgeführt, so dass ein späteres Prädikat, das TRUE ist, von einem früheren Prädikat mit dem Wert TRUE verdrängt wird.
- END IF END IF markiert das Ende einer IF-Anweisung.

Hinweis: ELSIF und ELSE sind in IF-Anweisungen optional. Sie können in IF-Anweisungen eine beliebige Anzahl von ELSIF-Schlüsselwörtern verwenden, aber nur ein ELSE-Schlüsselwort. 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 ;
/
```

Ausgabe:

```
anonymous block completed
```

ORACLE

Copyright © 2007, Oracle. Alle Rechte vorbehalten.

Einfache IF-Anweisungen

Das Beispiel auf der Folie zeigt eine einfache IF-Anweisung mit der THEN-Klausel. Die v_myage-Variable wird auf 31 initialisiert. Die Bedingung für die IF-Anweisung gibt FALSE zurück, da v_myage nicht kleiner als 11 ist. Aus diesem Grund wird die THEN-Klausel nie erreicht. Fügen Sie jetzt diesem Beispiel Code hinzu, um zu sehen, wie ELSE und ELSIF verwendet werden.

Eine IF-Anweisung kann mehrere bedingte Ausdrücke enthalten, die mit logischen Operatoren wie AND, OR und NOT verbunden sind. Beispiel:

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

```

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

Anweisung IF THEN ELSE

```
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 ;
/
```

Ausgabe:

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

ORACLE

Copyright © 2007, Oracle. Alle Rechte vorbehalten.

Anweisung IF THEN ELSE

Auf der vorherigen Folie wurde dem Code eine 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 fortgefahren. Die Ausgabe des Blockes ist auf der Folie dargestellt.

Klausel IF ELSIF ELSE

```
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 ;
/
```

Ausgabe:

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

ORACLE

Copyright © 2007, Oracle. Alle Rechte vorbehalten.

Klausel IF ELSIF ELSE

Die IF-Klausel enthält jetzt mehrere ELSIF-Klauseln und eine ELSE-Klausel. Die ELSIF-Klauseln können im Gegensatz zur ELSE-Klausel Bedingungen haben. Der Bedingung für ELSIF muss die THEN-Klausel folgen. Sie wird ausgeführt, wenn die Bedingung der ELSIF-Klausel TRUE zurückgibt.

Wenn Sie mehrere ELSIF-Klauseln haben und die erste Bedingung FALSE oder NULL ist, wird mit der nächsten ELSIF-Klausel fortgefahrene 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 letzte ELSE-Klausel ist optional.

NULL-Wert in IF-Anweisung

```
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 © 2007, Oracle. Alle Rechte vorbehalten.

NULL-Wert in IF-Anweisung

Im Beispiel auf der Folie wird die `v_myage`-Variable 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 der Basis von Bedingungen ausgeführt werden.
- Beachten Sie die Schreibweise der Schlüsselwörter:
 - `ELSIF` in einem Wort
 - `END IF` in zwei Wörtern
- Wenn die boolesche Kontrollbedingung `TRUE` ist, 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.

CASE-Ausdrücke

- CASE-Ausdrücke wählen ein Ergebnis aus und geben es zurück.
- Um das Ergebnis auszuwählen, verwendet ein CASE-Ausdruck verschiedene 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;
/
```

ORACLE

Copyright © 2007, Oracle. Alle Rechte vorbehalten.

CASE-Ausdrücke

CASE-Ausdrücke geben ein Ergebnis auf der Basis einer oder mehrerer Alternativen zurück. Um das Ergebnis zurückzugeben, verwendet der CASE-Ausdruck einen *Selector*. Anhand des Wertes dieses Ausdrucks wird eine von mehreren Alternativen zurückgegeben. 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 bietet auch eine Searched CASE-Anweisung:

```
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. Darüber hinaus enthalten ihre WHEN-Klauseln Suchbedingungen, die einen booleschen Wert ergeben, und keine Ausdrücke, die Werte eines beliebigen Typs zurückgeben.

CASE-Ausdrücke – Beispiel

```
SET VERIFY OFF
DECLARE
    v_grade  CHAR(1) := UPPER('&grade');
    appraisal VARCHAR2(20);
BEGIN
    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 '|| appraisal);
END;
/
```

ORACLE

Copyright © 2007, Oracle. Alle Rechte vorbehalten.

CASE-Ausdrücke – Beispiel

Im Beispiel auf der Folie verwendet der CASE-Ausdruck den Wert der v_grade-Variablen als Ausdruck. Dieser Wert wird vom Benutzer mit einer Substitutionsvariablen angenommen. Der CASE-Ausdruck gibt entsprechend dem vom Benutzer eingegebenen Wert den Wert der appraisal-Variablen zurück, der auf dem Wert der v_grade-Variablen basiert. Wenn Sie für v_grade den Wert a oder A eingeben, lautet die Ausgabe im Beispiel wie folgt:

```
anonymous block completed
Grade: A    Appraisal Excellent
```

Searched CASE-Ausdrücke

```
DECLARE
    v_grade  CHAR(1) := UPPER('&grade');
    appraisal VARCHAR2(20);
BEGIN
    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 ' || appraisal);
END;
/
```

ORACLE

Copyright © 2007, Oracle. Alle Rechte vorbehalten.

Searched CASE-Ausdrücke

Das vorangegangene Beispiel enthielt einen einzelnen Testausdruck, nämlich die `v_grade`-Variable.

Die WHEN-Klausel verglich 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.

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'|| deptname ||
    ' department. There are'||v_emps ||' employees in this
    department');
END;
/
```

ORACLE

Copyright © 2007, Oracle. Alle Rechte vorbehalten.

CASE-Anweisungen

Erinnern Sie sich daran, wie die IF-Anweisung verwendet wurde. Sie können in die Klauseln THEN und ELSE n PL/SQL-Anweisungen aufnehmen. Ebenso können Sie Anweisungen in die CASE-Anweisung aufnehmen. Die CASE-Anweisung ist im Vergleich zu mehreren IF- und ELSIF-Anweisungen leichter lesbar.

CASE-Ausdrücke und CASE-Anweisungen – Vergleich

CASE-Ausdrücke bewerten die Bedingung und geben einen Wert zurück, während CASE-Anweisungen die Bedingung bewerten 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;

NULL-Werte verwenden

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

- **Einfache Vergleiche mit NULL-Werten ergeben stets NULL.**
- **Das Anwenden des logischen NOT-Operators auf einen NULL-Wert ergibt NULL.**
- **Wenn die Bedingung in IF-Anweisungen NULL ergibt, wird die zugehörige Anweisungsfolge nicht ausgeführt.**

ORACLE

Copyright © 2007, Oracle. Alle Rechte vorbehalten.

NULL-Werte verwenden

Betrachten Sie folgendes 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 ergibt die IF-Bedingung NULL und die Anweisungsfolge wird 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, so dass die IF-Bedingung NULL ergibt und die Anweisungsfolge umgangen wird.

Logiktabellen

Eine 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 © 2007, Oracle. Alle Rechte vorbehalten.

Logiktabellen

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

Sie können eine komplexe boolesche Bedingung erstellen, indem Sie einfache boolesche Bedingungen mit den logischen Operatoren AND, OR und NOT kombinieren. Mit den logischen Operatoren werden die booleschen Variablenwerte geprüft und TRUE, FALSE oder NULL zurückgegeben. Auf der Folie werden die logischen Tabellen gezeigt:

- FALSE hat Vorrang vor AND-Bedingungen und TRUE Vorrang vor OR-Bedingungen.
- AND gibt nur TRUE zurück, wenn beide Operanden TRUE sind.
- OR gibt nur 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 Verneinung von NULL (NOT NULL) ergibt einen NULL-Wert, weil NULL-Werte unbestimmt sind.

Boolesche Bedingungen

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 © 2007, Oracle. Alle Rechte vorbehalten.

Boolesche Bedingungen

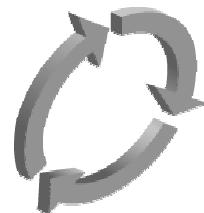
Mit der AND-Logiktabelle können Sie die möglichen Ergebnisse der booleschen Bedingung auf der Folie beurteilen.

Antwort

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

Kontrollierte Iteration – LOOP-Anweisungen

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



ORACLE

Copyright © 2007, Oracle. Alle Rechte vorbehalten.

Kontrollierte Iteration – LOOP-Anweisungen

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 zweite Typ von Kontrollstrukturen. PL/SQL bietet die folgenden Schleifentypen:

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

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

Basisschleifen

Syntax:

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

ORACLE

Copyright © 2007, Oracle. Alle Rechte vorbehalten.

Basisschleifen

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 END LOOP-Anweisung 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 die EXIT-Anweisung ist die Schleife unendlich.

EXIT-Anweisung

Mit der EXIT-Anweisung können Sie Schleifen abschließen. Die Anweisung nach der END LOOP-Anweisung wird ausgeführt. Sie können die EXIT-Anweisung als Aktion innerhalb einer IF-Anweisung oder als eigenständige Anweisung innerhalb der Schleife ausgeben. Die EXIT-Anweisung muss in einer Schleife platziert sein. 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 bewertet. Ergibt die Bedingung TRUE, wird die Schleife beendet und die nächste Anweisung nach der Schleife ausgeführt. Basisschleifen können mehrere EXIT-Anweisungen enthalten, es wird jedoch empfohlen, nur einen EXIT-Punkt zu haben.

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 © 2007, Oracle. Alle Rechte vorbehalten.

Basisschleifen (Fortsetzung)

Das auf der Folie gezeigte Beispiel einer Basisschleife wird wie folgt definiert:

Fügen Sie für den Länder-Code CA und die Stadt Montreal drei neue Standortnummern ein.

Hinweis: Die Anweisungen einer Basisschleife werden ausgeführt, bis die EXIT WHEN-Bedingung erfüllt ist.

Ist die Bedingung so in der Schleife platziert, dass sie erst nach den Schleifenanweisungen geprüft wird, wird die Schleife mindestens ein Mal 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.

WHILE-Schleifen

Syntax:

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

Mit der WHILE-Schleife können Sie Anweisungen wiederholen, so lange eine Bedingung TRUE ist.

ORACLE®

Copyright © 2007, Oracle. Alle Rechte vorbehalten.

WHILE-Schleifen

Mit WHILE-Schleifen können Sie eine Anweisungsfolge wiederholen, bis die Kontrollbedingung nicht mehr TRUE ist. Die Bedingung wird am Anfang jeder Iteration bewertet. Die Schleife wird abgeschlossen, wenn die Bedingung FALSE oder NULL ist. 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:

<i>condition</i>	Eine boolesche Variable oder ein boolescher Ausdruck (TRUE, FALSE oder NULL)
<i>statement</i>	Steht für eine oder mehrere PL/SQL- oder SQL-Anweisungen

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

Hinweis: Wenn die Bedingung NULL ergibt, wird die Schleife umgangen und die nächste Anweisung 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 © 2007, Oracle. Alle Rechte vorbehalten.

WHILE-Schleifen (Fortsetzung)

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

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

FOR-Schleifen

- Mit einer FOR-Schleife können Sie die Prüfung der Anzahl von Iterationen abkürzen.
- Deklarieren Sie den Zähler nicht, da er implizit deklariert wird.

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

ORACLE

Copyright © 2007, Oracle. Alle Rechte vorbehalten.

FOR-Schleifen

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

Für die Syntax gilt:

counter Eine implizit deklarierte Ganzzahl, deren Wert bei jeder Iteration der Schleife automatisch um 1 erhöht oder verringert wird (letzteres mit dem REVERSE-Schlüsselwort), bis die Unter- oder Obergrenze erreicht ist.

REVERSE Führt dazu, dass sich die Zählerwerte bei jeder Iteration von der Ober- zur Untergrenze verringern

Hinweis: Die Untergrenze wird dennoch als erstes referenziert.

lower_bound Gibt die Untergrenze für den Bereich der Zählerwerte an

upper_bound Gibt die Obergrenze für den Bereich der Zählerwerte an

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

FOR-Schleifen (Fortsetzung)

Hinweis: Die Anweisungsfolge wird bei jeder Inkrementierung des Zählers entsprechend den beiden Grenzen ausgeführt. Die Ober- und Untergrenze des Schleifenbereichs können Literale, Variablen oder Ausdrücke sein. Sie müssen jedoch Ganzzahlen ergeben. Die Grenzen werden auf Ganzzahlen gerundet, d. h. 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.

Diese Anweisung wird beispielsweise nur ein Mal 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 © 2007, Oracle. Alle Rechte vorbehalten.

FOR-Schleifen (Fortsetzung)

Sie haben bereits gelernt, wie Sie mit Hilfe der Basisschleife und der WHILE-Schleife drei neue Orte für den Länder-Code CA und die Stadt Montreal einfügen. Auf dieser Folie wird gezeigt, wie Sie dies mit der FOR-Schleife erreichen.

FOR-Schleifen

Richtlinien

- **Referenzieren Sie den Zähler nur innerhalb der Schleife, da er außerhalb nicht definiert ist.**
- **Dem Zähler kann kein Wert zugewiesen werden.**
- **Keine der Schleifengrenzen darf NULL sein.**

ORACLE®

Copyright © 2007, Oracle. Alle Rechte vorbehalten.

FOR-Schleifen (Fortsetzung)

Auf der Folie werden die Richtlinien für das Erstellen von FOR-Schleifen aufgeführt.

Hinweis: Als Unter- und Obergrenze einer LOOP-Anweisung müssen keine numerischen Literale verwendet werden. Es können auch Ausdrücke verwendet werden, 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;
/
```

Richtlinien für Schleifen

- **Verwenden Sie eine Basisschleife, wenn die Anweisungen innerhalb der Schleife mindestens ein Mal ausgeführt werden müssen.**
- **Verwenden Sie eine WHILE-Schleife, wenn die Bedingung am Anfang jeder Iteration bewertet werden muss.**
- **Verwenden Sie eine FOR-Schleife, wenn die Anzahl der Iterationen bekannt ist.**

ORACLE

Copyright © 2007, Oracle. Alle Rechte vorbehalten.

Richtlinien für Schleifen

Mit einer Basisschleife können Sie die zugehörige Anweisung mindestens ein Mal ausführen, selbst dann, wenn die Bedingung beim Aufrufen der Schleife bereits erfüllt ist. Ohne die EXIT-Anweisung ist 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 bewertet. Die Schleife wird abgeschlossen, wenn die Bedingung FALSE ist. Ist die Bedingung zu Beginn der Schleife FALSE, werden keine weiteren Iterationen durchgeführt.

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

Verschachtelte Schleifen und Labels

- Sie können Schleifen mehrfach verschachteln.
- Verwenden Sie Labels, um zwischen Blöcken und Schleifen zu unterscheiden.
- Beenden Sie die äußere Schleife mit der EXIT-Anweisung, die das Label referenziert.

ORACLE®

Copyright © 2007, Oracle. Alle Rechte vorbehalten.

Verschachtelte Schleifen und Labels

Sie können FOR-, WHILE- und Basisschleifen 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 der EXIT-Anweisung beenden.

Für Label-Namen gelten dieselben Regeln wie für andere Identifier. Platzieren Sie Labels vor Anweisungen, entweder in derselben oder einer separaten Zeile. Leerzeichen werden bei PL/SQL-Analysen nur in Literalen berücksichtigt. Kennzeichnen Sie Basisschleifen, indem Sie das Label zwischen Begrenzungszeichen (<<Label>>) vor dem Wort LOOP platzieren. In FOR- und WHILE-Schleifen müssen Sie das Label vor FOR oder WHILE platzieren.

Wenn die Schleife mit einem Label gekennzeichnet ist, können Sie den Label-Namen zur übersichtlicheren Strukturierung nach der END LOOP-Anweisung einfügen.

Verschachtelte Schleifen und Labels

```
...
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 © 2007, Oracle. Alle Rechte vorbehalten.

Verschachtelte Schleifen und Labels (Fortsetzung)

Im Beispiel auf der Folie sind zwei Schleifen vorhanden. Die äußere Schleife wird mit dem Label <<Outer_Loop>> und die innere Schleife mit dem Label <<Inner_Loop>> identifiziert. Die Identifier werden zwischen Begrenzungszeichen (<<Label>>) vor dem Wort LOOP platziert. Die innere Schleife ist in der äußeren Schleife verschachtelt. Die Label-Namen werden zur übersichtlichen Strukturierung nach den END LOOP-Anweisungen eingefügt.

PL/SQL-Anweisung CONTINUE

- **Definition**

- Fügt die Funktionalität zum Ausführen der nächsten Schleifeniteration hinzu
- Ermöglicht Programmierern, die nächste Schleifeniteration auszuführen
- Verwendet für die EXIT-Anweisung eine parallele Struktur und Semantik

- **Vorteile**

- Erleichtert die Programmierung
- Verbessert die bisherige Simulierung der CONTINUE-Anweisung.



ORACLE

Copyright © 2007, Oracle. Alle Rechte vorbehalten.

PL/SQL-Anweisung CONTINUE

Mit der CONTINUE-Anweisung können Sie die Kontrolle innerhalb einer Schleife an eine neue Iteration zurückgeben oder die Schleife verlassen. Diese Funktion ist in zahlreichen Programmiersprachen enthalten. In Oracle Database 11g bietet auch PL/SQL diese Funktion. Vor Oracle Database 11g bestand die Möglichkeit, die CONTINUE-Programmfunktion mit booleschen Variablen und Bedingungsanweisungen zu simulieren. In einigen Fällen ist diese Lösung ineffizient. Mit der CONTINUE-Anweisung können Sie Schleifeniterationen auf einfachere Weise kontrollieren. Sie ist mitunter effizienter als frühere Codierlösungen.

Die CONTINUE-Anweisung wird häufig verwendet, um Daten in einer Schleife vor dem Hauptverarbeitungsprozess zu filtern.

PL/SQL-Anweisung CONTINUE – 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;
        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 © 2007, Oracle. Alle Rechte vorbehalten.

PL/SQL-Anweisung CONTINUE – Beispiel

Die erste TOTAL-Zuweisung wird für jede der zehn Schleifeniterationen ausgeführt.

Die zweite TOTAL-Zuweisung wird für die ersten fünf Schleifeniterationen ausgeführt. Die CONTINUE-Anweisung gibt die Kontrolle innerhalb einer Schleife an eine neue Iteration zurück, so dass die zweite TOTAL-Zuweisung nicht für die letzten fünf Schleifeniterationen ausgeführt wird.

Das Endergebnis der TOTAL-Variablen ist 70.

PL/SQL-Anweisung CONTINUE – 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;
END two_loop;
```

```
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 © 2007, Oracle. Alle Rechte vorbehalten.

PL/SQL-Anweisung CONTINUE – Beispiel (Fortsetzung)

Sie können mit der CONTINUE-Anweisung zur nächsten Iteration einer äußeren Schleife springen. Kennzeichnen Sie 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 TRUE ist (ebenso wie das EXIT-Schlüsselwort). 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 BeforeTopLoop-Label übergeben.

Wenn dieses Schleifenpaar abgeschlossen ist, hat die TOTAL-Variable den Wert 20.

Sie können die CONTINUE-Anweisung auch in einem inneren Codeblock verwenden, der keine Schleife enthält, vorausgesetzt, der Block ist in einer entsprechenden äußeren Schleife verschachtelt.

Einschränkungen

- Die CONTINUE-Anweisung kann niemals außerhalb einer Schleife auftreten. Dies generiert einen Compiler-Fehler.
- Sie können mit der CONTINUE-Anweisung nicht die Grenzen von Prozeduren, Funktionen oder Methoden überschreiten. Dies generiert einen Compiler-Fehler.

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**

ORACLE®

Copyright © 2007, Oracle. Alle Rechte vorbehalten.

Zusammenfassung

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 IF-Konstrukt können Sie Anweisungen bedingt ausführen.

Kontrollierte Iterationskonstrukte führen Anweisungsfolgen wiederholt aus, bis eine angegebene Bedingung TRUE ist. Iterative Operationen führen Sie mit den verschiedenen Schleifenkonstrukten aus.

Übungen zu Lektion 5 – Überblick

Diese Übungen behandeln folgende Themen:

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

ORACLE

Copyright © 2007, Oracle. Alle Rechte vorbehalten.

Übungen zu Lektion 5 – Überblick

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.

Übungen zu Lektion 5

1. Führen Sie den Befehl in der Datei lab_05_01.sql aus, um die messages-Tabelle zu erstellen. Erstellen Sie einen PL/SQL-Block, um Zahlen in die messages-Tabelle einzufügen.
 - a. Fügen Sie die Zahlen 1 bis 10 außer 6 und 8 ein.
 - b. Führen Sie vor dem Ende des Blockes einen Commit-Befehl durch.
 - c. Führen Sie eine SELECT-Anweisung aus, um sicherzustellen, dass Ihr PL/SQL-Block funktioniert.

Folgende Meldung sollte eingeblendet werden:

```
anonymous block completed
RESULTS
-----
1
2
3
4
5
7
9
10

8 rows selected
```

2. Führen Sie das Skript lab_05_02.sql aus. Dieses Skript erstellt eine emp-Tabelle, die ein Replikat der employees-Tabelle ist. Es fügt eine neue Spalte (stars, Datentyp VARCHAR2 und Größe 50) in die emp-Tabelle ein. Erstellen Sie einen PL/SQL-Block, der pro 1000 Euro des Mitarbeitergehalts ein Sternchen in die stars-Spalte einfügt. Speichern Sie Ihr Skript unter dem Namen lab_05_02_soln.sql.
 - a. Deklarieren Sie im deklarativen Bereich des Blockes eine v_empno-Variable des Typs emp.employee_id, und initialisieren Sie sie auf 176. Deklarieren Sie eine v_asterisk-Variable des emp.stars-Typs, und initialisieren Sie sie auf NULL. Erstellen Sie eine sal-Variable des emp.salary-Typs.
 - b. Erstellen Sie im ausführbaren Bereich die Logik, um der Zeichenfolge für jede 1.000 Euro Gehalt ein Sternchen (*) hinzuzufügen. Wenn der Mitarbeiter beispielsweise 8.000 Euro verdient, sollte die Zeichenfolge acht Sternchen enthalten. Verdient der Mitarbeiter 12.500 Euro, sollte die Zeichenfolge 13 Sternchen enthalten.
 - c. Aktualisieren Sie die stars-Spalte für den Mitarbeiter mit der Sternzeichenfolge. Führen Sie vor dem Ende des Blockes einen Commit-Befehl durch.

Übungen zu Lektion 5 (Fortsetzung)

- d. Zeigen Sie die Zeile der emp-Tabelle an, um sicherzustellen, dass der PL/SQL-Block erfolgreich ausgeführt wurde.
- e. Führen Sie Ihr Skript aus, und speichern Sie es unter dem Namen lab_05_02_soln.sql. Hier die Ausgabe:

```
anonymous block completed
EMPLOYEE_ID          SALARY          STARS
-----
176                  8600            *****
1 rows selected
```

Oracle Internal & Oracle Academy
Use Only

Zusammengesetzte Datentypen verwenden

ORACLE®

Copyright © 2007, Oracle. Alle Rechte vorbehalten.

Oracle Internal & Oracle Academy
Use Only

Ziele

Nach Ablauf dieser Lektion haben Sie folgende Ziele erreicht:

- Benutzerdefinierte PL/SQL-Records erstellen
- Records mit dem %ROWTYPE-Attribut erstellen
- INDEX BY-Tabellen erstellen
- INDEX BY-Tabellen mit Records erstellen
- Unterschied zwischen Records, Tabellen und Record-Tabellen erklären

ORACLE®

Copyright © 2007, Oracle. Alle Rechte vorbehalten.

Lernziel

Sie haben bereits eine Einführung in zusammengesetzte Datentypen erhalten. In dieser Lektion werden zusammengesetzte Datentypen und ihre Verwendung beschrieben.

Zusammengesetzte Datentypen

- Können mehrere Werte aufnehmen (im Gegensatz zu skalaren Typen)
- Es gibt zwei Arten:
 - PL/SQL-Records
 - PL/SQL-Collections
 - INDEX BY-Tabellen oder assoziative Arrays
 - Nested Tables
 - VARRAYS

ORACLE®

Copyright © 2007, Oracle. Alle Rechte vorbehalten.

Zusammengesetzte Datentypen

Sie haben gelernt, dass Variablen mit skalarem Datentyp nur einen Wert aufnehmen können, während Variablen mit zusammengesetztem Datentyp mehrere Werte vom skalaren oder zusammengesetzten Datentyp aufnehmen können. Es gibt zwei Arten von zusammengesetzten Datentypen:

- **PL/SQL-Records:** Mit Records können Sie verwandte Daten unterschiedlichen Typs als logische Einheit behandeln. PL/SQL-Records können Variablen unterschiedlichen Typs enthalten. Sie können beispielsweise einen Record definieren, der Mitarbeiterdetails aufnimmt. Zu diesem Zweck müssen Sie eine Personalnummer als NUMBER speichern, einen Vor- und Nachnamen als VARCHAR2 usw. Wenn Sie einen Record zum Speichern von Mitarbeiterdetails erstellen, erstellen Sie eine logische kollektive Einheit. Dies erleichtert den Zugriff und die Bearbeitung von Daten.
- **PL/SQL-Collections:** Mit Collections können Sie Daten als eine Einheit behandeln. Es gibt drei Arten von Collections:
 - INDEX BY-Tabellen oder assoziative Arrays
 - Nested Tables
 - VARRAYS

Zusammengesetzte Datentypen (Fortsetzung)

Zweck zusammengesetzter Datentypen

Sie haben alle verwandten Daten in einer Einheit vorliegen. Sie können einfach auf die Daten zugreifen und sie bearbeiten. Zusammengesetzte Daten lassen sich leichter verwalten, verknüpfen und übertragen. Als Analogie kann eine Tasche dienen. Sie können alle Laptop-Komponenten in einer einzigen Tasche verstauen und müssen nicht jede einzelne Komponente in einer separaten Tasche aufbewahren.

Oracle Internal & Oracle Academy
Use Only

Zusammengesetzte Datentypen

- **Mit PL/SQL-Records können Sie Werte unterschiedlichen Datentyps speichern, jeweils aber nur ein Vorkommen.**
- **Mit PL/SQL-Collections können Sie Werte desselben Datentyps speichern.**

ORACLE

Copyright © 2007, Oracle. Alle Rechte vorbehalten.

Zusammengesetzte Datentypen (Fortsetzung)

In welchen Fällen verwenden Sie PL/SQL-Records oder PL/SQL-Collections, die beide zusammengesetzte Datentypen sind?

Verwenden Sie PL/SQL-Records, wenn Sie Werte unterschiedlichen Datentyps speichern möchten, die logisch zusammengehören. Wenn Sie einen Record zur Aufnahme von Mitarbeiterdetails erstellen, geben Sie an, dass alle gespeicherten Werte verwandt sind, da sie Informationen zu einem bestimmten Mitarbeiter bereitstellen.

Mit PL/SQL-Collections können Sie Werte desselben Datentyps speichern. Dies kann auch ein zusammengesetzter Datentyp sein (z. B. Records). Sie können eine Collection definieren, die die Vornamen aller Mitarbeiter aufnimmt. Sie können n Namen in der Collection speichern, wobei jedoch der 1. Name nicht mit dem 2. Namen verwandt ist. Die einzige Beziehung besteht darin, dass beides Mitarbeiternamen sind. Diese Collections sind mit Arrays in Programmiersprachen wie C, C++ und Java vergleichbar.

PL/SQL-Records

- **Müssen ein oder mehrere Komponenten (*Felder*) eines skalaren, RECORD- oder INDEX BY-Tabellendatentyps 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 © 2007, Oracle. Alle Rechte vorbehalten.

PL/SQL-Records

Records sind Gruppen zusammenhängender Datenelemente, die unter ihrem speziellen Namen und Datentyp in Feldern gespeichert sind.

- Jeder definierte Record kann beliebig viele Felder enthalten.
- Sie können Records Ausgangswerte zuweisen und sie als NOT NULL definieren.
- Felder ohne Ausgangswerte werden mit NULL initialisiert.
- Sie können das DEFAULT-Schlüsselwort auch für Felddefinitionen verwenden.
- Sie können RECORD-Typen im deklarativen Bereich jedes Blockes, Unterprogramms oder Packages definieren und benutzerdefinierte Records deklarieren.
- Sie können 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 © 2007, Oracle. Alle Rechte vorbehalten.

PL/SQL-Records erstellen

PL/SQL-Records sind benutzerdefinierte zusammengesetzte Datentypen. Verwenden Sie sie wie folgt:

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

Für die Syntax gilt:

<i>type_name</i>	Der Name des RECORD-Typs. (Dieser Identifier deklariert Records.)
<i>field_name</i>	Der Name eines Feldes innerhalb des Records
<i>field_type</i>	Der Datentyp des Feldes. (Er repräsentiert alle PL/SQL-Datentypen außer REF CURSOR. Sie können die Attribute %TYPE und %ROWTYPE verwenden.)
<i>expr</i>	Der <i>field_type</i> oder ein Ausgangswert

Das NOT NULL-Constraint verhindert, dass diesen Feldern NULL-Werte zugewiesen werden. Stellen Sie sicher, dass Sie die NOT NULL-Felder initialisieren.

Auf REF CURSOR wird im Anhang "REF-Cursor" näher eingegangen.

PL/SQL-Records erstellen

Variablen deklarieren, um Name, Beruf und Gehalt eines neuen Mitarbeiters zu speichern

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;
```

ORACLE

Copyright © 2007, Oracle. Alle Rechte vorbehalten.

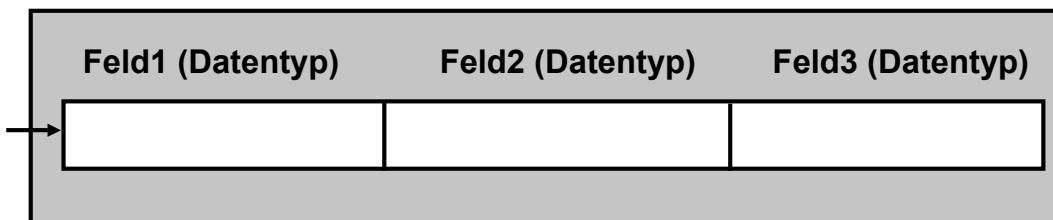
PL/SQL-Records erstellen (Fortsetzung)

Felddeklarationen zum Definieren von Records ähneln Variablen Deklarationen. Jedes Feld besitzt einen eindeutigen Namen und einen spezifischen Datentyp. Im Gegensatz zu skalaren Variablen gibt es für PL/SQL-Records keine vordefinierten Datentypen. Daher müssen Sie zuerst den Record-Typ erstellen und danach mit Hilfe dieses Typs einen Identifier deklarieren.

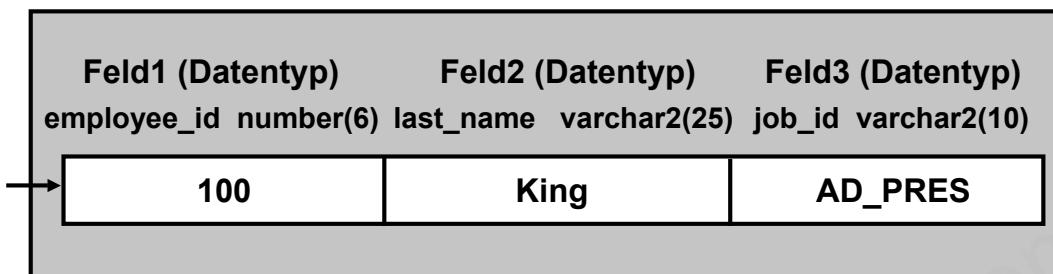
Im Beispiel auf der Folie wird ein Record-Typ (`t_rec`) definiert. Im nächsten Schritt wird ein Record (`v_myrec`) des `t_rec`-Typs deklariert.

Hinweis: Um zu verhindern, dass diesem Feld NULL-Werte zugewiesen werden, können Sie das NOT NULL-Constraint zu jeder Felddeklaration hinzufügen. Beachten Sie, dass Sie die mit NOT NULL deklarierten Felder initialisieren müssen.

Struktur von PL/SQL-Records



Beispiel:



ORACLE

Copyright © 2007, Oracle. Alle Rechte vorbehalten.

Struktur von PL/SQL-Records

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

record_name.field_name

Referenzieren Sie das job_id-Feld im emp_record-Record beispielsweise wie folgt:

emp_record.job_id

Anschließend können Sie dem Record-Feld einen Wert zuweisen:

emp_record.job_id := 'ST_CLERK';

Wenn Sie einen Block oder ein Unterprogramm starten, werden darin enthaltene benutzerdefinierte Records instanziert. Sie werden gelöscht, wenn Sie den Block oder das Unterprogramm beenden.

%ROWTYPE-Attribut

- Deklarieren Sie Variablen entsprechend einer Collection von Spalten in einer Datenbanktabelle oder -View.
- Stellen Sie %ROWTYPE die Datenbanktabelle oder -View voran.
- Felder im Record erhalten ihren Namen und Datentyp von den Spalten der Tabelle oder View.

Syntax:

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

ORACLE

Copyright © 2007, Oracle. Alle Rechte vorbehalten.

%ROWTYPE-Attribut

Sie haben gelernt, dass Sie mit %TYPE eine Variable eines Spaltentyps deklarieren können. Die Variable hat denselben Datentyp und dieselbe Größe wie die Tabellenspalte. Der Vorteil von %TYPE ist, dass Sie die Variable beibehalten können, wenn Sie die Spalte bearbeiten. Wenn die Variable in Berechnungen verwendet wird, müssen Sie außerdem nicht auf die Gesamtstelligenzahl achten.

Mit dem %ROWTYPE-Attribut können Sie einen Record deklarieren, der eine ganze Zeile einer Tabelle oder View aufnehmen kann. Die Felder im Record erhalten ihren Namen und Datentyp von den Spalten der Tabelle oder View. Der Record kann auch eine ganze Datenzeile speichern, die aus einem Cursor oder einer Cursor-Variablen gelesen wurde.

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

identifier Der für den ganzen Record gewählte Name

reference Der Name der Tabelle, der View, des Cursors bzw. der Cursor-Variablen, auf der/dem der Record basieren soll. (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;  
    ...
```

%ROWTYPE-Attribut (Fortsetzung)

Der emp_record-Record besteht aus den folgenden Feldern, von denen jedes eine Spalte der employees-Tabelle darstellt.

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,
 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
```

Referenzieren Sie das commission_pct-Feld im emp_record-Record beispielsweise wie folgt:

```
emp_record.commission_pct
```

Anschließend können Sie dem Record-Feld einen Wert zuweisen:

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

Records Werte zuweisen

Sie können einem Record 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 einen Record einem anderen Record zuweisen, wenn beide die gleichen Datentypen haben. Ein Record des Typs employees%ROWTYPE und ein benutzerdefinierter Record-Typ mit analogen Feldern der employees-Tabelle haben den gleichen Datentyp. Wenn ein benutzerdefinierter Record Felder enthält, die mit den Feldern eines %ROWTYPE-Records identisch sind, können Sie daher diesen benutzerdefinierten Record dem %ROWTYPE-Record zuweisen.

%ROWTYPE – Vorteile

- **Die Anzahl und Datentypen der zu Grunde liegenden Datenbankspalten sind unwichtig und können sich während der Laufzeit ändern.**
- **Das %ROWTYPE-Attribut ist nützlich, wenn Sie eine Zeile mit der SELECT *-Anweisung abrufen.**

ORACLE

Copyright © 2007, Oracle. Alle Rechte vorbehalten.

%ROWTYPE – Vorteile

Die Vorteile des %ROWTYPE-Attributs sind auf der Folie aufgelistet. Verwenden Sie das %ROWTYPE-Attribut, wenn Sie die Struktur der zu Grunde liegenden Datenbanktabelle nicht kennen.

Der Hauptvorteil des %ROWTYPE-Attributs ist, dass es die Verwaltung vereinfacht. Mit %ROWTYPE stellen Sie sicher, dass sich die Datentypen der mit dem Attribut deklarierten Variablen dynamisch ändern, wenn Sie die zu Grunde liegende Tabelle ändern. Wenn Sie die Spalten in einer Tabelle mit einer DDL-Anweisung ändern, wird die PL/SQL-Programmeinheit auf ungültig gesetzt. Wenn das Programm rekompiliert wird, spiegelt es automatisch das neue Tabellenformat wider.

Das %ROWTYPE-Attribut ist speziell dann nützlich, wenn Sie eine ganze Zeile aus einer Tabelle abrufen. Ohne dieses Attribut müssen Sie für jede der mit der SELECT-Anweisung abgerufenen Spalten eine Variable deklarieren.

%ROWTYPE-Attribut – 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;
/
```

ORACLE

Copyright © 2007, Oracle. Alle Rechte vorbehalten.

%ROWTYPE-Attribut – Beispiel

Die Folie zeigt ein Beispiel des %ROWTYPE-Attributs. Wenn ein Mitarbeiter in den Ruhestand geht, werden Informationen über diesen Mitarbeiter zu einer Tabelle mit Informationen über Mitarbeiter im Ruhestand hinzugefügt. Der Benutzer gibt die Personalnummer an. Der Record des vom Benutzer angegebenen Mitarbeiters wird aus der employees-Tabelle abgerufen und in der emp_rec-Variablen gespeichert, die mit dem %ROWTYPE-Attribut deklariert wird.

Die CREATE-Anweisung, mit der Sie die retired_emps-Tabelle 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))
```

In die retired_emps-Tabelle wird folgender Record eingefügt:

EMPNO	ENAME	JOB	MGR	HIREDATE	LEAVEDATE	SAL	COMM	DEPTNO
124	Mourgos	ST_MAN		100 16-NOV-99	17-MAY-07	5800	(null)	50

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;
```

ORACLE

Copyright © 2007, Oracle. Alle Rechte vorbehalten.

Records mit %ROWTYPE einfügen

Vergleichen Sie die `INSERT`-Anweisung auf der vorherigen Folie mit der `INSERT`-Anweisung auf dieser Folie. Der `emp_rec`-Record hat den `retired_emps`-Typ. Die Anzahl der Felder im Record muss mit der Anzahl der Feldnamen in der `INTO`-Klausel identisch sein. Mit diesem Record können Sie Werte in eine Tabelle einfügen. 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 `hire_date`-Wert in der `retired_emps`-Tabelle in das `leavedate`-Feld ein. Das Ruhestandsdatum kann nicht das Einstellungsdatum sein. Es wird der folgende Record eingefügt: (Auf der nächsten Folie sehen Sie, wie Sie die Zeile aktualisieren.)

EMPNO	ENAME	JOB	MGR	HIREDATE	LEAVEDATE	SAL	COMM	DEPTNO
124	Mourgos	ST_MAN	100	16-NOV-99	17-MAY-07	5800	(null)	50

Zeilen in einer Tabelle mit einem Record aktualisieren

```
SET VERIFY OFF
DECLARE
    v_employee_number number:= 124;
    v_emp_rec retired_emps%ROWTYPE;
BEGIN
    SELECT * INTO v_emp_rec FROM retired_emps;
    v_emp_rec.leavedate:=CURRENT_DATE;
    UPDATE retired_emps SET ROW = v_emp_rec WHERE
        empno=v_employee_number;
END;
/
SELECT * FROM retired_emps;
```

ORACLE

Copyright © 2007, Oracle. Alle Rechte vorbehalten.

Zeilen in einer Tabelle mit einem Record aktualisieren

Sie haben gelernt, wie Sie mit einem Record eine Zeile einfügen. Auf dieser Folie wird eine Zeile mit Hilfe eines Records aktualisiert. Das ROW-Schlüsselwort steht für die ganze Zeile. Der Code auf der Folie aktualisiert das Ruhestandsdatum des Mitarbeiters. Der Record wird aktualisiert.

	EMPNO	ENAME	JOB	MGR	HIREDATE	LEAVEDATE	SAL	COMM	DEPTNO
1	124	Mourgos	ST_MAN	100	16-NOV-99	17-MAY-07	5800	(null)	50

INDEX BY-Tabellen oder assoziative Arrays

- **PL/SQL-Strukturen mit zwei Spalten:**
 - Primärschlüssel des Ganzzahlen- oder Zeichenfolgen-Datentyps
 - Spalte des skalaren oder Record-Datentyps
- **Haben keine Größeneinschränkung. Die Größe hängt jedoch von den Werten ab, die der Schlüsseldatentyp aufnehmen kann.**

ORACLE

Copyright © 2007, Oracle. Alle Rechte vorbehalten.

INDEX BY-Tabellen oder assoziative Arrays

INDEX BY-Tabellen sind benutzerdefinierte, zusammengesetzte Datentypen (Collections). INDEX BY-Tabellen können Daten mit einem Primärschlüsselwert als Index speichern, wobei die Schlüsselwerte nicht zwingend sequenziell sind. INDEX BY-Tabellen sind Gruppen von Schlüsselwertpaaren.

INDEX BY-Tabellen haben nur zwei Spalten:

- Eine Spalte des Ganzzahlen- oder Zeichenfolgen-Datentyps, die als Primärschlüssel dient. Der Schlüssel kann numerisch sein: BINARY_INTEGER oder PLS_INTEGER. Die Schlüssel BINARY_INTEGER und PLS_INTEGER benötigen weniger Speicher als NUMBER. Sie können damit mathematische Ganzzahlen in kompakter Form darstellen und arithmetische Operationen mit Rechnerarithmetik implementieren. Arithmetische Operationen mit diesen Datentypen sind schneller als die NUMBER-Arithmetik. Der Schlüssel kann auch dem VARCHAR2-Typ oder einem seiner Subtypen angehören. In den Beispielen in diesem Kurs wird für die Schlüsselspalte der PLS_INTEGER-Datentyp verwendet.
- Eine Spalte des skalaren oder Record-Datentyps, in die Werte aufgenommen werden. Falls es sich um eine skalare Spalte handelt, kann sie nur einen Wert aufnehmen. Handelt es sich um eine Record-Spalte, kann sie mehrere Werte aufnehmen.

Für die INDEX BY-Tabellen gilt keine Größeneinschränkung. Der Schlüssel in der PLS_INTEGER-Spalte ist jedoch auf den Maximalwert eingeschränkt, den eine PLS_INTEGER-Spalte aufnehmen kann. Die Schlüssel können positiv und negativ sein. Die Schlüssel in INDEX BY-Tabellen sind nicht zwingend sequenziell.

INDEX BY-Tabellen erstellen

Syntax:

```
TYPE type_name IS TABLE OF
  {column_type | variable%TYPE
  | table.column%TYPE} [NOT NULL]
  | table%ROWTYPE
  [INDEX BY PLS_INTEGER | BINARY_INTEGER
  | VARCHAR2(<size>)];
identifier type_name;
```

INDEX BY-Tabelle deklarieren, um die Nachnamen von Mitarbeitern zu speichern:

```
...
TYPE ename_table_type IS TABLE OF
  employees.last_name%TYPE
  INDEX BY PLS_INTEGER;
...
ename_table ename_table_type;
```

ORACLE

Copyright © 2007, Oracle. Alle Rechte vorbehalten.

INDEX BY-Tabellen erstellen

Erstellen Sie INDEX BY-Tabellen in zwei Schritten.

1. Deklarieren Sie einen TABLE-Datentyp.
2. Deklarieren Sie eine Variable dieses Datentyps.

Für die Syntax gilt:

type_name Der Name des TABLE-Typs (Diese Typspezifikation wird in nachfolgenden Deklarationen der PL/SQL-Tabellen-Identifier verwendet.)

column_type Ein beliebiger skalarer oder zusammengesetzter Datentyp wie VARCHAR2, DATE, NUMBER oder %TYPE (Sie können den Spaltendatentyp mit dem %TYPE-Attribut zur Verfügung stellen.)

identifier Der Name des Identifiers, der eine ganze PL/SQL-Tabelle darstellt

INDEX BY-Tabellen erstellen (Fortsetzung)

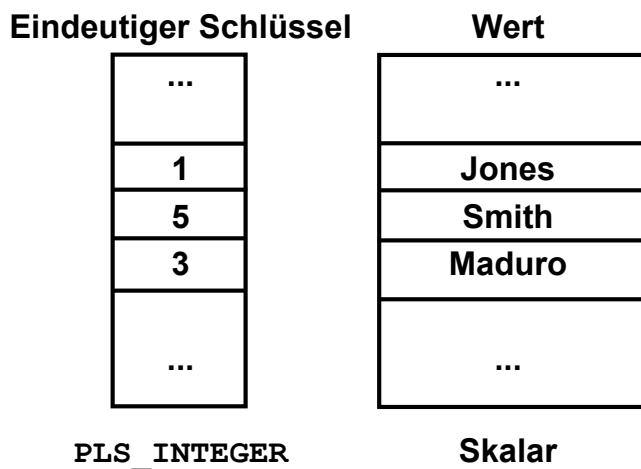
Das NOT NULL-Constraint verhindert, dass der PL/SQL-Tabelle dieses Typs NULL-Werte zugewiesen werden. Initialisieren Sie die INDEX BY-Tabelle nicht.

INDEX BY-Tabellen können Elemente eines beliebigen skalaren Typs aufnehmen.

INDEX BY-Tabellen werden bei der Erstellung nicht automatisch gefüllt. Sie müssen INDEX BY-Tabellen programmatisch in PL/SQL-Programmen füllen.

Oracle Internal & Oracle Academy
Use Only

Struktur von INDEX BY-Tabellen



ORACLE

Copyright © 2007, Oracle. Alle Rechte vorbehalten.

Struktur von INDEX BY-Tabellen

INDEX BY-Tabellen sind wie Datenbanktabellen hinsichtlich der Größe uneingeschränkt. Die Anzahl der Zeilen in einer **INDEX BY**-Tabelle kann sich dynamisch erhöhen, so dass die **INDEX BY**-Tabelle durch Hinzufügen neuer Zeilen wächst.

INDEX BY-Tabellen können eine Spalte und einen eindeutigen Identifier für diese Spalte haben. Sie können jedoch weder die Spalte noch den Identifier benennen. Die Spalte kann einem beliebigen skalaren oder Record-Datentyp angehören. Der Primärschlüssel ist eine Zahl oder Zeichenfolge. Sie können INDEX BY-Tabellen nicht bei ihrer Deklaration initialisieren. INDEX BY-Tabellen werden bei der Deklaration nicht gefüllt. Sie enthalten keine Schlüssel oder Werte. Um die INDEX BY-Tabelle zu füllen, müssen Sie eine explizite ausführbare Anweisung angeben.

INDEX BY-Tabellen erstellen

```
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;
/
```

ORACLE

Copyright © 2007, Oracle. Alle Rechte vorbehalten.

INDEX BY-Tabellen erstellen

Im Beispiel auf der Folie werden zwei INDEX BY-Tabellen erstellt.

Mit dem Schlüssel der INDEX BY-Tabelle können Sie auf ein Element in der Tabelle zugreifen.

Syntax:

```
INDEX_BY_table_name(index)
```

Hier gehört index dem PLS_INTEGER-Typ an.

Das folgende Beispiel zeigt, wie Sie die dritte Zeile einer INDEX BY-Tabelle namens ename_table referenzieren:

```
ename_table(3)
```

Die Größe von PLS_INTEGER liegt zwischen -2.147.483.647 und 2.147.483.647, d. h. der Primärschlüsselwert kann negativ sein. Die Indizierung muss nicht mit 1 beginnen.

Hinweis: Die exists (i)-Methode gibt TRUE zurück, wenn eine Zeile mit dem Index *i* zurückgegeben wird. Verhindern Sie mit der exists-Methode, dass auf Grund eines nicht vorhandenen Tabellenelements eine Fehlermeldung ausgelöst wird.

INDEX BY-Tabellenmethoden

Die folgenden Methoden vereinfachen die Verwendung von INDEX BY-Tabellen:

- **EXISTS**
- **COUNT**
- **FIRST**
- **LAST**
- **PRIOR**
- **NEXT**
- **DELETE**

ORACLE

Copyright © 2007, Oracle. Alle Rechte vorbehalten.

INDEX BY-Tabellenmethoden

INDEX BY-Tabellenmethoden sind Built In-Prozeduren oder -Funktionen, die für PL/SQL-Tabellen ausgeführt und mit der Punktschreibweise aufgerufen werden.

Syntax: *table_name.method_name[(parameters)]*

Methode	Beschreibung
EXISTS (<i>n</i>)	Gibt TRUE zurück, wenn das Element mit dem Index <i>n</i> in einer PL/SQL-Tabelle vorhanden ist
COUNT	Gibt die aktuelle Anzahl von Elementen in einer PL/SQL-Tabelle zurück
FIRST	<ul style="list-style-type: none">• Gibt die erste (kleinste) Index-Zahl in einer PL/SQL-Tabelle zurück• Gibt NULL zurück, wenn die PL/SQL-Tabelle leer ist
LAST	<ul style="list-style-type: none">• Gibt die letzte (größte) Index-Zahl in einer PL/SQL-Tabelle zurück• Gibt NULL zurück, wenn die PL/SQL-Tabelle leer ist
PRIOR (<i>n</i>)	Gibt die Index-Zahl zurück, die dem Index <i>n</i> in einer PL/SQL-Tabelle vorangeht
NEXT (<i>n</i>)	Gibt die Index-Zahl zurück, die dem Index <i>n</i> in einer PL/SQL-Tabelle folgt
DELETE	<ul style="list-style-type: none">• DELETE löscht alle Elemente aus einer PL/SQL-Tabelle.• DELETE (<i>n</i>) löscht das Element mit dem Index <i>n</i> aus einer PL/SQL-Tabelle.• DELETE (<i>m, n</i>) löscht alle Elemente aus einer PL/SQL-Tabelle, deren Index zwischen <i>m</i> und <i>n</i> liegt.

INDEX BY-Tabelle mit Records

Um eine ganze Zeile aus einer Tabelle aufzunehmen, definieren Sie eine INDEX BY-Tabellenvariable.

Beispiel:

```
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
```

ORACLE

Copyright © 2007, Oracle. Alle Rechte vorbehalten.

INDEX BY-Tabelle mit Records

Eine mit einem skalaren Datentyp deklarierte INDEX BY-Tabelle kann jeweils nur die Details einer Spalte in einer Datenbanktabelle speichern. Häufig müssen Sie jedoch alle durch eine Abfrage abgerufenen Spalten speichern. Die Lösung ist eine INDEX BY-Tabelle mit Records. Da nur eine Tabellendefinition erforderlich ist, um Informationen über alle Felder einer Datenbanktabelle aufzunehmen, erhöht die Record-Tabelle die Funktionalität von INDEX BY-Tabellen deutlich.

Tabellen mit Records referenzieren

Im Beispiel auf der Folie können Sie Felder im dept_table-Record referenzieren, da jedes Tabellenelement ein Record ist.

Syntax:

```
table(index).field
```

Beispiel:

```
dept_table(IT).location_id := 1400;  
location_id steht für ein Feld in dept_table.
```

INDEX BY-Tabelle mit Records (Fortsetzung)

Tabellen mit Records referenzieren (Fortsetzung)

Mit dem %ROWTYPE-Attribut können Sie einen Record deklarieren, der eine Zeile in einer Datenbanktabelle repräsentiert. Das %ROWTYPE-Attribut und der PL/SQL-Record mit zusammengesetztem Datentyp unterscheiden sich unter anderem in folgenden Punkten:

- PL/SQL-Record-Typen können benutzerdefiniert sein, während %ROWTYPE den Record implizit definiert.
- Bei PL/SQL-Records können Sie während der Deklaration die Felder und deren Datentypen angeben. Mit %ROWTYPE können Sie die Felder nicht angeben. Das %ROWTYPE-Attribut stellt auf der Basis der Tabellendefinition eine Tabellenzeile mit allen Feldern dar.
- Benutzerdefinierte Records sind statisch. %ROWTYPE-Records sind dynamisch, da sie auf einer Tabellenstruktur basieren. Wenn sich die Tabellenstruktur ändert, ändert sich auch die Record-Struktur.

INDEX BY-Tabelle mit Records – 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

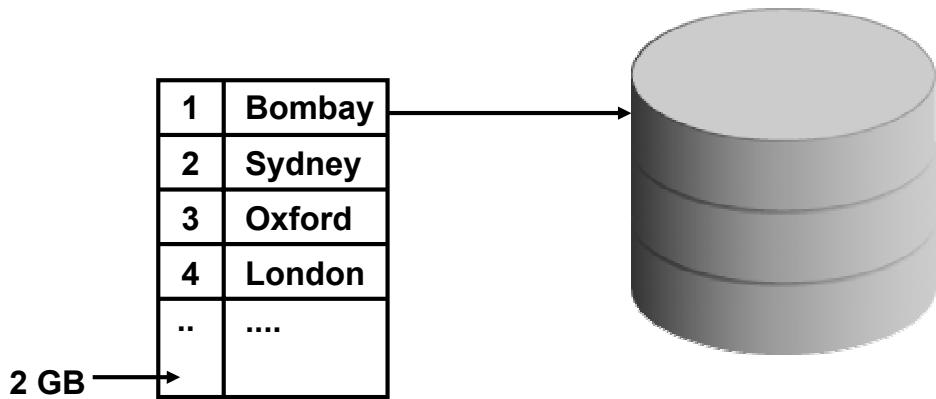
Copyright © 2007, Oracle. Alle Rechte vorbehalten.

INDEX BY-Tabelle mit Records – Beispiel

Im Beispiel auf der Folie wird eine INDEX BY-Tabelle mit Records des Typs `emp_table_type` deklariert, um die Details von Mitarbeitern zwischenzuspeichern, deren Personalnummer zwischen 100 und 104 liegt. Rufen Sie die Informationen der Mitarbeiter mit Hilfe einer Schleife aus der EMPLOYEES-Tabelle ab, und speichern Sie sie in der INDEX BY-Tabelle. Geben Sie mit einer weiteren Schleife die Nachnamen aus der INDEX BY-Tabelle aus. Beachten Sie, wie die Methoden `first` und `last` im Beispiel verwendet werden.

Hinweis: Die Folie zeigt eine Möglichkeit, wie Sie mit einer INDEX BY-Tabelle mit Records arbeiten können. Mit Cursorn können Sie dies jedoch auf effizientere Weise tun. Cursor werden im weiteren Verlauf der Lektion "Explizite Cursor" erläutert.

Nested Tables



ORACLE®

Copyright © 2007, Oracle. Alle Rechte vorbehalten.

Nested Tables

Nested Tables haben eine ähnliche Funktionalität wie INDEX BY-Tabellen, werden jedoch anders implementiert. Nested Tables sind ein gültiger Datentyp in Schematabellen, INDEX BY-Tabellen jedoch nicht. Der Schlüssel kann kein negativer Wert sein (was in der INDEX BY-Tabelle möglich ist). Nested Tables enthalten keinen Schlüssel, auch wenn wir die erste Spalte als Schlüssel bezeichnen. Sie enthalten eine Spalte mit geordneten Zahlen, die als die Schlüsselspalte betrachtet wird. 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 einer Nested Table abrufen, werden den Zeilen der Reihe nach untergeordnete Skripts zugewiesen, die mit 1 beginnen. Nested Tables können (im Gegensatz zu INDEX BY-Tabellen) in der Datenbank gespeichert werden.

Syntax

```
TYPE type_name IS TABLE OF
  {column_type | variable%TYPE
  | table.column%TYPE} [NOT NULL]
  | table.%ROWTYPE
```

Seit Oracle Database 10g können Sie Nested Tables vergleichen. Sie können prüfen, ob ein Element in einer Nested Table vorhanden ist, und ob eine Nested Table eine Teilmenge einer anderen Nested Table ist.

Nested Tables (Fortsetzung)

Beispiel:

```
TYPE location_type IS TABLE OF locations.city%TYPE;
offices location_type;
```

Wenn Sie Nested Tables nicht initialisieren, werden sie automatisch auf NULL initialisiert.

Initialisieren Sie die offices-Nested Table mit Hilfe eines Konstruktors:

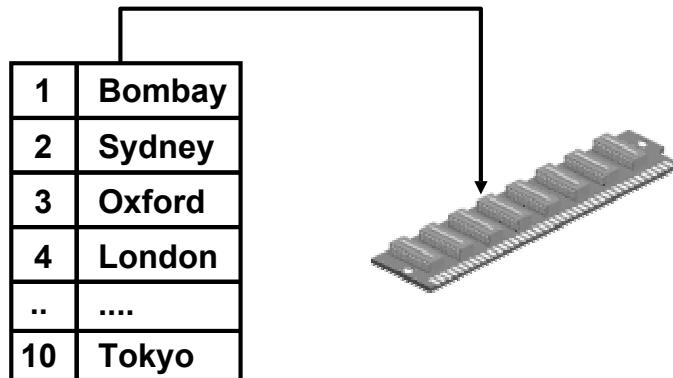
```
offices := location_type('Bombay', 'Tokyo', 'Singapore',
'Oxford');
```

Vollständiges Beispiel:

```
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;
/
```

```
anonymous block completed
Bombay
Tokyo
Singapore
Oxford
```

VARRAY



ORACLE®

Copyright © 2007, Oracle. Alle Rechte vorbehalten.

VARRAY

Arrays mit variabler Größe (VARRAYS) ähneln PL/SQL-Tabellen, sind jedoch in der Größe eingeschränkt. VARRAYS sind in Tabellen auf Schemaebene gültig. Elemente des VARRAY-Typs werden als VARRAYS bezeichnet. VARRAYS haben eine feste Obergrenze. Sie müssen wie bei Arrays in der C-Sprache die Obergrenze angeben, wenn Sie sie deklarieren. VARRAYS können wie Nested Tables maximal 2 GB groß sein. Nested Tables und VARRAYS unterscheiden sich im physischen Speichermodus. Die Elemente eines VARRAYS werden fortlaufend im Speicher und nicht in der Datenbank gespeichert. Sie können einen VARRAY-Typ in der Datenbank mit Hilfe von SQL erstellen.

Beispiel:

```
TYPE location_type IS VARRAY(3) OF locations.city%TYPE;  
offices location_type;
```

Die Größe dieses VARRAYS ist auf 3 eingeschränkt. Sie können das VARRAY mit Hilfe von Konstruktoren initialisieren. Wenn Sie versuchen, das VARRAY mit mehr als drei Elementen zu initialisieren, wird die Fehlermeldung "Subscript outside of limit" angezeigt.

Zusammenfassung

In dieser Lektion haben Sie Folgendes gelernt:

- **PL/SQL-Variablen mit zusammengesetzten Datentypen definieren und referenzieren**
 - PL/SQL-Records
 - INDEX BY-Tabellen
 - INDEX BY-Tabellen mit Records
- **PL/SQL-Records mit dem %ROWTYPE-Attribut definieren**

ORACLE

Copyright © 2007, Oracle. Alle Rechte vorbehalten.

Zusammenfassung

Ein PL/SQL-Record ist eine Collection individueller Felder, die eine Zeile in einer Tabelle darstellen. Mit Hilfe 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. Der Code wird leicht verwaltbar und lesbar.

Die Tabelle ist wie PL/SQL-Records ein zusammengesetzter Datentyp. INDEX BY-Tabellen sind Objekte des TABLE-Typs und sehen ähnlich aus wie Datenbanktabellen. Mit einem kleinen Unterschied. In INDEX BY-Tabellen greifen Sie auf Zeilen wie bei Arrays mit einem Primärschlüssel zu. INDEX BY-Tabellen sind in der Größe uneingeschränkt. Sie speichern einen Schlüssel und ein Wertepaar. Als Datentyp für die Schlüsselspalte sind Ganzzahl oder Zeichenfolge möglich. Der Datentyp der Spalte, in die der Wert aufgenommen wird, kann beliebig sein.

Der Schlüssel für die Nested Tables kann kein negativer Wert sein (was in INDEX BY-Tabellen möglich ist). Die Schlüssel sind außerdem der Reihe nach angeordnet.

Arrays mit variabler Größe (VARARRAYS) ähneln PL/SQL-Tabellen, sind jedoch in der Größe eingeschränkt.

Übungen zu Lektion 6 – Überblick

Diese Übungen behandeln folgende Themen:

- **INDEX BY-Tabellen deklarieren**
- **Daten mit INDEXBY-Tabellen verarbeiten**
- **PL/SQL-Records deklarieren**
- **Daten mit PL/SQL-Records verarbeiten**

ORACLE®

Copyright © 2007, Oracle. Alle Rechte vorbehalten.

Übungen zu Lektion 6 – Überblick

In dieser Übung definieren, erstellen und verwenden Sie INDEX BY-Tabellen und einen PL/SQL-Record.

Übungen zu Lektion 6

1. Erstellen Sie einen PL/SQL-Block, der Informationen über ein bestimmtes Land ausgibt.
 - a. Deklarieren Sie einen PL/SQL-Record basierend auf der Struktur der countries-Tabelle.
 - b. Deklarieren Sie eine v_countryid-Variable. Ordnen Sie CA zu v_countryid zu.
 - c. Deklarieren Sie im deklarativen Bereich mit dem %ROWTYPE-Attribut die Variable v_country_record des countries-Typs.
 - d. Rufen Sie im ausführbaren Bereich mit countryid alle Informationen aus der countries-Tabelle ab. Zeigen Sie die ausgewählten Informationen zum Land an. Die Ausgabe des Beispiels:

```
anonymous block completed
Country Id: CA Country Name: Canada Region: 2
```

- e. Testen Sie ggf. den PL/SQL-Block für die Länder mit den IDs DE, UK und US.
2. Um den Namen einiger Abteilungen aus der departments-Tabelle abzurufen, erstellen Sie einen PL/SQL-Block. Geben Sie die jeweiligen Abteilungsnamen auf dem Bildschirm aus, indem Sie eine INDEX BY-Tabelle aufnehmen. Speichern Sie das Skript unter dem Namen lab_06_02_soln.sql.
 - a. Deklarieren Sie eine INDEX BY-Tabelle dept_table_type des Typs departments.department_name. Um den Namen der Abteilungen temporär zu speichern, deklarieren Sie eine Variable my_dept_table des Typs dept_table_type.
 - b. Deklarieren Sie zwei Variablen: loop_count und deptno des NUMBER-Typs. Ordnen Sie loop_count die Zahl 10 und deptno die Zahl 0 zu.
 - c. Rufen Sie mit Hilfe einer Schleife die Namen von 10 Abteilungen ab, und speichern Sie sie in der INDEX BY-Tabelle. Beginnen Sie mit der department_id 10. Erhöhen Sie deptno bei jeder Iteration der Schleife um 10. Die folgende Tabelle zeigt die department_id, für die Sie department_name abrufen und in der INDEX BY-Tabelle speichern sollen.

DEPARTMENT_ID	DEPARTMENT_NAME
10	Administration
20	Marketing
30	Purchasing
40	Human Resources
50	Shipping
60	IT
70	Public Relations
80	Sales
90	Executive
100	Finance

Übungen zu Lektion 6 (Fortsetzung)

- d. Rufen Sie mit einer weiteren Schleife die Abteilungsnamen aus der INDEX BY-Tabelle ab, und zeigen Sie sie an.
- e. Führen Sie Ihr Skript aus, und speichern Sie es unter dem Namen lab_06_02_soln.sql. Die Ausgabe des Beispiels:

```
anonymous block completed
Administration
Marketing
Purchasing
Human Resources
Shipping
IT
Public Relations
Sales
Executive
Finance
```

Übungen zu Lektion 6 (Fortsetzung)

3. Bearbeiten Sie den in der 2. Übung erstellten Block so, dass alle Informationen über die einzelnen Abteilungen aus der departments-Tabelle abgerufen und angezeigt werden. Verwenden Sie eine INDEX BY-Tabelle mit Records.
- a. Laden Sie das Skript lab_06_02_soln.sql.
 - b. Sie haben als Typ der INDEX BY-Tabelle departments.department_name deklariert. Ändern Sie die Deklaration der INDEX BY-Tabelle, so dass die Nummer, der Name und der Standort aller Abteilungen temporär gespeichert werden. Verwenden Sie das %ROWTYPE-Attribut.
 - c. Um alle derzeit in der departments-Tabelle gespeicherten Abteilungsinformationen abzurufen, bearbeiten Sie die SELECT-Anweisung. Speichern Sie sie in der INDEX BY-Tabelle.
 - d. Rufen Sie mit einer weiteren Schleife die Abteilungsinformationen aus der INDEX BY-Tabelle ab, und zeigen Sie sie an. Die Ausgabe des Beispiels:

```
anonymous block completed  
Department Number: 10 Department Name: Administration Manager Id: 200  
Location Id: 1700  
Department Number: 20 Department Name: Marketing Manager Id: 201  
Location Id: 1800  
Department Number: 30 Department Name: Purchasing Manager Id: 114  
Location Id: 1700  
Department Number: 40 Department Name: Human Resources Manager Id: 203  
Location Id: 2400  
Department Number: 50 Department Name: Shipping Manager Id: 121 Location  
Id: 1500  
Department Number: 60 Department Name: IT Manager Id: 103 Location  
Id: 1400  
Department Number: 70 Department Name: Public Relations Manager Id: 204  
Location Id: 2700  
Department Number: 80 Department Name: Sales Manager Id: 145 Location  
Id: 2500  
Department Number: 90 Department Name: Executive Manager Id: 100  
Location Id: 1700  
Department Number: 100 Department Name: Finance Manager Id: 108 Location  
Id: 1700
```

Explizite Cursor

ORACLE®

Copyright © 2007, Oracle. Alle Rechte vorbehalten.

Oracle Internal & Oracle Academy
Use Only

Ziele

Nach Ablauf dieser Lektion haben Sie folgende Ziele erreicht:

- Zwischen impliziten und expliziten Cursorn unterscheiden
- Gründe für die Verwendung expliziter Cursor kennen
- Explizite Cursor deklarieren und kontrollieren
- Daten mit einfachen und Cursor FOR-Schleifen lesen
- Cursor mit Parametern deklarieren und verwenden
- Zeilen mit der FOR UPDATE-Klausel sperren
- Aktuelle Zeile mit der Klausel WHERE CURRENT OF referenzieren

ORACLE®

Copyright © 2007, Oracle. Alle Rechte vorbehalten.

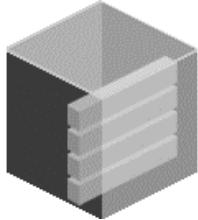
Lernziel

Sie haben implizite Cursor kennen gelernt, 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.

Cursor

Jeder vom Oracle-Server ausgeführten SQL-Anweisung ist ein eigener Cursor zugeordnet:

- **Implizite Cursor:** Von PL/SQL für alle DML-Anweisungen und die PL/SQL-Anweisung `SELECT` deklariert und verwaltet
- **Explizite Cursor:** Vom Programmierer deklariert und verwaltet



ORACLE

Copyright © 2007, Oracle. Alle Rechte vorbehalten.

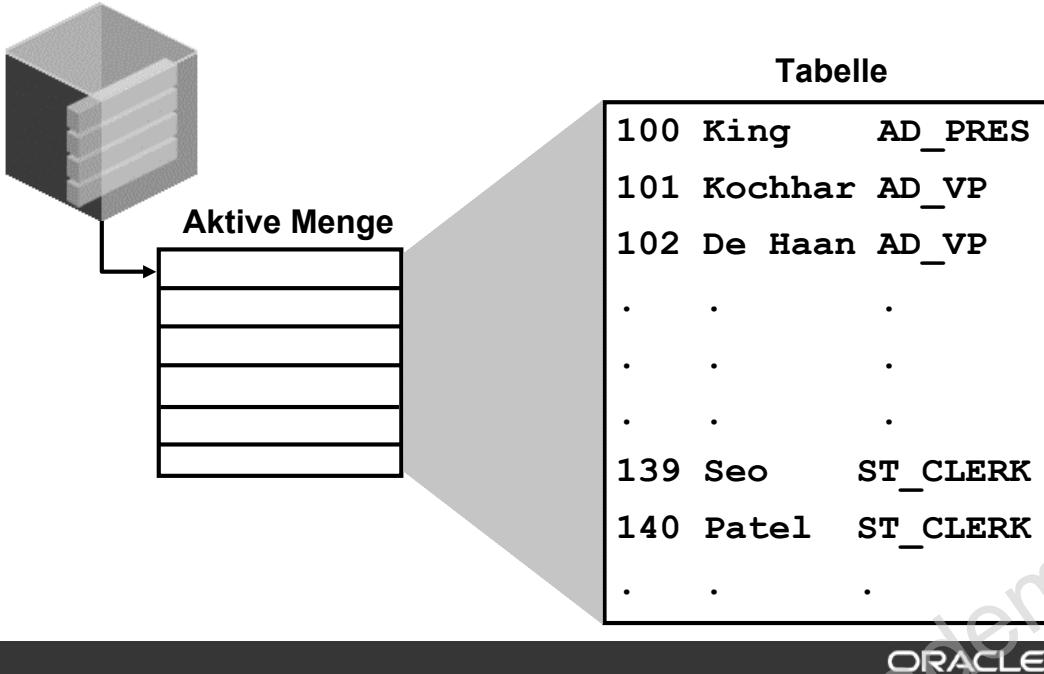
Cursor

Um SQL-Anweisungen auszuführen und Verarbeitungsinformationen zu speichern, verwendet der Oracle-Server Arbeitsbereiche (als *private SQL-Bereiche* bezeichnet). Mit expliziten Cursoren können Sie private SQL-Bereiche benennen und auf die darin gespeicherten Informationen zugreifen.

Cursor-Typ	Beschreibung
Implizit	Implizite Cursor werden von PL/SQL implizit für alle DML-Anweisungen und die PL/SQL-Anweisung <code>SELECT</code> deklariert.
Explizit	Für Abfragen, die mehr als eine Zeile zurückgeben, werden vom Programmierer explizite Cursor deklariert, verwaltet und über spezielle Anweisungen im ausführbaren Teil des Blockes bearbeitet.

Um jede SQL-Anweisung zu verarbeiten, der kein explizit deklarierter Cursor zugeordnet ist, öffnet der Oracle-Server Cursor implizit. Mit PL/SQL können Sie den zuletzt verwendeten impliziten Cursor als SQL-Cursor referenzieren.

Explizite Cursor-Operationen



Copyright © 2007, Oracle. Alle Rechte vorbehalten.

Explizite Cursor-Operationen

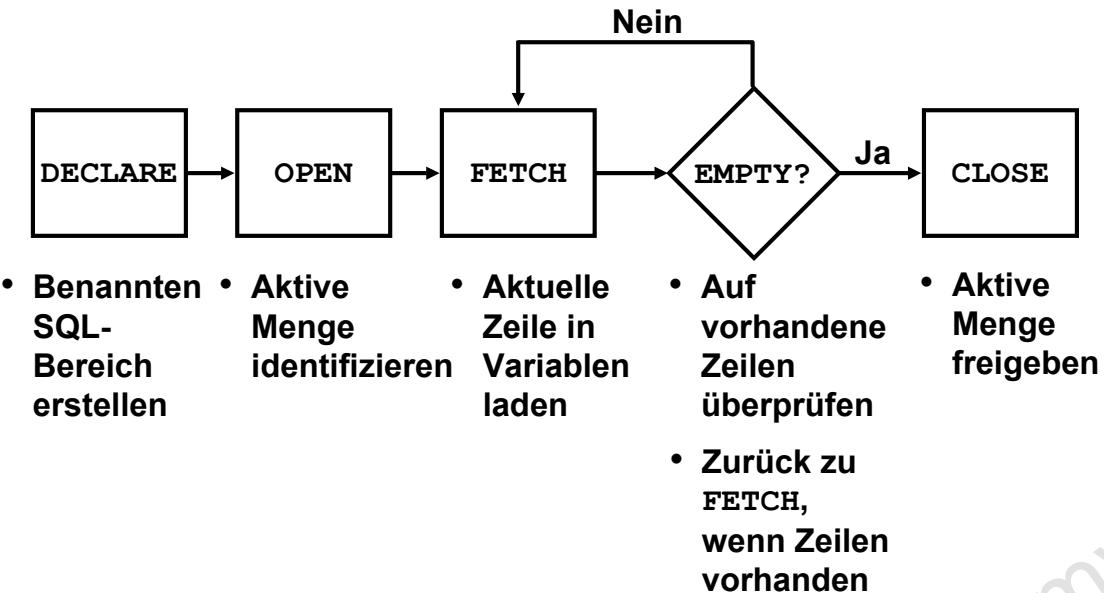
Sie deklarieren explizite Cursor in PL/SQL, wenn eine SELECT-Anweisung mehrere Zeilen zurückgibt. Sie können jede von der SELECT-Anweisung zurückgegebene Zeile verarbeiten.

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 aktiven Menge "zeigt". Das Programm kann auf diese Weise die Zeilen der Reihe nach verarbeiten.

Funktionen expliziter Cursor:

- Zeilenweise Verarbeitung über die erste von der Abfrage zurückgegebene Zeile hinaus ermöglichen
- Derzeit verarbeitete Zeile überwachen
- Programmierern die manuelle Kontrolle expliziter Cursor im PL/SQL-Block ermöglichen

Explizite Cursor kontrollieren



ORACLE

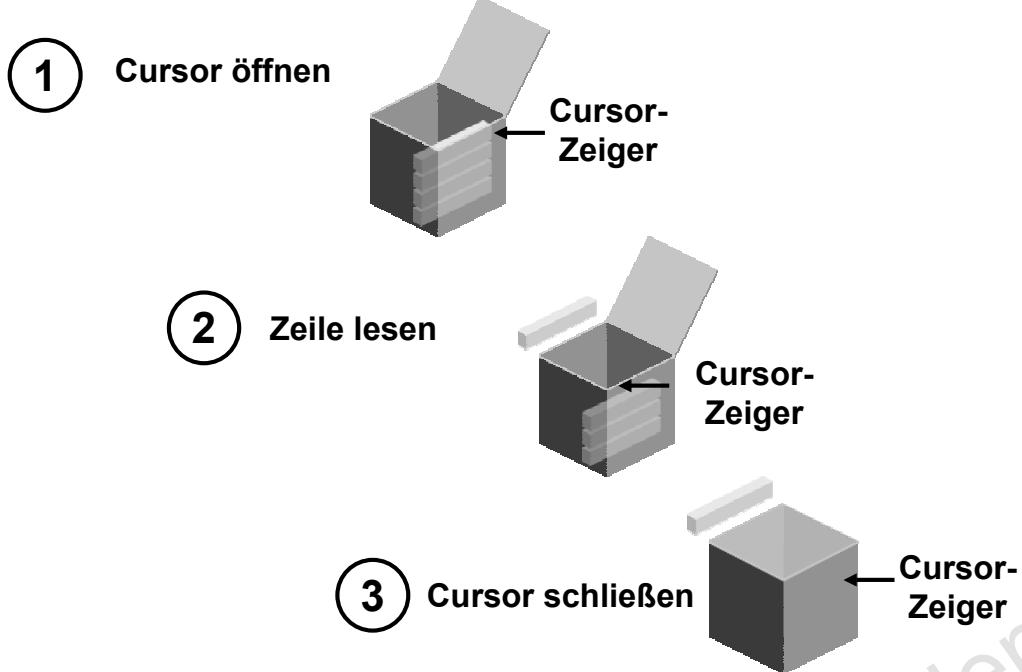
Copyright © 2007, Oracle. Alle Rechte vorbehalten.

Explizite Cursor kontrollieren

Sie kennen nun die Grundlagen von Cursorn. Verwenden Sie sie wie folgt:

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 und bindet alle referenzierten Variablen. Die von der Abfrage ermittelten Zeilen werden als *aktive Menge* bezeichnet. Sie stehen jetzt zum Lesen zur Verfügung.
3. Lesen Sie Daten aus dem Cursor.
Im Ablaufdiagramm auf der Folie wird nach jeder gelesenen Zeile überprüft, ob der Cursor weitere Zeilen enthält. Wenn alle verarbeitbaren Zeilen eingelesen wurden, müssen Sie den Cursor schließen.
4. Schließen Sie den Cursor.
Die CLOSE-Anweisung gibt die aktive Zeilenmenge frei. Um eine neue aktive Menge zu erstellen, können Sie den Cursor jetzt erneut öffnen.

Explizite Cursor kontrollieren



ORACLE

Copyright © 2007, Oracle. Alle Rechte vorbehalten.

Explizite Cursor kontrollieren (Fortsetzung)

PL/SQL öffnet einen Cursor, verarbeitet die abgerufenen Zeilen und schließt den Cursor wieder. Der Cursor markiert die aktuelle Position in der aktiven Menge.

1. Die `OPEN`-Anweisung führt die dem Cursor zugeordnete Abfrage aus, identifiziert die aktive Menge und positioniert den Cursor an die erste Zeile.
2. Die `FETCH`-Anweisung ruft die aktuelle Zeile ab und bewegt den Cursor in die nächste Zeile, bis keine weitere Zeile mehr vorhanden ist oder eine bestimmte Bedingung erfüllt wurde.
3. Die `CLOSE`-Anweisung gibt den Cursor frei.

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 © 2007, Oracle. Alle Rechte vorbehalten.

Cursor deklarieren

Die Syntax zum Deklarieren von Cursorn ist auf der Folie dargestellt. Für die Syntax gilt:

cursor_name Ein PL/SQL-Identifier
select_statement Eine SELECT-Anweisung ohne INTO-Klausel

Die aktive Menge eines Cursors wird durch die SELECT-Anweisung in der Cursor-Deklaration bestimmt. In PL/SQL muss eine SELECT-Anweisung eine INTO-Klausel enthalten. Die SELECT-Anweisung in der Cursor-Deklaration kann jedoch keine INTO-Klausel enthalten. Der Grund dafür ist, dass Sie im deklarativen Bereich nur einen Cursor definieren, aber keine Zeilen in den Cursor abrufen.

Hinweis

- Nehmen Sie die INTO-Klausel nicht in die Cursor-Deklaration auf, da sie später in der FETCH-Anweisung enthalten ist.
- Wenn Sie Zeilen in einer bestimmten Reihenfolge verarbeiten möchten, geben Sie in der Abfrage die ORDER BY-Klausel an.
- Der Cursor kann eine beliebige gültige SELECT-Anweisung sein, z. B. ein Join oder eine Unterabfrage.

Cursor deklarieren (Fortsetzung)

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`.

Um alle Details zur Abteilung mit der `location_id` 1700 abzurufen, deklarieren Sie den Cursor `c_dept_cursor`. Beachten Sie, dass eine Variable verwendet wird, während Sie den Cursor deklarieren. Diese Variablen werden als Bind-Variablen bezeichnet. Sie müssen sichtbar sein, wenn Sie den Cursor deklarieren. Die Variablen werden nur einmal überprüft, während der Cursor geöffnet wird. Sie haben gelernt, dass Sie in PL/SQL mehrere Zeilen mit expliziten Cursorn abrufen und bearbeiten. Dieses Beispiel zeigt jedoch, dass Sie explizite Cursor auch verwenden 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;
```

ORACLE

Copyright © 2007, Oracle. Alle Rechte vorbehalten.

Cursor öffnen

Die OPEN-Anweisung führt die dem Cursor zugeordnete Abfrage aus, identifiziert die aktive Menge und positioniert den Cursor-Zeiger an die erste Zeile. Nehmen Sie die OPEN-Anweisung in den ausführbaren Bereich des PL/SQL-Blockes auf.

OPEN ist eine ausführbare Anweisung, die folgende Operationen ausführt:

1. Kontextbereichen dynamisch Speicher zuweisen
2. SELECT-Anweisungen analysieren
3. Eingabevervariablen binden (Die Anweisung legt die Werte für die Eingabevervariablen fest, indem sie ihre Speicheradressen abruft.)
4. Aktive Menge identifizieren (die den Suchkriterien entsprechende Zeilenmenge). Wenn Sie die OPEN-Anweisung ausführen, werden keine Zeilen in der aktiven Menge in Variablen abgerufen. Stattdessen ruft die FETCH-Anweisung die Zeilen aus dem Cursor in die Variablen ab.
5. Zeiger in der ersten Zeile der aktiven Menge positionieren

Hinweis: Wenn die Abfrage bei geöffnetem Cursor keine Zeilen zurückgibt, löst PL/SQL keine Exception aus. Mit dem Attribut <cursor_name>%ROWCOUNT können Sie die Anzahl der mit einem expliziten Cursor zurückgegebenen Zeilen ermitteln.

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 ;
/
```

ORACLE

Copyright © 2007, Oracle. Alle Rechte vorbehalten.

Daten aus Cursorn lesen

Die `FETCH`-Anweisung ruft die Zeilen nacheinander aus dem Cursor ab. Nachdem der Cursor eine Zeile gelesen hat, geht er zur nächsten Zeile der aktiven Menge. Mit dem `%NOTFOUND`-Attribut können Sie bestimmen, ob die aktive Menge vollständig abgerufen wurde.

Sehen Sie sich das Beispiel auf der Folie an. Um die gelesenen Werte aus dem Cursor aufzunehmen, deklarieren Sie die beiden Variablen `empno` und `lname`. Betrachten Sie die `FETCH`-Anweisung.

Die Ausgabe des PL/SQL-Blockes lautet:

```
anonymous block completed
114  Raphaely
```

Sie haben die Werte erfolgreich aus dem Cursor in die Variablen gelesen. In Abteilung 30 arbeiten sechs Mitarbeiter, aber es wurde nur eine Zeile gelesen. Um alle Zeilen zu lesen, müssen Sie Schleifen verwenden. Auf der nächsten Folie sehen Sie, wie Sie mit einer Schleife alle Zeilen lesen.

Die `FETCH`-Anweisung führt die folgenden Operationen aus:

1. Daten der aktuellen Zeile in die PL/SQL-Ausgabevariablen lesen
2. Zeiger in der aktiven Menge zur nächsten Zeile bewegen

Daten aus Cursorn lesen (Fortsetzung)

Sie können in die INTO-Klausel der FETCH-Anweisung genauso viele Variablen aufnehmen, wie Spalten in der SELECT-Anweisung angegeben sind. Stellen Sie sicher, dass die Datentypen kompatibel sind. Passen Sie die einzelnen Variablen an die Position der Spalten an. Sie können auch einen Record für den Cursor definieren und in der FETCH INTO-Klausel referenzieren. Prüfen Sie schließlich, ob der Cursor Zeilen enthält. Wenn eine FETCH-Anweisung keine Werte bereitstellt, sind in der aktiven Menge keine weiteren zu verarbeitenden Zeilen vorhanden und es wird kein Fehler aufgezeichnet.

Oracle Internal & Oracle Academy
Use Only

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 © 2007, Oracle. Alle Rechte vorbehalten.

Daten aus Cursorn lesen (Fortsetzung)

Beachten Sie, dass die Zeilen mit einer einfachen Schleife gelesen werden. Das %NOTFOUND-Cursor-Attribut 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 © 2007, Oracle. Alle Rechte vorbehalten.

Cursor schließen

Die CLOSE-Anweisung deaktiviert den Cursor, gibt den Kontextbereich frei und löscht die Prozedurvariable der aktiven Menge. Schließen Sie den Cursor, wenn die FETCH-Anweisung vollständig verarbeitet ist. Sie können den Cursor erneut öffnen, falls erforderlich. Dies ist jedoch nur möglich, wenn Sie den Cursor geschlossen haben. Wenn Sie versuchen, Daten aus einem Cursor zu lesen, nachdem er geschlossen wurde, wird die Exception INVALID_CURSOR ausgelöst.

Hinweis: Es ist zwar möglich, den PL/SQL-Block zu beenden, ohne den Cursor zu schließen, Sie sollten deklarierte Cursor jedoch stets explizit schließen, um Ressourcen freizugeben.

Die maximale Anzahl der geöffneten Cursor pro Session ist begrenzt. Sie wird in der Datenbankparameterdatei durch den OPEN_CURSORS-Parameter bestimmt. (Der Default-Wert ist OPEN_CURSORS = 50.)

Cursor und Records

Zeilen der aktiven Menge verarbeiten, indem Sie Werte in einen PL/SQL-Record einlesen

```
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 © 2007, Oracle. Alle Rechte vorbehalten.

Cursor und Records

Sie haben bereits gelernt, dass Sie Records definieren können, die in Tabellen in Spaltenform vorhanden sind. Sie können Records auch auf der ausgewählten Spaltenliste basierend in expliziten Cursorn definieren. Dies ist nützlich, wenn Sie die Zeilen der aktiven Menge verarbeiten, da Sie einfach in den Record lesen können. Die Werte der Zeile werden somit direkt in die entsprechenden Felder des Records geladen.

Cursor FOR-Schleifen

Syntax:

```
FOR record_name IN cursor_name LOOP
    statement1;
    statement2;
    . . .
END LOOP;
```

- Mit Cursor FOR-Schleifen können Sie die Verarbeitung expliziter Cursor verkürzen.
- OPEN-, FETCH-, CLOSE- und EXIT-Vorgänge werden implizit durchgeführt.
- Der Record wird implizit deklariert.

ORACLE

Copyright © 2007, Oracle. Alle Rechte vorbehalten.

Cursor FOR-Schleifen

Sie haben gelernt, wie Sie Daten mit Hilfe 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
cursor_name

Der Name des implizit deklarierten Records
Ein PL/SQL-Identifier für den zuvor deklarierten Cursor

Richtlinien

- Deklarieren Sie nicht den Record, der die Schleife kontrolliert. Er wird implizit deklariert.
- Testen Sie die Cursor-Attribute, falls erforderlich, im Verlauf der Schleife.
- Stellen Sie die Parameter für einen Cursor in der FOR-Anweisung nach dem Cursor-Namen in Klammern zur Verfügung, falls erforderlich.

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 ;
/
```

ORACLE

Copyright © 2007, Oracle. Alle Rechte vorbehalten.

Cursor FOR-Schleifen (Fortsetzung)

Diese Folie zeigt noch einmal das Beispiel, in dem Daten mit Hilfe einer einfachen Schleife aus Cursors gelesen wurden, diesmal jedoch unter Verwendung der Cursor FOR-Schleife.

In diesem Fall wird der `emp_record`-Record implizit deklariert. Mit diesem impliziten Record können Sie auf die gelesenen Daten zugreifen (wie auf der Folie gezeigt). Beachten Sie, dass keine Variablen deklariert werden, um die gelesenen Daten mit Hilfe der `INTO`-Klausel aufzunehmen. Der Code enthält keine `OPEN`- und `CLOSE`-Anweisungen, um den Cursor zu öffnen bzw. zu schließen.

Explizite Cursor-Attribute

Ermitteln Sie mit expliziten Cursor-Attributen die Statusinformationen eines Cursors.

Attribut	Typ	Beschreibung
%ISOPEN	Boolesch	Ist TRUE, wenn der Cursor geöffnet ist
%NOTFOUND	Boolesch	Ist TRUE, wenn die zuletzt durchgeführte Fetch-Anweisung keine Zeile zurückgibt
%FOUND	Boolesch	Ist TRUE, wenn die zuletzt durchgeführte Fetch-Anweisung eine Zeile zurückgibt. Gegenstück zu %NOTFOUND
%ROWCOUNT	Zahl	Ergibt die Gesamtanzahl der bis zu diesem Zeitpunkt zurückgegebenen Zeilen

ORACLE®

Copyright © 2007, Oracle. Alle Rechte vorbehalten.

Explizite Cursor-Attribute

Wie bei impliziten Cursoren gibt es vier Attribute, mit denen Sie Statusinformationen über einen Cursor ermitteln können. Wenn Sie diese Attribute an den Namen der Cursor-Variablen anhängen, geben sie nützliche Informationen darüber zurück, wie Sie eine Anweisung zum Bearbeiten von Cursorn ausführen.

Hinweis: Sie können Cursor-Attribute nicht direkt in einer SQL-Anweisung referenzieren.

%ISOPEN-Attribut

- Lesen Sie Zeilen nur, wenn der Cursor geöffnet ist.
- Um zu prüfen, ob der Cursor geöffnet ist, verwenden Sie vor einem Zeilenabruf das %ISOPEN-Cursor-Attribut.

Beispiel:

```
IF NOT c_emp_cursor%ISOPEN THEN
    OPEN c_emp_cursor;
END IF;
LOOP
    FETCH c_emp_cursor...
```

ORACLE

Copyright © 2007, Oracle. Alle Rechte vorbehalten.

%ISOPEN-Attribut

- Sie können Zeilen nur lesen, wenn der Cursor geöffnet ist. Um zu ermitteln, ob der Cursor geöffnet ist, verwenden Sie das %ISOPEN-Cursor-Attribut.
- Lesen Sie Zeile in einer Schleife. Bestimmen Sie mit Cursor-Attributen, wann die Schleife beendet werden soll.
- Verwenden Sie das %ROWCOUNT-Cursor-Attribut für folgende Aufgaben:
 - Genaue Anzahl von Zeilen verarbeiten
 - Zeilen in einer Schleife lesen und bestimmen, wann die Schleife beendet werden soll

Hinweis: %ISOPEN gibt den Cursor-Status 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 ;
/
```

ORACLE

Copyright © 2007, Oracle. Alle Rechte vorbehalten.

%ROWCOUNT und %NOTFOUND – Beispiel

Im Beispiel auf der Folie werden nacheinander die ersten 10 Mitarbeiter abgerufen. Dieses Beispiel zeigt, wie Sie die Attribute %ROWCOUNT und %NOTFOUND für EXIT-Bedingungen in einer Schleife verwenden können. Die Ausgabe des PL/SQL-Blockes lautet:

```
anonymous block completed
100 King
101 Kochhar
102 De Haan
103 Hunold
104 Ernst
105 Austin
106 Pataballa
107 Lorentz
108 Greenberg
109 Faviet
```

Cursor FOR-Schleifen mit Unterabfragen

Sie müssen keinen Cursor deklarieren.

Beispiel:

```
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 ;
/
```

ORACLE

Copyright © 2007, Oracle. Alle Rechte vorbehalten.

Cursor FOR-Schleifen mit Unterabfragen

Beachten Sie, dass dieser PL/SQL-Block keinen deklarativen Bereich hat. Der Unterschied zwischen Cursor FOR-Schleifen mit und ohne Unterabfragen liegt in der Cursor-Deklaration. Wenn Sie Cursor FOR-Schleifen mit Unterabfragen erstellen, müssen Sie den Cursor nicht im deklarativen Bereich deklarieren. Sie müssen die SELECT-Anweisung angeben, die die aktive Menge innerhalb der Schleife bestimmt.

Diese Folie zeigt noch einmal das Beispiel der Cursor FOR-Schleife, diesmal jedoch mit Unterabfragen.

Hinweis: Sie können explizite Cursor-Attribute nicht referenzieren, wenn Sie in einer Cursor FOR-Schleife Unterabfragen verwenden, da Sie dem Cursor keinen expliziten Namen geben können.

Cursor mit Parametern

Syntax:

```
CURSOR cursor_name
  [ (parameter_name datatype, ... ) ]
IS
  select_statement;
```

- Übergeben Sie Parameterwerte an einen Cursor, sobald der Cursor geöffnet und die Abfrage ausgeführt wird.
- Öffnen Sie einen expliziten Cursor mehrmals mit jeweils einer anderen aktiven Menge.

```
OPEN cursor_name (parameter_value, . . . . .) ;
```

ORACLE®

Copyright © 2007, Oracle. Alle Rechte vorbehalten.

Cursor mit Parametern

Sie können Parameter 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. Der Cursor wird geschlossen und erneut mit einer neuen Gruppe von Parametern geöffnet.

Jeder formale Parameter in der Cursor-Deklaration muss einen entsprechenden aktuellen 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>	Ein PL/SQL-Identifier für den deklarierten Cursor
<i>parameter_name</i>	Der Name eines Parameters
<i>datatype</i>	Der skalare Datentyp des Parameters
<i>select_statement</i>	Eine SELECT-Anweisung ohne INTO-Klausel

Die Parameterschreibweise 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);
    ...

```

ORACLE

Copyright © 2007, Oracle. Alle Rechte vorbehalten.

Cursor mit Parametern (Fortsetzung)

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 ...
```

FOR UPDATE-Klausel

Syntax:

```
SELECT ...
FROM      ...
FOR UPDATE [OF column_reference] [NOWAIT | WAIT n];
```

- **Verweigern Sie mit expliziten Sperren anderen Sessions während einer Transaktion den Zugriff.**
- **Sperren Sie die Zeilen, bevor Sie sie aktualisieren oder löschen.**

ORACLE®

Copyright © 2007, Oracle. Alle Rechte vorbehalten.

FOR UPDATE-Klausel

Wenn mehrere Sessions auf eine Datenbank zugreifen, kann es vorkommen, dass die Zeilen einer bestimmten Tabelle aktualisiert werden, nachdem Sie den Cursor öffnen. Sie sehen die aktualisierten Daten nur, wenn Sie den Cursor erneut öffnen. Es ist daher ratsam, die Zeilen zu sperren, bevor Sie sie aktualisieren oder löschen. Sie können zu diesem Zweck die FOR UPDATE-Klausel der Cursor-Abfrage verwenden.

Für die Syntax gilt:

<i>column_reference</i>	Eine Spalte in der abgefragten Tabelle (Sie können auch eine Liste von Spalten verwenden.)
NOWAIT	Gibt einen Oracle-Server-Fehler zurück, wenn die Zeilen durch eine andere Session gesperrt sind.

Die FOR UPDATE-Klausel ist die letzte Klausel in einer SELECT-Anweisung, nach ORDER BY (sofern vorhanden). Wenn Sie mehrere Tabellen abfragen, können Sie die Zeilensperrung mit der FOR UPDATE-Klausel auf bestimmte Tabellen beschränken. FOR UPDATE OF *col_name(s)* sperrt Zeilen nur in Tabellen, die *col_name(s)* enthalten.

FOR UPDATE-Klausel (Fortsetzung)

Die Anweisung SELECT . . . FOR UPDATE identifiziert die zu aktualisierenden oder zu löschen Zeilen und sperrt dann jede Zeile in der Ergebnismenge. Dies ist nützlich, wenn Sie eine Aktualisierung auf der 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 NOWAIT-Schlüsselwort 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 NOWAIT-Schlüsselwort 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 die Operation SELECT FOR UPDATE erforderlichen Zeilensperren nicht zuteilen kann, wartet er auf unbegrenzte Zeit. Vermeiden Sie dies mit dem NOWAIT-Schlüsselwort. 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. Wenn Sie WAIT an Stelle von NOWAIT verwenden, 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. Es wird jedoch zur besseren Lesbarkeit und Verwaltung empfohlen.

Klausel WHERE CURRENT OF

Syntax:

```
WHERE CURRENT OF cursor ;
```

- Aktualisieren oder löschen Sie mit Hilfe von Cursorn die aktuelle Zeile.
- Um die Zeilen zuerst zu sperren, nehmen Sie die FOR UPDATE-Klausel in die Cursor-Abfrage auf.
- Referenzieren Sie mit der Klausel WHERE CURRENT OF die aktuelle Zeile eines expliziten Cursors.

```
UPDATE employees  
      SET salary = ...  
 WHERE CURRENT OF c_emp_cursor;
```

ORACLE

Copyright © 2007, Oracle. Alle Rechte vorbehalten.

Klausel WHERE CURRENT OF

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 Anweisungen UPDATE oder DELETE verwendet, während Sie die FOR UPDATE-Klausel in der Cursor-Deklaration angeben. Indem Sie die Klauseln kombinieren, können Sie die aktuelle Zeile in der entsprechenden Datenbanktabelle aktualisieren und löschen. Sie können somit die Anweisungen UPDATE und DELETE auf die aktuelle Zeile anwenden, ohne die ROWID explizit zu referenzieren. Sie müssen die FOR UPDATE-Klausel in die Cursor-Abfrage aufnehmen, damit die Zeilen bei Ausführung der OPEN-Anweisung gesperrt werden.

Für die Syntax gilt:

cursor Der Name eines deklarierten Cursors (Der Cursor muss mit der FOR UPDATE-Klausel deklariert sein.)

Cursor mit Unterabfragen

Beispiel:

```
DECLARE
  CURSOR my_cursor IS
    SELECT t1.department_id, t1.department_name,
           t2.staff
      FROM departments t1, (SELECT department_id,
                                         COUNT(*) AS staff
                                    FROM employees
                                   GROUP BY department_id) t2
     WHERE t1.department_id = t2.department_id
       AND t2.staff >= 3;
  ...

```

ORACLE

Copyright © 2007, Oracle. Alle Rechte vorbehalten.

Cursor mit Unterabfragen

Unterabfragen sind (üblicherweise in Klammern eingeschlossene) Abfragen, die in andere SQL-Anweisungen aufgenommen werden. Wenn Sie Unterabfragen bewerten, wird der äußeren Abfrage ein Wert oder eine Gruppe von Werten bereitgestellt. Unterabfragen werden häufig in der WHERE-Klausel einer SELECT-Anweisung verwendet. Sie können sie auch in der FROM-Klausel verwenden, indem Sie für diese Abfrage eine temporäre Datenquelle erstellen.

Im Beispiel auf der Folie erstellt die Unterabfrage eine Datenquelle, die die Abteilungsnummern und die Anzahl der Mitarbeiter (mit dem Alias STAFF) jeder Abteilung enthält. Der Tabellenalias t2 verweist auf die temporäre Datenquelle in der FROM-Klausel. Wenn Sie diesen Cursor öffnen, enthält die aktive Menge die Abteilungsnummer, den Abteilungsnamen und die Gesamtanzahl der Mitarbeiter in den Abteilungen mit drei oder mehr Mitarbeitern.

Zusammenfassung

In dieser Lektion haben Sie Folgendes gelernt:

- **Cursor-Typen 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**
- **Cursor-Status mit Hilfe von Cursor-Attributen bestimmen**
- **Aktuell gelesene Zeile mit den Klauseln FOR UPDATE und WHERE CURRENT OF aktualisieren oder löschen**

ORACLE®

Copyright © 2007, Oracle. Alle Rechte vorbehalten.

Zusammenfassung

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 Cursor-Variable hat vier Attribute: %FOUND, %ISOPEN, %NOTFOUND und %ROWCOUNT. Wenn Sie die Attribute an den Namen der Cursor-Variablen anhängen, geben sie nützliche Informationen über die Ausführung einer SQL-Anweisung zurück. Sie können Cursor-Attribute in prozeduralen Anweisungen, nicht jedoch in SQL-Anweisungen verwenden.

Bearbeiten Sie die vom Cursor gelesenen Zeilen mit Hilfe einfacher oder Cursor FOR-Schleifen. Wenn Sie einfache Schleifen verwenden, müssen Sie den Cursor öffnen, die Zeilen lesen und den Cursor schließen. Cursor FOR-Schleifen führen diese Vorgänge implizit durch. Sperren, die Sie aktualisieren oder löschen, sollten Sie mit einer FOR UPDATE-Klausel 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 7 – Überblick

Diese Übungen behandeln folgende Themen:

- **Explizite Cursor zum Abfragen von Tabellenzeilen deklarieren und verwenden**
- **Cursor FOR-Schleifen verwenden**
- **Mit Hilfe von Cursor-Attributen den Cursor-Status testen**
- **Cursor mit Parametern deklarieren und verwenden**
- **Die Klauseln FOR UPDATE und WHERE CURRENT OF verwenden**

ORACLE

Copyright © 2007, Oracle. Alle Rechte vorbehalten.

Übungen zu Lektion 7 – Überblick

In diesen Übungen verarbeiten Sie anhand des Gelernten eine Reihe von Zeilen einer Tabelle und füllen eine andere Tabelle mit Hilfe einer Cursor FOR-Schleife mit den Ergebnissen. Außerdem erstellen Sie einen Cursor mit Parametern.

Übungen zu Lektion 7

1. Erstellen Sie einen PL/SQL-Block, der die ersten n Spitzenverdiener ermittelt.
 - a. Führen Sie das Skript `lab_07_01.sql` aus, um eine neue `top_salaries`-Tabelle zu erstellen, in der Sie die Gehälter der Mitarbeiter speichern.
 - b. Deklarieren Sie im deklarativen Bereich eine `v_num`-Variable des NUMBER-Typs, die eine Zahl n enthält. n steht für die ersten n Spitzenverdiener aus der `employees`-Tabelle. Beispiel: Um die fünf höchsten Gehälter anzuzeigen, geben Sie 5 ein. Deklarieren Sie eine weitere `sal`-Variable des `employees.salary`-Typs. Deklarieren Sie den Cursor `c_emp_cursor`, der die Gehälter der Mitarbeiter in absteigender Reihenfolge abruft.
 - c. Öffnen Sie die Schleife im ausführbaren Bereich. Lesen Sie die höchsten n Gehälter, und fügen Sie sie in die `top_salaries`-Tabelle ein. Sie können die Daten mit einer einfachen Schleife bearbeiten. Verwenden Sie für die `exit`-Bedingung versuchsweise auch die Attribute `%ROWCOUNT` und `%FOUND`.
 - d. Zeigen Sie die Zeilen nach dem Einfügen in die `top_salaries`-Tabelle mit einer `SELECT`-Anweisung an. Die Ausgabe zeigt die fünf höchsten Gehälter in der `employees`-Tabelle an.

SALARY

24000
17000
14000
13500
13000

- e. Testen Sie eine Reihe von Sonderfällen wie `v_num = 0` oder Fälle, in denen `v_num` größer als die Anzahl der Mitarbeiter in der `employees`-Tabelle ist. Leeren Sie die `top_salaries`-Tabelle nach jedem Test.
2. Erstellen Sie einen PL/SQL-Block für die folgenden Aufgaben:
 - a. Deklarieren Sie im deklarativen Bereich eine `v_deptno`-Variable des NUMBER-Typs, und ordnen Sie einen Wert zu, der die Abteilungsnummer enthält.
 - b. Deklarieren Sie den Cursor `c_emp_cursor`, der `last_name`, `salary` und `manager_id` der Mitarbeiter abruft, die in der unter `v_deptno` angegebenen Abteilung arbeiten.

Übungen zu Lektion 7 (Fortsetzung)

- c. Bearbeiten Sie die abgerufenen Daten im ausführbaren Bereich mit der Cursor FOR-Schleife. Wenn das Gehalt des Mitarbeiters unter 5000 liegt und die Managernummer entweder 101 oder 124 ist, zeigen Sie die Meldung <<last_name>> Due for a raise an. Zeigen Sie andernfalls die Meldung <<last_name>> Not due for a raise an.
- d. Testen Sie den PL/SQL-Block mit folgenden Werten:

Abteilungsnummer	Meldung
10	Whalen Due for a raise
20	Hartstein Not Due for a raise Fay Not Due for a raise
50	Weiss Not Due for a raise Fripp Not Due for a raise Kaufling Not Due for a raise Vollman Not Due for a raise. OConnell Due for a raise Grant Due for a raise
80	Russell Not Due for a raise Partners Not Due for a raise Errazuriz Not Due for a raise Cambrault Not Due for a raise . . . Livingston Not Due for a raise Johnson Not Due for a raise

Übungen zu Lektion 7 (Fortsetzung)

3. Erstellen Sie einen PL/SQL-Block, der Cursor mit Parametern deklariert und verwendet.
- Um aus der departments-Tabelle die Nummer und den Namen einer Abteilung abzurufen, deren department_id kleiner als 100 ist, verwenden Sie einen Cursor in einer Schleife.
- Um aus der employees-Tabelle die Details zu Nachname, Tätigkeit, Einstellungsdatum und Gehalt der Mitarbeiter abzurufen, die in einer Abteilung mit einer employee_id unter 120 arbeiten, übergeben Sie die Abteilungsnummer als Parameter an einen anderen Cursor.
- a. Um für die Abteilungen mit einer department_id unter 100 die department_id und den department_name abzurufen, deklarieren Sie im deklarativen Bereich einen dept_cursor-Cursor. Sortieren Sie nach department_id.
 - b. Deklarieren Sie einen weiteren emp_cursor-Cursor, der die Abteilungsnummer als Parameter annimmt und last_name, job_id, hire_date und salary der Abteilungsmitarbeiter abruft, deren employee_id kleiner als 120 ist.
 - c. Deklarieren Sie Variablen, die die von den einzelnen Cursorn abgerufenen Werte aufnehmen. Deklarieren Sie Variablen mit dem %TYPE-Attribut.
 - d. Öffnen Sie dept_cursor, verwenden Sie eine einfache Schleife, und lesen Sie Werte in die deklarierten Variablen ein. Zeigen Sie die Abteilungsnummer und den Abteilungsnamen an.
 - e. Öffnen Sie emp_cursor für jede Abteilung, indem Sie die aktuelle Abteilungsnummer als Parameter übergeben. Starten Sie eine weitere Schleife, und lesen Sie die Werte von emp_cursor in Variablen ein. Geben Sie alle aus der employees-Tabelle abgerufenen Details aus.
- Hinweis:** Sie können nach den Details zu den einzelnen Abteilungen eine Zeile ausgeben. Verwenden Sie geeignete Attribute für die exit-Bedingung. Ermitteln Sie außerdem vor dem Öffnen des Cursors, ob bereits ein Cursor geöffnet ist.
- f. Schließen Sie alle Schleifen und Cursor, und beenden Sie anschließend den ausführbaren Bereich. Führen Sie das Skript aus.

Übungen zu Lektion 7 (Fortsetzung)

Die Ausgabe des Beispiels:

```
anonymous block completed
Department Number : 10 Department Name : Administration
-----
Department Number : 20 Department Name : Marketing
-----
Department Number : 30 Department Name : Purchasing
Raphaely    PU_MAN    07-DEC-94    11000
Khoo       PU_CLERK   18-MAY-95    3100
Baida      PU_CLERK   24-DEC-97    2900
Tobias     PU_CLERK   24-JUL-97    2800
Himuro     PU_CLERK   15-NOV-98    2600
Colmenares PU_CLERK   10-AUG-99    2500
-----
Department Number : 40 Department Name : Human Resources
-----
Department Number : 50 Department Name : Shipping
-----
Department Number : 60 Department Name : IT
Hunold     IT_PROG    03-JAN-90    9000
Ernst      IT_PROG    21-MAY-91    6000
Austin     IT_PROG    25-JUN-97    4800
Pataballa  IT_PROG    05-FEB-98    4800
Lorentz    IT_PROG    07-FEB-99    4200
-----
Department Number : 70 Department Name : Public Relations
-----
Department Number : 80 Department Name : Sales
-----
Department Number : 90 Department Name : Executive
King       AD_PRES    17-JUN-87    24000
Kochhar    AD_VP      21-SEP-89    17000
De Haan    AD_VP      13-JAN-93    17000
```

8

Exceptions behandeln

ORACLE®

Copyright © 2007, Oracle. Alle Rechte vorbehalten.

Oracle Internal & Oracle Academy
Use Only

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**

ORACLE®

Copyright © 2007, Oracle. Alle Rechte vorbehalten.

Lernziel

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 von einem funktionierenden Code ohne Laufzeitfehler ausgegangen. Der Code kann jedoch zur Laufzeit unvorhergesehene Fehler verursachen. In dieser Lektion wird beschrieben, wie Sie diese Fehler im PL/SQL-Block beheben.

Exceptions – 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);
END;
```

ORACLE

Copyright © 2007, Oracle. Alle Rechte vorbehalten.

Exceptions – Beispiel

Sehen Sie sich das Beispiel auf der Folie an. Der Code ist frei von Syntaxfehlern, so dass der anonyme Block erfolgreich ausführbar sein muss. Die SELECT-Anweisung im Block ruft den Nachnamen von John ab. Wenn Sie den Code ausführen, wird die folgende Ausgabe 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 diese Exceptions im PL/SQL-Block behandeln.

Exceptions – 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 © 2007, Oracle. Alle Rechte vorbehalten.

Exceptions – Beispiel (Fortsetzung)

Sie haben PL/SQL-Blöcke mit einem deklarativen Bereich (beginnend mit dem DECLARE-Schlüsselwort) und einem ausführbaren Bereich (mit den Schlüsselwörtern BEGIN und END beginnend bzw. endend) erstellt. Fügen Sie zur Exception-Behandlung einen weiteren optionalen Abschnitt namens *EXCEPTION-Abschnitt* ein. Er beginnt mit dem EXCEPTION-Schlüsselwort. Wenn vorhanden, ist dies der letzte Abschnitt eines PL/SQL-Blockes. Betrachten Sie den EXCEPTION-Abschnitt des Codes auf der Folie. Die Syntax und Anweisungen sind in diesem Fall nebenschließlich. Beides wird im weiteren Verlauf der Lektion erläutert.

Um die aufgetretene Exception zu behandeln, wurde der Code der vorherigen Folie neu geschrieben. Die Ausgabe des Codes lautet:

```
anonymous block completed
Your select statement retrieved multiple
rows. Consider using a cursor.
```

Im Gegensatz zum vorherigen Code endet das PL/SQL-Programm nicht abrupt. Wenn die Exception ausgelöst wird, übernimmt der Exception-Abschnitt die Kontrolle und alle Anweisungen im Exception-Abschnitt werden ausgeführt. Der PL/SQL-Block endet erfolgreich auf normale Weise.

Exceptions mit PL/SQL behandeln

- **Exceptions sind PL/SQL-Fehler, die bei der Ausführung von Programmen ausgelöst werden.**
- **Exceptions auslösen:**
 - Implizit vom Oracle-Server
 - Explizit vom Programm
- **Exceptions behandeln:**
 - Mit einem Handler abfangen
 - An die aufrufende Umgebung propagieren

ORACLE

Copyright © 2007, Oracle. Alle Rechte vorbehalten.

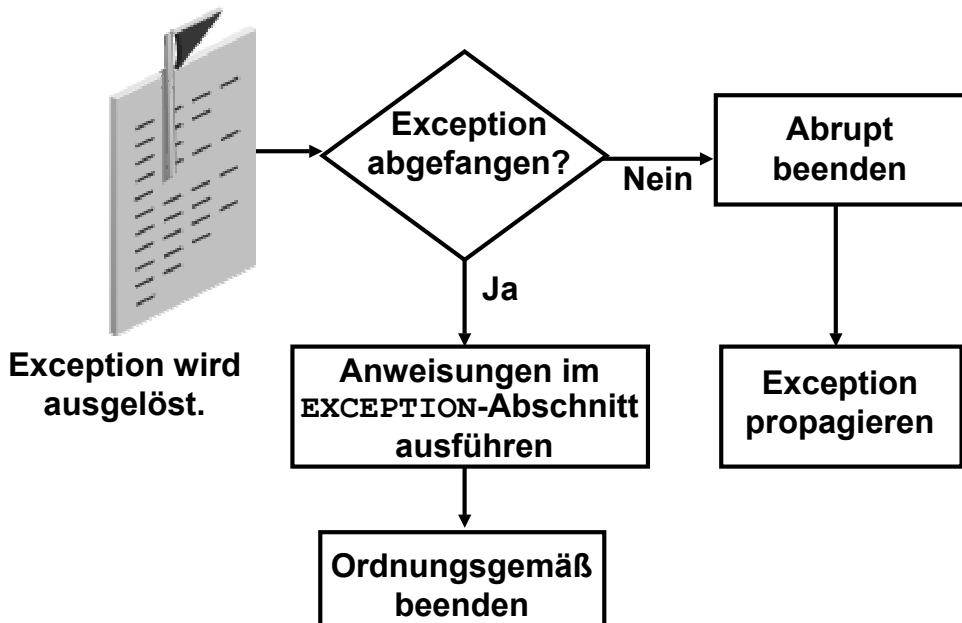
Exceptions mit PL/SQL behandeln

Eine Exception ist ein Fehler in PL/SQL, der bei der Ausführung eines Blockes ausgelöst wird. Blöcke enden immer, wenn PL/SQL eine Exception auslöst. Wenn Sie jedoch einen Exception Handler angeben, werden vor Beendigung des Blockes noch abschließende Aktionen ausgeführt.

Zwei Methoden für das Auslösen von Exceptions

- Ein Oracle-Fehler tritt auf und die damit verknüpfte Exception wird automatisch ausgelöst. Beispiel: Wenn der ORA-01403-Fehler auftritt, weil mit einer SELECT-Anweisung keine Zeilen aus der Datenbank abgerufen wurden, löst PL/SQL die Exception NO_DATA_FOUND aus. Diese Fehler werden in vordefinierte Exceptions konvertiert.
- Je nach Geschäftsfunktionalität des Programms müssen Sie eine Exception möglicherweise explizit auslösen. Setzen Sie zu diesem Zweck die RAISE-Anweisung im Block ab. Die ausgelöste Exception kann benutzerdefiniert oder vordefiniert sein. Es gibt auch einige nicht vordefinierte Oracle-Fehler. Dazu gehören alle Oracle-Standardfehler, die nicht vordefiniert sind. Sie können Exceptions explizit deklarieren und den nicht vordefinierten Oracle-Fehlern zuordnen.

Exceptions behandeln



ORACLE®

Copyright © 2007, Oracle. Alle Rechte vorbehalten.

Exceptions behandeln

Exceptions abfangen

Um Exceptions abzufangen, nehmen Sie einen EXCEPTION-Abschnitt 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-Abschnitt 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 endet erfolgreich.

Exceptions propagieren

Wenn die Exception im ausführbaren Bereich des Blockes ausgelöst wird und kein entsprechender Exception Handler vorhanden ist, endet der PL/SQL-Block mit einem Fehler. Die Exception wird daraufhin an einen umschließenden Block oder die aufrufende Umgebung propagiert. Die aufrufende Umgebung kann eine beliebige Anwendung sein (z. B. SQL*Plus, das das PL/SQL-Programm auruft).

Exception-Typen

- Vordefinierter Oracle-Server
 - Nicht vordefinierter Oracle-Server
- } Implizit ausgelöst

- Benutzerdefiniert
- Explizit ausgelöst

ORACLE®

Copyright © 2007, Oracle. Alle Rechte vorbehalten.

Exception-Typen

Es gibt drei Typen von Exceptions:

Exception	Beschreibung	Anweisungen für die Behandlung
Vordefinierter Oracle-Server-Fehler	Einer von ca. 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-Server-Fehler	Alle sonstigen Standardfehler des Oracle-Servers	Sie müssen diese 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	Erstellen Sie eine Deklaration im deklarativen Bereich, und ermöglichen Sie eine explizite Auslösung.

Hinweis: Einige Anwendungs-Tools mit Client-seitigem PL/SQL (z. B. Oracle Developer Forms) enthalten eigene Exceptions.

Exceptions abfangen

Syntax:

```
EXCEPTION
  WHEN exception1 [OR exception2 . . .] THEN
    statement1;
    statement2;
    . . .
  [WHEN exception3 [OR exception4 . . .] THEN
    statement1;
    statement2;
    . . .]
  [WHEN OTHERS THEN
    statement1;
    statement2;
    . . .]
```

ORACLE

Copyright © 2007, Oracle. Alle Rechte vorbehalten.

Exceptions abfangen

Sie können Fehler abfangen, indem Sie in den Abschnitt für die Exception-Behandlung des PL/SQL-Blockes einen entsprechenden Handler aufnehmen. Jeder Handler besteht aus einer WHEN-Klausel, die einen Exception-Namen gefolgt von einer Anwendungsfolge angibt. Die Anwendungsfolge wird bei Auslösen der Exception ausgeführt. Um bestimmte Exceptions zu behandeln, können Sie eine beliebige Anzahl von Handlern in einen EXCEPTION-Abschnitt aufnehmen. Jede Exception kann jedoch nur einen Handler haben.

Für die Syntax gilt:

<i>exception</i>	Der Standardname einer vordefinierten Exception oder der Name einer benutzerdefinierten Exception, die im deklarativen Bereich deklariert ist
<i>statement</i>	Eine oder mehrere PL/SQL- oder SQL-Anweisungen
OTHERS	Eine optionale Klausel für die Exception-Behandlung, die alle nicht explizit behandelten Exceptions abfängt

Exceptions abfangen (Fortsetzung)

WHEN OTHERS-Exception Handler

Der Abschnitt für die Exception-Behandlung fängt nur angegebene Exceptions ab. Alle anderen Exceptions können Sie nur mit dem OTHERS-Exception Handler abfangen. Dieser Handler fängt alle noch nicht behandelten Exceptions ab. Verwenden Sie daher OTHERS gegebenenfalls als letzten definierten Exception Handler.

```
WHEN NO_DATA_FOUND THEN  
    statement1;  
    ...  
WHEN TOO_MANY_ROWS THEN  
    statement1;  
    ...  
WHEN OTHERS THEN  
    statement1;
```

Betrachten Sie das vorangegangene Beispiel. Wenn das Programm die Exception NO_DATA_FOUND auslöst, werden die Anweisungen des entsprechenden Handlers ausgeführt. Wird die Exception TOO_MANY_ROWS ausgelöst, werden die Anweisungen des entsprechenden Handlers ausgeführt. Falls jedoch eine andere Exception ausgelöst wird, werden die Anweisungen des OTHERS-Exception Handlers ausgeführt.

Der OTHERS-Handler fängt alle noch nicht abgefangenen Exceptions ab. Einige Oracle-Tools haben eigene vordefinierte Exceptions, mit denen Sie Ereignisse in der Anwendung auslösen können. Der OTHERS-Handler fängt auch diese Exceptions ab.

Exceptions abfangen – Richtlinien

- Das **EXCEPTION**-Schlüsselwort steht am Anfang des Abschnitts 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 © 2007, Oracle. Alle Rechte vorbehalten.

Exceptions abfangen – Richtlinien

- Beginnen Sie den Bereich für die Exception-Behandlung des Blockes mit dem EXCEPTION-Schlüsselwort.
- 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.
- Platzieren Sie die OTHERS-Klausel nach allen anderen Klauseln für die Exception-Behandlung.
- Es ist nur eine OTHERS-Klausel zulässig.
- Sie können Exceptions nicht in Zuweisungs- oder SQL-Anweisungen verwenden.

Vordefinierte Oracle-Server-Fehler abfangen

- Referenzieren Sie den vordefinierten Namen in der Exception-Behandlungsroutine.
- Beispiele für vordefinierte Exceptions:
 - NO_DATA_FOUND
 - TOO_MANY_ROWS
 - INVALID_CURSOR
 - ZERO_DIVIDE
 - DUP_VAL_ON_INDEX

ORACLE®

Copyright © 2007, Oracle. Alle Rechte vorbehalten.

Vordefinierte Oracle-Server-Fehler abfangen

Fangen Sie einen vordefinierten Oracle-Server-Fehler ab, indem Sie seinen vordefinierten Namen innerhalb der entsprechenden Exception-Behandlungsroutine referenzieren.

Eine vollständige Liste vordefinierter Exceptions finden Sie im Handbuch *PL/SQL User's Guide and Reference*.

Hinweis: PL/SQL deklariert vordefinierte Exceptions im STANDARD-Package.

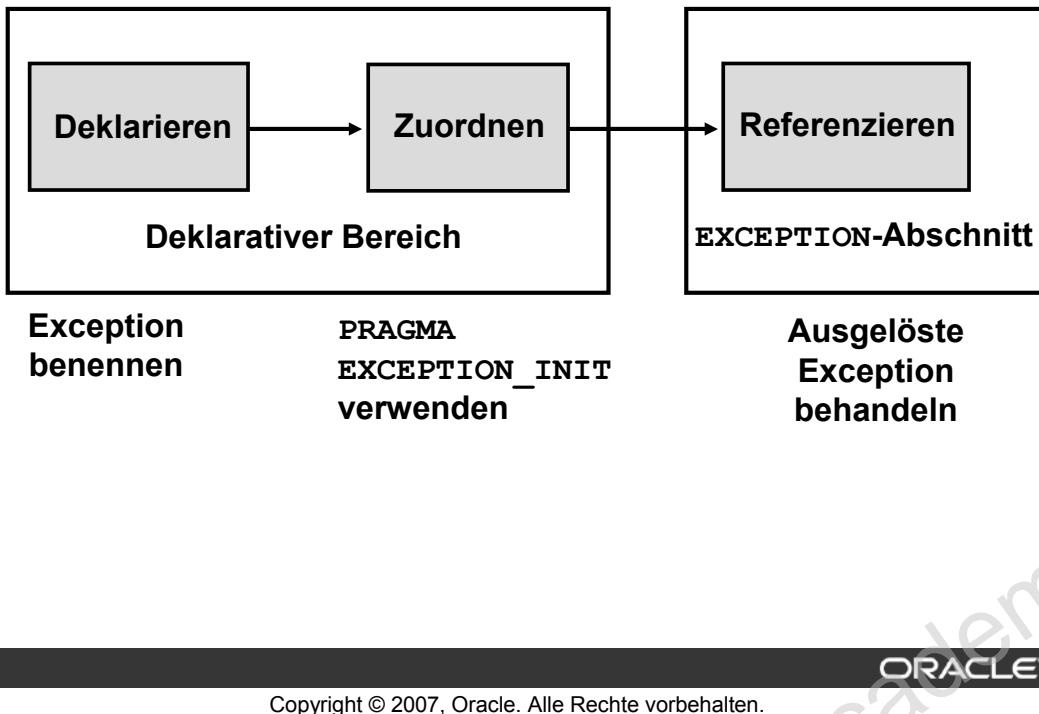
Vordefinierte Exceptions

Name der Exception	Fehlernummer des Oracle-Servers	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, andere Collection-Methoden als EXISTS auf eine nicht initialisierte Nested Table bzw. ein VARRAY 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 Cursor-Vorgang 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 Passwort.
NO_DATA_FOUND	ORA-01403	Die Single Row-Anweisung SELECT hat keine Daten zurückgegeben.
NOT_LOGGED_ON	ORA-01012	Ein 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-Cursor-Variablen haben inkompatible Rückgabetypen.

Vordefinierte Exceptions (Fortsetzung)

Name der Exception	Fehlernummer des Oracle-Servers	Beschreibung
STORAGE_ERROR	ORA-06500	Zu wenig Speicher für PL/SQL oder Speicher beschädigt.
SUBSCRIPT_BEYOND_COUNT	ORA-06533	Es wurde eine Nested Table oder ein VARRAY-Element mit Hilfe einer Index-Nummer referenziert, die größer als die Anzahl der Elemente in der Collection ist.
SUBSCRIPT_OUTSIDE_LIMIT	ORA-06532	Es wurde eine Nested Table oder ein VARRAY-Element mit Hilfe einer Index-Nummer referenziert, die außerhalb des gültigen Bereichs liegt (z. B. -1).
SYS_INVALID_ROWID	ORA-01410	Die Konvertierung einer Zeichenfolge in eine universelle ROWID war nicht erfolgreich, da die Zeichenfolge keine gültige ROWID ist.
TIMEOUT_ON_RESOURCE	ORA-00051	Timeout, während der Oracle-Server auf eine Ressource wartet.
TOO_MANY_ROWS	ORA-01422	Die Single Row-Anweisung SELECT hat mehrere Zeilen zurückgegeben.
VALUE_ERROR	ORA-06502	Fehler beim Konvertieren, Abschneiden, in der Arithmetik oder bei der Größeneinschränkung.
ZERO_DIVIDE	ORA-01476	Es wurde versucht, durch Null zu teilen.

Nicht vordefinierte Oracle-Server-Fehler abfangen



Nicht vordefinierte Oracle-Server-Fehler abfangen

Nicht vordefinierte Exceptions ähneln vordefinierten Exceptions, werden jedoch im Oracle-Server nicht als PL/SQL-Exceptions definiert. Es sind 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 definierte Oracle-Server-Fehler 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 PRAGMA-Schlüsselwort (auch als *Pseudoanweisung* bezeichnet) geben Sie an, dass es sich um eine Compiler-Anweisung 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 Fehlernummer des Oracle-Servers zu interpretieren.

Nicht vordefinierte Fehler

So fangen Sie die Fehlernummer -01400 ("cannot insert NULL") des Oracle-Servers ab:

```
DECLARE
    e_insert_excep EXCEPTION;
    PRAGMA EXCEPTION_INIT(e_insert_excep, -01400);
BEGIN
    INSERT INTO departments
        (department_id, department_name) VALUES (280, NULL);
EXCEPTION
    WHEN e_insert_excep THEN
        DBMS_OUTPUT.PUT_LINE('INSERT OPERATION FAILED');
        DBMS_OUTPUT.PUT_LINE(SQLERRM);
END;
/
```

The diagram illustrates the flow of exception handling in the provided PL/SQL code. Circle 1 points to the declaration of the exception variable `e_insert_excep`. Circle 2 points to the `PRAGMA EXCEPTION_INIT` statement which initializes the exception with error number -01400. Circle 3 points to the `WHEN` clause where the exception is caught.

ORACLE®

Copyright © 2007, Oracle. Alle Rechte vorbehalten.

Nicht vordefinierte Fehler

1. Deklarieren Sie den Namen der Exception im deklarativen Bereich.

Syntax:

```
exception EXCEPTION;
```

In dieser Syntax ist `exception` der Name der Exception.

2. Ordnen Sie die deklarierte Exception der Nummer des Standardfehlers des Oracle-Servers 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 Standardfehlers des Oracle-Servers.

3. Referenzieren Sie die deklarierte Exception innerhalb der entsprechenden Exception-Behandlungsroutine.

Beispiel

Im Beispiel auf der Folie wird versucht, den NULL-Wert für die `department_name`-Spalte der `departments`-Tabelle einzufügen. Die Operation 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 `SQLERRM`-Funktion rufen Sie die Fehlermeldung ab. `SQLERRM` wird auf den folgenden Folien näher erläutert.

Funktionen zum Afangen von Exceptions

- **SQLCODE:** Gibt den numerischen Wert des Fehler-Codes zurück
- **SQLERRM:** Gibt die der Fehlernummer zugeordnete Meldung zurück

ORACLE

Copyright © 2007, Oracle. Alle Rechte vorbehalten.

Funktionen zum Afangen von Exceptions

Bei einer Exception können Sie den zugeordneten Fehler-Code oder die Fehlermeldung mit zwei Funktionen identifizieren. Sie können die Aktionen basierend auf dem Code-Wert oder der Meldung wählen.

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 Fehler-Codes zurück. (Sie können den Wert einer NUMBER-Variablen zuweisen.)
SQLERRM	Gibt die der Fehlernummer zugeordnete Meldung zurück.

SQLCODE-Werte – Beispiele

SQLCODE-Wert	Beschreibung
0	Keine Exception aufgetreten
1	Benutzerdefinierte Exception
+100	Exception NO_DATA_FOUND
Negative Zahl	Eine andere Fehlernummer des Oracle-Servers

Funktionen zum Auffangen von Exceptions

Beispiel

```
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 © 2007, Oracle. Alle Rechte vorbehalten.

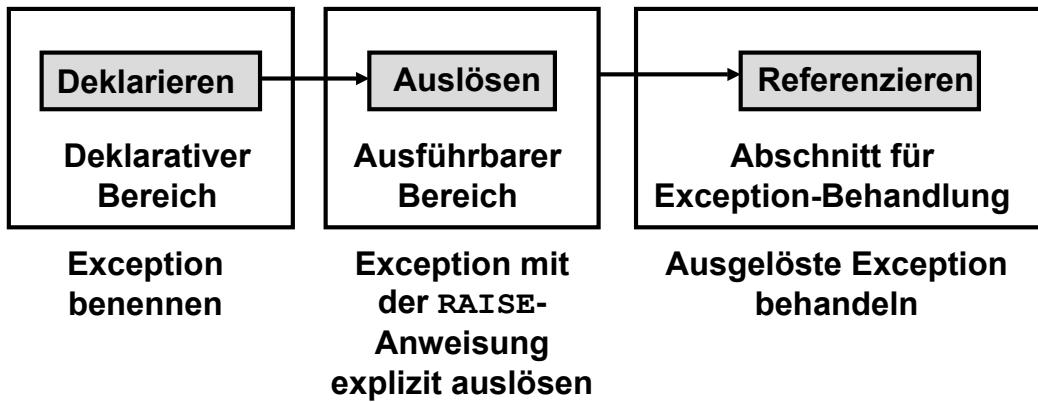
Funktionen zum Auffangen von Exceptions (Fortsetzung)

Wenn eine Exception im WHEN OTHERS-Exception Handler abgefangen wird, können Sie diese Fehler mit einer Reihe generischer Funktionen identifizieren. Im Beispiel auf der Folie sind die Werte von SQLCODE und SQLERRM Variablen zugeordnet, die von einer SQL-Anweisung verwendet werden.

Sie können SQLCODE oder SQLERRM nicht direkt in einer SQL-Anweisung verwenden. Stattdessen müssen Sie die Werte lokalen Variablen zuordnen und diese dann wie nachfolgend gezeigt in der SQL-Anweisung verwenden:

```
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 © 2007, Oracle. Alle Rechte vorbehalten.

Benutzerdefinierte Exceptions abfangen

Mit PL/SQL können Sie entsprechend den Anforderungen Ihrer Anwendung eigene Exceptions definieren. Sie können beispielsweise Benutzer auffordern, eine Abteilungsnummer einzugeben. Um Fehlerbedingungen in den Eingabedaten zu behandeln, definieren Sie Exceptions. 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-Abschnitt 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;
/
```

Copyright © 2007, Oracle. Alle Rechte vorbehalten.

ORACLE

Benutzerdefinierte Exceptions abfangen (Fortsetzung)

Fangen Sie benutzerdefinierte Exceptions ab, indem Sie sie deklarieren und explizit auslösen.

1. Deklarieren Sie den Namen der benutzerdefinierten Exception im deklarativen Bereich.

Syntax:

```
exception EXCEPTION;
```

In dieser Syntax ist *exception* der Name der Exception.

2. Lösen Sie die Exception mit Hilfe der RAISE-Anweisung explizit im ausführbaren Bereich 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

Dieser Block aktualisiert den *department_name* einer Abteilung. Der Benutzer gibt die Abteilungsnummer und den neuen Namen an. Falls die angegebene Abteilungsnummer nicht vorhanden ist, werden in der *departments*-Tabelle keine Zeilen aktualisiert. Lösen Sie eine Exception aus, und geben Sie eine Meldung für den Benutzer aus, 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 allein 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 © 2007, Oracle. Alle Rechte vorbehalten.

Exceptions in Unterblöcken propagieren

Wenn ein Unterblock eine Exception behandelt, endet er normal. Der umschließende Block übernimmt sofort nach der END-Anweisung des Unterblocks die Kontrolle.

Wenn jedoch eine PL/SQL-Anweisung eine Exception auslöst und der aktuelle Block keinen Handler für diese Exception enthält, wird die Exception an nachfolgende umschließende Blöcke propagiert, bis ein Handler gefunden wird. Wenn keiner dieser Blöcke die Exception behandelt, wird in der Host-Umgebung 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 no_rows-Exception 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, propagiert sie an den äußeren Block, wo PL/SQL den Handler sucht.

Prozedur RAISE_APPLICATION_ERROR

Syntax:

```
raise_application_error (error_number,  
                      message[, {TRUE | FALSE}]);
```

- Mit dieser Prozedur setzen Sie benutzerdefinierte Fehlermeldungen in gespeicherten Unterprogrammen ab.
- Sie können der Anwendung Fehler melden und die Rückgabe nicht behandelter Exceptions vermeiden.

ORACLE

Copyright © 2007, Oracle. Alle Rechte vorbehalten.

Prozedur RAISE_APPLICATION_ERROR

Um eine vordefinierte Exception durch Rückgabe eines speziellen Fehler-Codes und einer entsprechenden Fehlermeldung interaktiv zu übermitteln, verwenden Sie die Prozedur RAISE_APPLICATION_ERROR. Mit der Prozedur RAISE_APPLICATION_ERROR können Sie der Anwendung Fehler melden und die Rückgabe von unbehandelten Exceptions vermeiden.

Für die Syntax gilt:

<i>error_number</i>	Eine benutzerdefinierte Zahl zwischen –20.000 und –20.999 für die Exception
<i>message</i>	Die benutzerdefinierte Meldung für die Exception (eine Zeichenfolge mit bis zu 2.048 Byte)
TRUE FALSE	Optionaler boolescher Parameter. (Wenn TRUE, wird der Fehler im Stack vorheriger Fehler platziert. Wenn FALSE (Default-Wert), ersetzt der Fehler alle vorhergehenden Fehler.)

Prozedur RAISE_APPLICATION_ERROR

- **Wird an zwei Stellen verwendet:**
 - Im ausführbaren Bereich
 - Im Exception-Abschnitt
- **Gibt Fehlerbedingungen genauso wie andere Oracle-Server-Fehler an den Benutzer zurück**

ORACLE

Copyright © 2007, Oracle. Alle Rechte vorbehalten.

Prozedur RAISE_APPLICATION_ERROR (Fortsetzung)

Sie können die Prozedur RAISE_APPLICATION_ERROR im ausführbaren Bereich und/oder dem Exception-Abschnitt eines PL/SQL-Programms verwenden. Der zurückgegebene Fehler entspricht den vom Oracle-Server erstellten vordefinierten, nicht definierten oder benutzerdefinierten Fehlern. Der Benutzer sieht die Fehlernummer und die Fehlermeldung.

Prozedur RAISE_APPLICATION_ERROR

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-Abschnitt:

```
...
EXCEPTION
    WHEN NO_DATA_FOUND THEN
        RAISE_APPLICATION_ERROR (-20201,
            'Manager is not a valid employee.');
END;
```

ORACLE

Copyright © 2007, Oracle. Alle Rechte vorbehalten.

Prozedur RAISE_APPLICATION_ERROR (Fortsetzung)

Die Folie zeigt, dass die Prozedur RAISE_APPLICATION_ERROR im ausführbaren Bereich und Exception-Abschnitt eines PL/SQL-Programms verwendet werden kann.

Hier ein weiteres Beispiel zur Verwendung der Prozedur RAISE_APPLICATION_ERROR:

```
DECLARE
    e_name EXCEPTION;
    PRAGMA EXCEPTION_INIT (e_name, -20999);
BEGIN
    ...
    DELETE FROM employees
        WHERE last_name = 'Higgins';
    IF SQL%NOTFOUND THEN
        RAISE_APPLICATION_ERROR(-20999,'This is not a
            valid last name');
    END IF;
EXCEPTION
    WHEN e_name THEN
        -- handle the error
    ...
END;
/
```

Zusammenfassung

In dieser Lektion haben Sie Folgendes gelernt:

- **PL/SQL-Exceptions definieren**
- **Dem PL/SQL-Block einen EXCEPTION-Abschnitt 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 © 2007, Oracle. Alle Rechte vorbehalten.

Zusammenfassung

In dieser Lektion haben Sie gelernt, wie Sie verschiedene Typen von Exceptions behandeln.

In PL/SQL werden als Exceptions Warnungen oder Fehlerbedingungen zur Laufzeit bezeichnet.

Vordefinierte Exceptions sind vom Oracle-Server definierte Fehlerbedingungen. Nicht vordefinierte Exceptions können beliebige Standardfehler des Oracle-Servers 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 können Sie eigene Exceptions definieren.

Um beispielsweise überzogene Bankkonten zu kennzeichnen, definieren Sie eine Exception namens INSUFFICIENT_FUNDS.

Bei einem Fehler wird eine Exception ausgelöst. Die normale Ausführung stoppt und übergibt die Kontrolle an den Bereich für die 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 8 – Überblick

Diese Übungen behandeln folgende Themen:

- Benannte Exceptions behandeln**
- Benutzerdefinierte Exceptions erstellen und aufrufen**

ORACLE®

Copyright © 2007, Oracle. Alle Rechte vorbehalten.

Übungen zu Lektion 8 – Überblick

In dieser Übung erstellen Sie Exception Handler für bestimmte Situationen.

Übungen zu Lektion 8

1. In dieser Übung wird die Verwendung von vordefinierten Exceptions erläutert. Um den Namen des Mitarbeiters mit einem gegebenen Gehaltswert auszuwählen, erstellen Sie einen PL/SQL-Block.
 - a. Löschen Sie alle Records in der messages-Tabelle.
 - b. Deklarieren Sie im deklarativen Bereich zwei Variablen: v_ename des Typs employees.last_name und v_emp_sal des employees.salary-Typs. Initialisieren Sie letztere Variable mit 6000.
 - c. Rufen Sie im ausführbaren Bereich die Nachnamen der Mitarbeiter ab, deren Gehalt dem Wert in v_emp_sal entspricht.

Hinweis: Verwenden Sie keine expliziten Cursor.
Wenn das eingegebene Gehalt nur eine Zeile zurückgibt, fügen Sie in die messages-Tabelle den Namen des Mitarbeiters und die Höhe seines Gehalts ein.
 - d. Wenn das eingegebene Gehalt keine Zeilen zurückgibt, behandeln Sie die Exception mit einem entsprechenden Exception Handler, und fügen Sie in die messages-Tabelle die Meldung "No employee with a salary of <salary>" ein.
 - e. Wenn das eingegebene Gehalt keine Zeilen zurückgibt, behandeln Sie die Exception mit einem entsprechenden Exception Handler, und fügen Sie in die messages-Tabelle die Meldung "More than one employee with a salary of <salary>" ein.
 - f. Behandeln Sie beliebige weitere Exceptions mit einem entsprechenden Exception Handler, und fügen Sie in die messages-Tabelle die Meldung "Some other error occurred" ein.
 - g. Um zu prüfen, ob der PL/SQL-Block erfolgreich ausgeführt wurde, zeigen Sie die Zeilen aus der messages-Tabelle an. Die Ausgabe des Beispiels:

RESULTS
More than one employee with a salary of 6000

2. In dieser Übung wird gezeigt, wie Sie Exceptions mit einem Standardfehler des Oracle-Servers deklarieren. Verwenden Sie den Oracle-Server-Fehler ORA-02292 (integrity constraint violated – child record found).
 - a. Deklarieren Sie im deklarativen Bereich die Exception e_childrecord_exists. Ordnen Sie die deklarierte Exception dem Standardfehler -02292 des Oracle-Servers zu.
 - b. Zeigen Sie im ausführbaren Bereich "Deleting department 40 ..." an. Um die Abteilung mit der department_id 40 zu löschen, nehmen Sie eine DELETE-Anweisung auf.

Übungen zu Lektion 8 (Fortsetzung)

- c. Um die Exception `e_childrecord_exists` zu behandeln und die entsprechende Meldung anzuzeigen, fügen Sie einen Exception-Abschnitt ein. Die Ausgabe des Beispiels:

```
anonymous block completed
Deleting department 40.....
Cannot delete this department.
There are employees in this department (child records exist.)
```


9

Stored Procedures und Stored Functions erstellen

ORACLE®

Copyright © 2007, Oracle. Alle Rechte vorbehalten.

Oracle Internal & Oracle Academy
Use Only

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**

ORACLE®

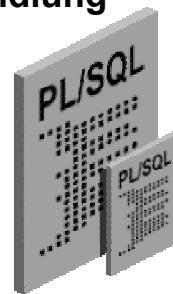
Copyright © 2007, Oracle. Alle Rechte vorbehalten.

Lernziel

Sie haben bereits anonyme Blöcke kennen gelernt. In dieser Lektion erhalten Sie eine Einführung in benannte Blöcke. Diese werden auch als *Unterprogramme* bezeichnet. Prozeduren und Funktionen sind PL/SQL-Unterprogramme. In dieser Lektion lernen Sie den Unterschied zwischen anonymen Blöcken und Unterprogrammen kennen.

Prozeduren und Funktionen

- **Benannte PL/SQL-Blöcke**
- **Werden als PL/SQL-Unterprogramme bezeichnet**
- **Haben ähnliche Blockstrukturen wie anonyme Blöcke:**
 - **Optionaler deklarativer Bereich (ohne DECLARE-Schlüsselwort)**
 - **Obligatorischer ausführbarer Bereich**
 - **Optionaler Abschnitt zur Exception-Behandlung**



ORACLE®

Copyright © 2007, Oracle. Alle Rechte vorbehalten.

Prozeduren und Funktionen

Bislang wurden in diesem Kurs nur anonyme Blöcke als Beispiele für PL/SQL-Code besprochen. Wie der Name bereits sagt, sind *anonyme* Blöcke unbenannte ausführbare PL/SQL-Blöcke. Da sie unbenannt sind, können Sie sie weder wiederverwenden noch zur späteren Verwendung speichern.

Prozeduren und Funktionen sind benannte PL/SQL-Blöcke. Sie werden auch als *Unterprogramme* bezeichnet. Diese Unterprogramme werden kompiliert und in der Datenbank gespeichert.

Unterprogramme haben eine ähnliche Blockstruktur wie anonyme Blöcke. Sie können sie nicht nur auf Schemaebene, sondern auch in einem anderen PL/SQL-Block deklarieren. Unterprogramme enthalten folgende Bereiche und Abschnitte:

Deklarativer Bereich: Unterprogramme können optional einen deklarativen Bereich enthalten.

Im Gegensatz zu anonymen Blöcken beginnt der deklarative Bereich von Unterprogrammen jedoch nicht mit dem DECLARE-Schlüsselwort. Der optionale deklarative Bereich folgt in der Unterprogrammdeklaration dem Schlüsselwort IS oder AS.

Ausführbarer Bereich: 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 bzw. END.

Exception-Abschnitt: Sie können diesen Abschnitt optional einfügen, um Exceptions zu behandeln.

Anonyme Blöcke und Unterprogramme – Unterschiede

Anonyme Blöcke	Unterprogramme
Unbenannte PL/SQL-Blöcke	Benannte PL/SQL-Blöcke
Jedes Mal kompiliert	Nur einmal kompiliert
Nicht in der Datenbank gespeichert	In der Datenbank gespeichert
Können nicht von anderen Anwendungen aufgerufen werden	Benannt, können daher von anderen Anwendungen aufgerufen werden
Geben keine Werte zurück	Aufgerufene Funktionen müssen Werte zurückgeben
Nehmen keine Parameter an	Nehmen Parameter an

ORACLE

Copyright © 2007, Oracle. Alle Rechte vorbehalten.

Anonyme Blöcke und Unterprogramme – Unterschiede

Die Tabelle auf der Folie zeigt nicht nur die Unterschiede zwischen anonymen Blöcken und Unterprogrammen, sondern verdeutlicht auch die allgemeinen Vorteile von Unterprogrammen.

Anonyme Blöcke sind keine persistenten Datenbankobjekte. Sie werden nur einmal kompiliert und ausgeführt. Sie werden nicht zur Wiederverwendung in der Datenbank gespeichert. Wenn Sie anonymous Blöcke wieder verwenden möchten, müssen Sie erneut das Skript ausführen, das den anonymen Block erstellt hat. Die Blöcke werden daraufhin rekompiliert und ausgeführt.

Prozeduren und Funktionen werden kompiliert und in kompilierter Form in der Datenbank gespeichert. Eine Rekompilierung erfolgt nur, wenn Sie sie ändern. Da diese Unterprogramme in der Datenbank gespeichert werden, kann sie jede Anwendung mit den entsprechenden Berechtigungen verwenden. Die aufrufende Anwendung kann Parameter an die Prozeduren übergeben, wenn diese Parameter annehmen. Entsprechend kann eine aufrufende Anwendung einen Wert abrufen, wenn sie eine Funktion oder Prozedur aufruft.

Prozedur – Syntax

```
CREATE [OR REPLACE] PROCEDURE procedure_name
  [(argument1 [mode1] datatype1,
    argument2 [mode2] datatype2,
    . . .)]
IS | AS
procedure_body;
```

ORACLE®

Copyright © 2007, Oracle. Alle Rechte vorbehalten.

Prozedur – Syntax

Die Folie zeigt die Syntax, mit der Sie Prozeduren erstellen. Für die Syntax gilt:

procedure_name Der Name der zu erstellenden Prozedur

argument Der Name des Prozedurparameters. Jedem Argument ist ein Modus und ein Datentyp zugeordnet. Sie können eine beliebige Anzahl von Argumenten durch Komma getrennt angeben.

mode Argumentmodus:
IN (Default)
OUT
IN OUT

datatype Der Datentyp des zugeordneten Parameters. Der Datentyp von Parametern kann keine explizite Größe haben. Verwenden Sie stattdessen %TYPE.

Procedure_body Der PL/SQL-Block, der den Code bildet

Die Argumentliste ist in Prozedurdeklarationen optional. Die Verwendung von dynamischem SQL wird im Kurs *Oracle Database 10g: PL/SQL-Datenbankprogrammierung* ausführlich behandelt.

Prozeduren – Beispiel

```
...
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 © 2007, Oracle. Alle Rechte vorbehalten.

Prozeduren – Beispiel

Betrachten Sie den Code auf der Folie. Die add_dept-Prozedur fügt eine neue Abteilung mit der Abteilungsnummer 280 und dem Abteilungsnamen ST-Curriculum ein. Die Prozedur deklariert im deklarativen Bereich die beiden Variablen dept_id und dept_name. Der deklarative Bereich einer Prozedur folgt direkt auf die Prozedurdeklaration. Er beginnt nicht mit dem DECLARE-Schlüsselwort. Die Prozedur überprüft mit dem impliziten Cursor-Attribut oder dem SQL-Attribut SQL%ROWCOUNT, ob die Zeile erfolgreich eingefügt wurde. SQL%ROWCOUNT sollte in diesem Fall 1 zurückgeben.

Hinweis: Wenn Sie ein Objekt erstellen (z. B. eine Tabelle, Prozedur oder Funktion), werden die Einträge in die user_objects-Tabelle geschrieben. Wenn der Code auf der Folie erfolgreich ausgeführt wird, können Sie die user_objects-Tabelle mit dem folgenden Befehl überprüfen:

```
SELECT object_name,object_type FROM user_objects;
```

OBJECT_NAME	OBJECT_TYPE
DEPT	TABLE
TOP_SALARIES	TABLE
ADD_DEPT	PROCEDURE

Prozeduren – Beispiel (Fortsetzung)

Die Quelle der Prozedur wird in der `user_source`-Tabelle gespeichert. Überprüfen Sie die Quelle der Prozedur mit dem folgenden Befehl:

```
SELECT * FROM user_source WHERE name='ADD_DEPT';
```

NAME	TYPE	LINE	TEXT
ADD_DEPT	PROCEDURE	1	PROCEDURE add_dept IS
ADD_DEPT	PROCEDURE	2	dept_id dept.department_id%TYPE;
ADD_DEPT	PROCEDURE	3	dept_name dept.department_name%TYPE;
ADD_DEPT	PROCEDURE	4	BEGIN
ADD_DEPT	PROCEDURE	5	dept_id:=280;
ADD_DEPT	PROCEDURE	6	dept_name:='ST-Curriculum';
ADD_DEPT	PROCEDURE	7	INSERT INTO dept(department_id,department_name)
ADD_DEPT	PROCEDURE	8	VALUES(dept_id,dept_name);
ADD_DEPT	PROCEDURE	9	DBMS_OUTPUT.PUT_LINE(' Inserted ' SQL%ROWCOUNT ' row ');
ADD_DEPT	PROCEDURE	10	END;

Oracle Internal & Oracle Academy
Use Only

Prozeduren aufrufen

```
BEGIN
  add_dept;
END ;
/
SELECT department_id, department_name FROM dept
WHERE department_id=280;
```

```
anonymous block completed
Inserted 1 row

DEPARTMENT_ID      DEPARTMENT_NAME
-----
280                ST-Curriculum

1 rows selected
```

ORACLE

Copyright © 2007, Oracle. Alle Rechte vorbehalten.

Prozeduren aufrufen

Die Folie zeigt, wie Sie eine Prozedur aus einem anonymen Block aufrufen. Nehmen Sie den Aufruf der Prozedur in den ausführbaren Bereich des anonymen Blockes auf. Sie können die Prozedur auch aus einer beliebigen Anwendung aufrufen (z. B. einer Forms- oder Java-Anwendung). Die SELECT-Anweisung im Code überprüft, ob die Zeile erfolgreich eingefügt wurde.

Sie können Prozeduren auch mit der SQL-Anweisung CALL <procedure_name> aufrufen.

Funktion – Syntax

```
CREATE [OR REPLACE] FUNCTION function_name
  [(argument1 [mode1] datatype1,
    argument2 [mode2] datatype2,
    . . .)]
  RETURN datatype
  IS | AS
  function_body;
```

ORACLE®

Copyright © 2007, Oracle. Alle Rechte vorbehalten.

Funktion – Syntax

Die Folie zeigt die Syntax, mit der Sie Funktionen erstellen. Für die Syntax gilt:

<i>function_name</i>	Der Name der zu erstellenden Funktion
<i>argument</i>	Der Name des Funktionsparameters. (Jedes Argument ist einem Modus und einem Datentyp zugeordnet. Sie können eine beliebige Anzahl von Argumenten durch Komma getrennt angeben. Übergeben Sie das Argument, wenn Sie die Funktion aufrufen.)
<i>mode</i>	Der Parametertyp. (Es dürfen nur IN-Parameter deklariert werden.)
<i>datatype</i>	Der Datentyp des zugeordneten Parameters
RETURN <i>datatype</i>	Der Datentyp des von der Funktion zurückgegebenen Wertes
<i>function_body</i>	Der PL/SQL-Block, der den Funktions-Code bildet

Die Argumentliste ist in Funktionsdeklarationen optional. Im Gegensatz zu Prozeduren müssen Funktionen einen Wert an das aufrufende Programm zurückgeben. Daher enthält die Syntax *return_type*, womit Sie den Datentyp des von der Funktion zurückgegebenen Wertes angeben. Prozeduren können Werte über den Parameter OUT oder IN OUT zurückgeben.

Funktionen – Beispiel

```
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 © 2007, Oracle. Alle Rechte vorbehalten.

Funktionen – Beispiel

Mit der `check_sal`-Funktion ermitteln Sie, ob das Gehalt eines bestimmten Mitarbeiters über oder unter dem Durchschnittsgehalt der Mitarbeiter dieser Abteilung liegt. Die Funktion gibt `TRUE` zurück, wenn das Gehalt des Mitarbeiters über dem Durchschnittsgehalt der Mitarbeiter der Abteilung liegt. Andernfalls gibt sie `FALSE` zurück. Die Funktion gibt `NULL` zurück, wenn die Exception `NO_DATA_FOUND` ausgelöst wird.

Beachten Sie, dass die Funktion nach dem Mitarbeiter mit der Personalnummer 205 sucht. Um nur nach dieser Personalnummer zu suchen, ist die Funktion hartcodiert. Wenn Sie 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 ;
/
```

```
anonymous block completed
Salary > average
```

ORACLE

Copyright © 2007, Oracle. Alle Rechte vorbehalten.

Funktionen aufrufen

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 `check_sal`-Funktion 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 wie im folgenden Beispiel gezeigt die Argumente überprüfen 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 © 2007, Oracle. Alle Rechte vorbehalten.

Parameter an Funktionen übergeben

Um das Gehalt des Mitarbeiters mit der Personalnummer 205 zu überprüfen, haben Sie die Funktion im vorherigen Beispiel hartcodiert. Der Code auf dieser Folie entfernt dieses Constraint, da er die Personalnummer als Parameter annimmt. Sie können jetzt verschiedene Personalnummern übergeben und das jeweilige Gehalt überprüfen.

Funktionen werden im Kurs *Oracle Database 10g: PL/SQL-Datenbankprogrammierung* ausführlich behandelt.

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 © 2007, Oracle. Alle Rechte vorbehalten.

Funktionen mit einem Parameter aufrufen

Der Code auf der Folie ruft die Funktion zweimal auf, indem er Parameter übergibt. Er generiert die folgende Ausgabe:

```
anonymous block completed
Checking for employee with id 205
Salary > average
Checking for employee with id 70
The function returned NULL due to exception
```

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**

ORACLE®

Copyright © 2007, Oracle. Alle Rechte vorbehalten.

Zusammenfassung

Mit anonymen Blöcken können Sie in PL/SQL beliebige Funktionalitäten entwickeln. Die Haupteinschränkung anonymer Blöcke ist jedoch, dass sie nicht gespeichert werden und daher nicht wiederverwendbar sind.

An Stelle von anonymen Blöcken können Sie PL/SQL-Unterprogramme erstellen. Prozeduren und Funktionen werden als Unterprogramme (benannte PL/SQL-Blöcke) bezeichnet. Unterprogramme drücken wieder verwendbare Logik durch Parameterisierung aus. Prozeduren und Funktionen haben eine ähnliche Struktur wie anonyme Blöcke. Die Unterprogramme werden in der Datenbank gespeichert und sind daher wiederverwendbar.

Übungen zu Lektion 9 – Überblick

Diese Übungen behandeln folgende Themen:

- Vorhandene anonyme Blöcke in Prozeduren konvertieren**
- Prozeduren ändern, um Parameter anzunehmen**
- Anonyme Blöcke erstellen, um die Prozedur aufzurufen**

ORACLE®

Copyright © 2007, Oracle. Alle Rechte vorbehalten.

Oracle Internal & Oracle Academy
Use Only

Übungen zu Lektion 9

1. Laden Sie das Skript lab_02_04_soln.sql, das Sie in der 4. Übung der 2. Lektion erstellt haben.
 - a. Bearbeiten Sie das Skript, um den anonymen Block in eine Prozedur namens greet zu konvertieren.
 - b. Führen Sie das Skript aus, um die Prozedur zu erstellen.
 - c. Speichern Sie das Skript unter dem Namen lab_09_01_soln.sql.
 - d. Um den Workspace zu leeren, klicken Sie auf die Schaltfläche **Clear**.
 - e. Um die greet-Prozedur aufzurufen, erstellen Sie einen anonymen Block und führen ihn aus. Die Ausgabe des Beispiels:

```
anonymous block completed
Hello World
TODAY IS : 30-MAY-07
TOMORROW IS : 31-MAY-07
```

2. Laden Sie das Skript lab_09_01_soln.sql.
 - a. Löschen Sie die greet-Prozedur mit dem folgenden Befehl:
DROP PROCEDURE greet
 - b. Bearbeiten Sie die Prozedur so, dass sie ein Argument des VARCHAR2-Typs annimmt. Nennen Sie das Argument p_name.
 - c. Geben Sie statt Hello World die Zeichenfolge Hello <name> aus.
 - d. Speichern Sie das Skript unter dem Namen lab_09_02_soln.sql.
 - e. Um die Prozedur zu erstellen, führen Sie das Skript aus.
 - f. Um die greet-Prozedur mit einem Parameter aufzurufen, erstellen Sie einen anonymen Block und führen ihn aus. Die Ausgabe des Beispiels:

```
anonymous block completed
Hello Neema
TODAY IS : 30-MAY-07
TOMORROW IS : 31-MAY-07
```

A

Lösungen

Oracle Internal & Oracle Academy
Use Only

Allgemeine Hinweise:

- Speichern Sie alle Übungsdateien in das folgende Verzeichnis: D:\labs\SQL1\labs.
- Geben Sie Ihre SQL-Anweisungen in SQL Worksheet ein. Um ein Skript in SQL Developer zu speichern, wählen Sie im Menü **File** die Option **Save as** oder klicken mit der rechten Maustaste in das SQL Worksheet und wählen **Save file**. Speichern Sie die SQL-Anweisung als Skript lab_<lessonno>_<stepno>.sql. Um ein vorhandenes Skript zu bearbeiten, öffnen Sie die Skriptdatei zunächst mit **File > Open**. Speichern Sie das geänderte Skript dann mit **Save as** unter einem anderen Dateinamen.
- Um die Abfrage auszuführen, klicken Sie in SQL Worksheet auf das Symbol **Execute Statement** (oder drücken Sie F9). Verwenden Sie für DML- und DDL-Anweisungen das Symbol **Run Script** (oder drücken Sie F5).
- Geben Sie, nachdem Sie ein gespeichertes Skript ausgeführt haben, Ihre nächste Abfrage nicht in dasselbe Worksheet ein. Öffnen Sie ein neues Worksheet.

Oracle Internal & Oracle Academy
Use Only

Übungen zu Lektion 1 – Einführung in PL/SQL

Sie speichern Ihre Skripts im Ordner `labs`, der das Arbeitsverzeichnis ist. Bitten Sie Ihren Dozenten, Ihnen den Pfad zum Ordner `labs` für diesen Kurs mitzuteilen. Die Lösungen zu den Übungen befinden sich im Ordner `soln`.

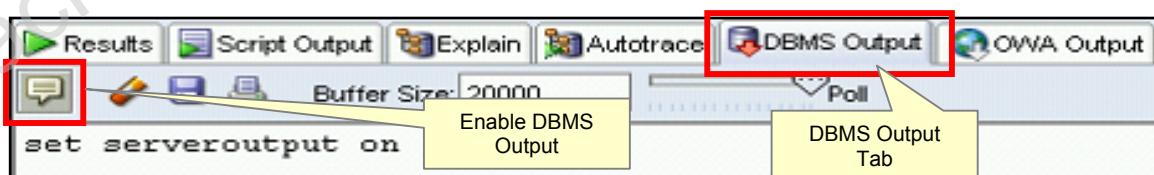
1. Welche der folgenden PL/SQL-Blöcke werden erfolgreich ausgeführt?

- a. BEGIN
END;
- b. DECLARE
amount INTEGER(10);
END;
- c. DECLARE
BEGIN
END;
- d. DECLARE
amount INTEGER(10);
BEGIN
DBMS_OUTPUT.PUT_LINE(amount);
END;

*Der Block unter **a** wird nicht ausgeführt, da der ausführbare Bereich keine Anweisungen enthält. Der Block unter **b** verfügt über keinen obligatorischen ausführbaren Bereich, der mit dem **BEGIN**-Schlüsselwort beginnt.*

*Der Block unter **c** enthält alle notwendigen Bestandteile, jedoch enthält der ausführbare Bereich keine Anweisungen.*

2. Erstellen Sie einen einfachen anonymen Block, der den Text "Hello World" ausgibt, und führen Sie ihn aus. Führen Sie das Skript aus, und speichern Sie es unter dem Namen `lab_01_02_soln.sql`.
 - a. Starten Sie SQL Developer. Der Dozent stellt Ihnen die notwendigen Informationen zur Verfügung.
 - b. Aktivieren Sie in SQL Developer die Ausgabefunktion. Klicken Sie hierfür in der Registerkarte **DBMS Output** auf die Schaltfläche **Enable DBMS Output**.



- c. Geben Sie den folgenden Code im Workspace ein.

```
BEGIN  
DBMS_OUTPUT.PUT_LINE('Hello World');  
END;
```

- d. Klicken Sie auf die Schaltfläche **Run Script**.

- e. In der Registerkarte **Script Output** wird die folgende Ausgabe angezeigt:

```
anonymous block completed  
Hello World
```

- f. Klicken Sie auf die Schaltfläche **Save**. Wählen Sie den Ordner, in dem die Datei gespeichert werden soll. Geben Sie lab_01_02_soln.sql als Dateiname ein, und klicken Sie auf die Schaltfläche **Save**.

Übungen zu Lektion 2 – PL/SQL-Variablen deklarieren

1. Geben Sie die gültigen und ungültigen Identifier an:

a. today	Gültig
b. last_name	Gültig
c. today's_date	Ungültig – Zeichen "'" nicht zulässig
d. Number_of_days_in_February_this_year	Ungültig – Zu lang
e. Isleap\$year	Gültig
f. #number	Ungültig – "#" kein gültiges Anfangszeichen
g. NUMBER#	Gültig
h. number1to7	Gültig
2. Geben Sie die gültigen und ungültigen Variablen Deklarationen und -initialisierungen an:

a. number_of_copies	PLS_INTEGER;	Gültig
b. PRINTER_NAME	constant VARCHAR2(10);	Ungültig
c. deliver_to	VARCHAR2(10) := Johnson;	Ungültig
d. by_when	DATE := SYSDATE + 1;	Gültig

Die Deklaration unter b ist ungültig, weil Konstanten bei der Deklaration initialisiert werden müssen.

Die Deklaration unter c ist ungültig, weil Zeichenfolgenliterale in einfache Anführungszeichen eingeschlossen werden müssen.

3. Prüfen Sie den folgenden anonymen Block, und wählen Sie die richtige Aussage.

```
DECLARE
    v_fname VARCHAR2(20);
    v_lname VARCHAR2(15) DEFAULT 'fernandez';
BEGIN
    DBMS_OUTPUT.PUT_LINE(v_fname || ' ' || v_lname);
END;
```

- a. Der Block wird erfolgreich ausgeführt. Er gibt "fernandez" aus.
 - b. Der Block erzeugt einen Fehler, weil die fname-Variable ohne Initialisierung verwendet wird.
 - c. Der Block wird erfolgreich ausgeführt. Er gibt "null fernandez" aus.
 - d. Der Block erzeugt einen Fehler, da Sie mit dem DEFAULT-Schlüsselwort keine Variablen des VARCHAR2-Typs initialisieren können.
 - e. Der Block erzeugt einen Fehler, da die v_fname-Variable nicht deklariert ist.
 - a. **Der Block wird erfolgreich ausgeführt. Er gibt "fernandez" aus.**
4. Erstellen Sie einen anonymen Block. Laden Sie in SQL Plus das Skript lab_01_02_soln.sql, das Sie in der zweiten Aufgabe der Übungen zu Lektion 1 erstellt haben:
Wählen Sie **File > Open**.
Wählen Sie die Datei lab_01_02_soln.sql. Klicken Sie auf die Schaltfläche **Open**. In Ihrem Workspace befindet sich der Code in der .sql-Datei.

- a. Fügen Sie diesem PL/SQL-Block einen deklarativen Bereich hinzu. Deklarieren Sie im deklarativen Bereich die folgenden Variablen:

1. today-Variable des DATE-Typs. Initialisieren Sie today mit SYSDATE.

```
DECLARE  
  v_today DATE:=SYSDATE;
```

2. tomorrow-Variable des today-Typs. Deklarieren Sie die Variable mit dem %TYPE-Attribut.

```
v_tomorrow v_today%TYPE;
```

- b. Initialisieren Sie im ausführbaren Bereich die tomorrow-Variable mit einem Ausdruck, der das Datum von morgen berechnet (erhöhen Sie den Wert in today um eins). Geben Sie die Werte von today und tomorrow im Anschluss an "Hello World" aus.

```
BEGIN  
  v_tomorrow:=v_today +1;  
  DBMS_OUTPUT.PUT_LINE('Hello World');  
  DBMS_OUTPUT.PUT_LINE('TODAY IS : '|| v_today);  
  DBMS_OUTPUT.PUT_LINE('TOMORROW IS : ' || v_tomorrow);  
END;
```

- c. Führen Sie Ihr Skript aus, und speichern Sie es unter dem Namen lab_02_04_soln.sql. Führen Sie die Anweisungen unter Schritt 2 f) der Übung zu Lektion 1 aus, um die Datei zu speichern. Die Ausgabe lautet:

```
anonymous block completed  
Hello World  
TODAY IS : 09-MAY-07  
TOMORROW IS : 10-MAY-07
```

5. Bearbeiten Sie das Skript lab_02_04_soln.sql.

- Fügen Sie Code hinzu, um zwei Bind-Variablen zu erstellen.
Erstellen Sie die Bind-Variablen `basic_percent` und `pf_percent` des NUMBER-Typs.

```
VARIABLE b_basic_percent NUMBER  
VARIABLE b_pf_percent NUMBER
```

- Weisen Sie den Bind-Variablen `basic_percent` und `pf_percent` im ausführbaren Bereich des PL/SQL-Blockes die Werte 45 bzw. 12 zu.

```
:b_basic_percent:=45;  
:b_pf_percent:=12;
```

- Beenden Sie den PL/SQL-Block mit einem Schrägstrich (/), und zeigen Sie den Wert der Bind-Variablen mit dem PRINT-Befehl an.

```
/  
PRINT b_basic_percent  
PRINT b_pf_percent
```

ODER

```
PRINT
```

- Führen Sie das Skript aus, und speichern Sie es unter dem Namen `lab_02_05_soln.sql`. Die Ausgabe lautet:

```
anonymous block completed  
Hello World  
TODAY IS : 16-MAY-07  
TOMORROW IS : 17-MAY-07  
  
basic_percent  
--  
45  
  
pf_percent  
--  
12
```

Übungen zu Lektion 3 – Ausführbare Anweisungen erstellen

```
DECLARE
  v_weight      NUMBER(3) := 600;
  v_message     VARCHAR2(255) := 'Product 10012';
BEGIN
  DECLARE
    v_weight      NUMBER(3) := 1;
    v_message     VARCHAR2(255) := 'Product 11001';
    v_new_locn   VARCHAR2(50) := 'Europe';
  BEGIN
    v_weight := v_weight + 1;
    v_new_locn := 'Western ' || v_new_locn;
  1 → END;
  v_weight := v_weight + 1;
  v_message := v_message || ' is in stock';
  v_new_locn := 'Western ' || v_new_locn;
  2 →
END;
/
```

1. Beurteilen Sie den vorangegangenen PL/SQL-Block, und bestimmen Sie den Datentyp und Wert jeder der folgenden Variablen entsprechend den Regeln für Gültigkeitsbereiche.
 - a. Wert von `v_weight` an 1. Position:
2
Der Datentyp ist NUMBER.
 - b. Wert von `v_new_locn` an 1. Position:
Western Europe
Der Datentyp ist VARCHAR2.
 - c. Wert von `v_weight` an 2. Position:
601
Der Datentyp ist NUMBER.
 - d. Wert von `v_message` an 2. Position:
Product 10012 is in stock.
Der Datentyp ist VARCHAR2.
 - e. Wert von `v_new_locn` an 2. Position:
Unzulässig, da `v_new_locn` außerhalb des Unterblockes nicht angezeigt wird.

```

DECLARE
    v_customer      VARCHAR2(50) := 'Womansport';
    v_credit_rating VARCHAR2(50) := 'EXCELLENT';
BEGIN
    DECLARE
        v_customer      NUMBER(7) := 201;
        v_name         VARCHAR2(25) := 'Unisports';
    BEGIN
        v_credit_rating := 'GOOD';
        ...
    END;
    ...
END;

```

2. Bestimmen Sie im vorangegangenen PL/SQL-Block den Wert und Datentyp für jeden der folgenden Fälle.

- a. Wert von `v_customer` im verschachtelten Block:

201

Der Datentyp ist NUMBER.F

- b. Wert von `v_name` im verschachtelten Block:

Unisports

Der Datentyp ist VARCHAR2.

- c. Wert von `v_credit_rating` im verschachtelten Block:

GOOD

Der Datentyp ist VARCHAR2.

- d. Wert von `v_customer` im Hauptblock:

Womansport

Der Datentyp ist VARCHAR2.

- e. Wert von `v_name` im Hauptblock:

name wird im Hauptblock nicht angezeigt. Es wird eine Fehlermeldung eingeblendet.

- f. Wert von `v_credit_rating` im Hauptblock:

GOOD

Der Datentyp ist VARCHAR2.

3. Verwenden Sie die Session, mit der Sie schon die Übungen in der Lektion "PL/SQL-Variablen deklarieren" ausgeführt haben. Wenn Sie eine neue Session geöffnet haben, führen Sie `lab_02_05_soln.sql` aus. Bearbeiten Sie `lab_02_05_soln.sql`.

- a. Kommentieren Sie die Zeilen zum Erstellen der Bind-Variablen mit einzeiligen Kommentaren.

```

-- VARIABLE b_basic_percent NUMBER
-- VARIABLE b_pf_percent NUMBER

```

- b. Kommentieren Sie im ausführbaren Bereich die Zeilen zum Zuweisen von Werten zu Bind-Variablen mit mehrzeiligen Kommentaren.

```
/*:b_basic_percent:=45;
:b_pf_percent:=12;*/
```

- c. Deklarieren Sie die Variablen: fname des VARCHAR2-Typs mit der Größe 15 und emp_sal des NUMBER-Typs mit der Größe 10.

```
DECLARE
  v_basic_percent NUMBER:=45;
  v_pf_percent NUMBER:=12;
  v_fname VARCHAR2(15);
  v_emp_sal NUMBER(10);
```

- d. Fügen Sie die folgende SQL-Anweisung in den ausführbaren Bereich ein:

```
SELECT first_name, salary INTO v_fname, v_emp_sal
  FROM employees WHERE employee_id=110;
```

- e. Ändern Sie die Zeile für die Anzeige von "Hello World" so, dass "Hello" und der Vorname angezeigt wird. Sie können wahlweise die Zeilen für die Datumsanzeige kommentieren und die Bind-Variablen anzeigen.

```
DBMS_OUTPUT.PUT_LINE(' Hello '|| v_fname);
```

- f. Berechnen Sie den Beitrag des Mitarbeiters zur Unterstützungskasse.
Der Anteil für die Unterstützungskasse liegt bei 12 % des Grundgehalts und das Grundgehalt beträgt 45 % des Gehalts. Verwenden Sie zum Berechnen die Bind-Variablen. Verwenden Sie möglichst nur einen Ausdruck, um den Betrag für die Unterstützungskasse zu berechnen. Geben Sie das Gehalt des Mitarbeiters und seinen Beitrag zur Unterstützungskasse aus.

```
DBMS_OUTPUT.PUT_LINE('YOUR SALARY IS : '||v_emp_sal);
DBMS_OUTPUT.PUT_LINE('YOUR CONTRIBUTION TOWARDS PF:
  '||v_emp_sal*v_basic_percent/100*v_pf_percent/100);
END;
```

- g. Führen Sie das Skript aus, und speichern Sie es unter dem Namen lab_03_03_soln.sql. Die Ausgabe kann beispielsweise wie folgt lauten:

```
anonymous block completed
Hello John
YOUR SALARY IS : 8200
YOUR CONTRIBUTION TOWARDS PF:
  442.8
```

Übungen zu Lektion 4 – Mit dem Oracle-Server interagieren

1. Erstellen Sie einen PL/SQL-Block, der in der departments-Tabelle die höchste Abteilungsnummer wählt und in die Variable v_max_deptno speichert. Zeigen Sie die höchste Abteilungsnummer an.

- a. Deklarieren Sie im deklarativen Bereich eine Variable v_max_deptno des NUMBER-Typs.

```
DECLARE  
  v_max_deptno  NUMBER;
```

- b. Starten Sie den ausführbaren Bereich mit dem BEGIN-Schlüsselwort. Nehmen Sie eine SELECT-Anweisung auf, mit der Sie die höchste department_id aus der departments-Tabelle abrufen können.

```
BEGIN  
  SELECT MAX(department_id)  INTO v_max_deptno  FROM departments;
```

- c. Zeigen Sie v_max_deptno an, und beenden Sie den ausführbaren Bereich.

```
DBMS_OUTPUT.PUT_LINE('The maximum department_id is : ' || v_max_deptno);  
END;
```

- d. Führen Sie Ihr Skript als lab_04_01_soln.sql aus, und speichern Sie es. Die Ausgabe kann beispielsweise wie folgt lauten:

```
anonymous block completed  
The maximum department_id is : 270
```

2. Bearbeiten Sie den in der 1. Übung erstellten PL/SQL-Block, und fügen Sie eine neue Abteilung in die departments-Tabelle ein.

- a. Laden Sie das Skript lab_04_01_soln.sql. Deklarieren Sie zwei Variablen:
v_dept_name des Typs departments.department_name.
v_dept_id des NUMBER-Typs.
Weisen Sie der Variablen v_dept_name im deklarativen Bereich die Bezeichnung Education zu.

```
v_dept_name departments.department_name%TYPE:= 'Education';  
v_dept_id NUMBER;
```

- b. Sie haben die derzeit höchste Abteilungsnummer bereits aus der departments-Tabelle abgerufen. Erhöhen Sie die Zahl um 10, und weisen Sie das Ergebnis dept_id zu.

```
v_dept_id := 10 + v_max_deptno;  
...
```

- c. Nehmen Sie eine INSERT-Anweisung auf, um Daten in die Spalten department_name, department_id und location_id der departments-Tabelle einzufügen.
 Verwenden Sie die Werte in dept_name und dept_id für department_name bzw. department_id, und verwenden Sie NULL für location_id.

```
...
INSERT INTO departments (department_id, department_name, location_id)
VALUES (v_dept_id, v_dept_name, NULL);
```

- d. Mit dem SQL-Attribut SQL%ROWCOUNT können Sie die Anzahl der betroffenen Zeilen anzeigen.

```
DBMS_OUTPUT.PUT_LINE (' SQL%ROWCOUNT gives ' || SQL%ROWCOUNT);
...
```

- e. Führen Sie eine SELECT-Anweisung aus, um zu prüfen, ob die neue Abteilung eingefügt wurde. Sie können den PL/SQL-Block mit "/" abschließen und die SELECT-Anweisung in Ihr Skript aufnehmen.

```
...
/
SELECT * FROM departments WHERE department_id= 280;
```

- f. Führen Sie Ihr Skript aus, und speichern Sie es unter dem Namen lab_04_02_soln.sql. Die Ausgabe kann beispielsweise wie folgt lauten:

```
anonymous block completed
The maximum department_id is : 270
SQL%ROWCOUNT gives 1
```

DEPARTMENT_ID	DEPARTMENT_NAME	MANAGER_ID	LOCATION_ID
280	Education	(null)	(null)

3. In der 2. Übung stellen Sie location_id auf NULL ein. Erstellen Sie einen PL/SQL-Block, der die location_id für die neue Abteilung auf 3000 aktualisiert. Mit der Bind-Variablen dept_id können Sie die Zeile aktualisieren.

- a. Wenn Sie eine neue Session gestartet haben, löschen Sie die in die departments-Tabelle eingefügte Abteilung, und führen Sie das Skript lab_04_02_soln.sql aus.

```
DELETE FROM departments WHERE department_id=280;
```

- b. Starten Sie den ausführbaren Bereich des Blockes mit dem BEGIN-Schlüsselwort. Nehmen Sie die UPDATE-Anweisung auf, um die location_id für die neue Abteilung (dept_id = 280) auf 3000 einzustellen.

```
BEGIN
    UPDATE departments SET location_id=3000 WHERE
    department_id=280;
```

- c. Beenden Sie den ausführbaren Bereich des Blockes mit dem END-Schlüsselwort.
Schließen Sie den PL/SQL-Block mit "/" ab. Nehmen Sie eine SELECT-Anweisung auf, um die aktualisierte Abteilung anzuzeigen.

```
END;
/
SELECT * FROM departments WHERE department_id=280;
```

- d. Nehmen Sie eine DELETE-Anweisung auf, um die hinzugefügte Abteilung zu löschen.

```
DELETE FROM departments WHERE department_id=280;
```

- e. Führen Sie Ihr Skript aus, und speichern Sie es unter dem Namen lab_04_03_soln.sql. Hier eine Beispielausgabe:

```
anonymous block completed
DEPARTMENT_ID  DEPARTMENT_NAME  MANAGER_ID  LOCATION_ID
-----
280           Education          3000
1 rows selected
1 rows deleted
```

Übungen zu Lektion 5 – Kontrollstrukturen erstellen

1. Führen Sie den Befehl in der Datei lab_05_01.sql aus, um die messages-Tabelle zu erstellen. Erstellen Sie einen PL/SQL-Block, um Zahlen in die messages-Tabelle einzufügen.
 - a. Fügen Sie die Zahlen 1 bis 10 außer 6 und 8 ein.
 - b. Führen Sie vor dem Ende des Blockes einen Commit-Befehl durch.

```
BEGIN
FOR i in 1..10 LOOP
  IF i = 6 or i = 8 THEN
    null;
  ELSE
    INSERT INTO messages (results)
    VALUES (i);
  END IF;
END LOOP;
COMMIT;
END;
/
```

- c. Führen Sie eine SELECT-Anweisung aus, um sicherzustellen, dass Ihr PL/SQL-Block funktioniert.

```
SELECT * FROM messages;
```

Folgende Meldung sollte eingeblendet werden:

```
anonymous block completed
RESULTS
-----
1
2
3
4
5
7
9
10
8 rows selected
```

2. Führen Sie das Skript lab_05_02.sql aus. Dieses Skript erstellt eine emp-Tabelle, die ein Replikat der employees-Tabelle ist. Es fügt eine neue Spalte (stars, Datentyp VARCHAR2 und Größe 50) in die emp-Tabelle ein. Erstellen Sie einen PL/SQL-Block, der pro 1000 Euro des Mitarbeitergehalts ein Sternchen in die stars-Spalte einfügt. Speichern Sie Ihr Skript unter dem Namen lab_05_02_soln.sql.

- a. Deklarieren Sie im deklarativen Bereich des Blockes eine v_empno-Variable des Typs emp.employee_id, und initialisieren Sie sie auf 176. Deklarieren Sie eine v_asterisk-Variable des emp.stars-Typs, und initialisieren Sie sie auf NULL. Erstellen Sie eine sal-Variable des emp.salary-Typs.

```
DECLARE
  v_empno      emp.employee_id%TYPE := 176;
  v_asterisk   emp.stars%TYPE := NULL;
  v_sal        emp.salary%TYPE;
```

- b. Erstellen Sie im ausführbaren Bereich die Logik, um der Zeichenfolge für jede 1.000 Euro Gehalt ein Sternchen (*) hinzuzufügen. Wenn der Mitarbeiter beispielsweise 8.000 Euro verdient, sollte die Zeichenfolge acht Sternchen enthalten. Verdient der Mitarbeiter 12.500 Euro, sollte die Zeichenfolge 13 Sternchen enthalten.

```
SELECT NVL(ROUND(salary/1000), 0)  INTO v_sal
  FROM emp WHERE employee_id = v_empno;

FOR i IN 1..v_sal
  LOOP
    v_asterisk := v_asterisk || '*';
  END LOOP;
```

- c. Aktualisieren Sie die stars-Spalte für den Mitarbeiter mit der Sternzeichenfolge. Führen Sie vor dem Ende des Blockes einen Commit-Befehl durch.

```
UPDATE emp SET stars = v_asterisk
WHERE employee_id = v_empno;
COMMIT;
```

- d. Zeigen Sie die Zeilen aus der emp-Tabelle an, um zu prüfen, ob der PL/SQL-Block erfolgreich ausgeführt wurde.

```
SELECT employee_id, salary, stars
  FROM emp WHERE employee_id = 176;
```

- e. Führen Sie Ihr Skript aus, und speichern Sie es unter dem Namen lab_05_02_soln.sql. Hier eine Beispieldatenausgabe:

anonymous block completed		
EMPLOYEE_ID	SALARY	STARS
176	8600	*****

1 rows selected

Übungen zu Lektion 6 – Zusammengesetzte Datentypen verwenden

1. Erstellen Sie einen PL/SQL-Block, der Informationen über ein bestimmtes Land ausgibt.
 - a. Deklarieren Sie einen PL/SQL-Record basierend auf der Struktur der countries-Tabelle.
 - b. Deklarieren Sie eine v_countryid-Variable. Ordnen Sie CA zu v_countryid zu.

```
SET VERIFY OFF
DECLARE
    v_countryid varchar2(20) := 'CA';
```

- c. Deklarieren Sie im deklarativen Bereich mit dem %ROWTYPE-Attribut die Variable v_country_record des countries-Typs.

```
v_country_record countries%ROWTYPE;
```

- d. Rufen Sie im ausführbaren Bereich mit v_countryid alle Informationen aus der countries-Tabelle ab. Zeigen Sie die ausgewählten Informationen zum Land an. Die Ausgabe kann beispielsweise wie folgt lauten:

```
BEGIN
    SELECT      *
    INTO v_country_record
    FROM countries
    WHERE country_id = UPPER(v_countryid);

    DBMS_OUTPUT.PUT_LINE ('Country Id: ' || v_country_record.country_id ||
        ' Country Name: ' || v_country_record.country_name
        || ' Region: ' || v_country_record.region_id);

END;
```

```
anonymous block completed
Country Id: CA Country Name: Canada Region: 2
```

- e. Testen Sie ggf. den PL/SQL-Block für die Länder mit den IDs DE, UK und US.

2. Um den Namen einiger Abteilungen aus der departments-Tabelle abzurufen, erstellen Sie einen PL/SQL-Block. Geben Sie die jeweiligen Abteilungsnamen auf dem Bildschirm aus, indem Sie eine INDEX BY-Tabelle aufnehmen. Speichern Sie Ihr Skript unter dem Namen lab_06_02_soln.sql.
- Deklarieren Sie eine INDEX BY-Tabelle dept_table_type des Typs departments.department_name. Um den Namen der Abteilungen temporär zu speichern, deklarieren Sie eine Variable my_dept_table des Typs dept_table_type.

```
DECLARE
  TYPE dept_table_type is table of departments.department_name%TYPE
  INDEX BY PLS_INTEGER;
  my_dept_table dept_table_type;
```

- Deklarieren Sie zwei Variablen: f_loop_count und v_deptno des NUMBER-Typs. Ordnen Sie f_loop_count die Zahl 10 und v_deptno die Zahl 0 zu.

```
f_loop_count      NUMBER (2) := 10;
v_deptno          NUMBER (4) := 0;
```

- Rufen Sie mit Hilfe einer Schleife die Namen von 10 Abteilungen ab, und speichern Sie sie in der INDEX BY-Tabelle. Beginnen Sie mit der department_id 10. Erhöhen Sie deptno bei jeder Iteration der Schleife um 10. Die folgende Tabelle zeigt die department_id, für die Sie department_name abrufen und in der INDEX BY-Tabelle speichern sollen.

DEPARTMENT_ID	DEPARTMENT_NAME
10	Administration
20	Marketing
30	Purchasing
40	Human Resources
50	Shipping
60	IT
70	Public Relations
80	Sales
90	Executive
100	Finance

```

BEGIN
  FOR i IN 1..f_loop_count
  LOOP
    v_deptno:=v_deptno+10;
    SELECT department_name
    INTO my_dept_table(i)
    FROM departments
    WHERE department_id = v_deptno;
  END LOOP;

```

- d. Rufen Sie mit einer weiteren Schleife die Abteilungsnamen aus der INDEX BY-Tabelle ab, und zeigen Sie sie an.

```

FOR i IN 1..f_loop_count
LOOP
  DBMS_OUTPUT.PUT_LINE (my_dept_table(i));
END LOOP;
END;

```

- e. Führen Sie Ihr Skript aus, und speichern Sie es unter dem Namen lab_06_02_soln.sql. Die Ausgabe kann beispielsweise wie folgt lauten:

```

anonymous block completed
Administration
Marketing
Purchasing
Human Resources
Shipping
IT
Public Relations
Sales
Executive
Finance

```

3. Bearbeiten Sie den in der 2. Übung erstellten Block so, dass alle Informationen über die einzelnen Abteilungen aus der departments-Tabelle abgerufen und angezeigt werden. Verwenden Sie eine INDEX BY-Tabelle mit Records.
- Laden Sie das Skript lab_06_02_soln.sql.
 - Sie haben als Typ der INDEX BY-Tabelle departments.department_name deklariert. Ändern Sie die Deklaration der INDEX BY-Tabelle, so dass die Nummer, der Name und der Standort aller Abteilungen temporär gespeichert werden. Verwenden Sie das %ROWTYPE-Attribut.

```

DECLARE
    TYPE dept_table_type is table of departments%ROWTYPE
    INDEX BY PLS_INTEGER;
    my_dept_table dept_table_type;
    f_loop_count      NUMBER (2):=10;
    v_deptno          NUMBER (4):=0;

```

- c. Um alle derzeit in der departments-Tabelle gespeicherten Abteilungsinformationen abzurufen, bearbeiten Sie die SELECT-Anweisung. Speichern Sie sie in der INDEX BY-Tabelle.

```

BEGIN
    FOR i IN 1..f_loop_count
    LOOP
        v_deptno := v_deptno + 10;
        SELECT *
        INTO my_dept_table(i)
        FROM departments
        WHERE department_id = v_deptno;
    END LOOP;

```

- d. Rufen Sie mit einer weiteren Schleife die Abteilungsinformationen aus der INDEX BY-Tabelle ab, und zeigen Sie sie an. Die Ausgabe kann beispielsweise wie folgt lauten:

```

FOR i IN 1..f_loop_count
LOOP
    DBMS_OUTPUT.PUT_LINE ('Department Number: ' ||
my_dept_table(i).department_id
    || ' Department Name: ' || my_dept_table(i).department_name
    || ' Manager Id: ' || my_dept_table(i).manager_id
    || ' Location Id: ' || my_dept_table(i).location_id);
END LOOP;
END;

```

```

anonymous block completed
Department Number: 10 Department Name: Administration Manager
Id: 200 Location Id: 1700
Department Number: 20 Department Name: Marketing Manager Id: 201
Location Id: 1800
Department Number: 30 Department Name: Purchasing Manager Id:
114 Location Id: 1700
Department Number: 40 Department Name: Human Resources Manager
Id: 203 Location Id: 2400
Department Number: 50 Department Name: Shipping Manager Id: 121
Location Id: 1500

```

```
Department Number: 60 Department Name: IT Manager Id: 103
Location Id: 1400
Department Number: 70 Department Name: Public Relations Manager
Id: 204 Location Id: 2700
Department Number: 80 Department Name: Sales Manager Id: 145
Location Id: 2500
Department Number: 90 Department Name: Executive Manager Id: 100
Location Id: 1700
Department Number: 100 Department Name: Finance Manager Id: 108
Location Id: 1700
```

Oracle Internal & Oracle Academy
Use Only

Übungen zu Lektion 7 – Explizite Cursor

1. Erstellen Sie einen PL/SQL-Block, der die ersten n Spitzengehälter ermittelt.
 - a. Führen Sie das Skript lab_07_01.sql aus, um eine neue top_salaries-Tabelle zu erstellen, in der Sie die Gehälter der Mitarbeiter speichern.
 - b. Deklarieren Sie im deklarativen Bereich eine v_num-Variable des NUMBER-Typs, die eine Zahl n enthält. n steht für die ersten n Spitzengehälter aus der employees-Tabelle. Beispiel: Um die fünf höchsten Gehälter anzuzeigen, geben Sie 5 ein. Deklarieren Sie eine weitere sal-Variable des employees.salary-Typs. Deklarieren Sie einen Cursor, c_emp_cursor, der die Gehälter der Mitarbeiter in absteigender Reihenfolge abruft. Die Gehälter dürfen nicht dupliziert werden.

```
DECLARE
  v_num      NUMBER(3) := 5;
  v_sal      employees.salary%TYPE;
  CURSOR     c_emp_cursor IS
    SELECT salary
    FROM   employees
    ORDER BY salary DESC;
```

- c. Öffnen Sie die Schleife im ausführbaren Bereich. Rufen Sie die höchsten n Gehälter ab, und fügen Sie sie in die top_salaries-Tabelle ein. Sie können die Daten mit einer einfachen Schleife bearbeiten. Verwenden Sie für die exit-Bedingung versuchsweise auch die Attribute %ROWCOUNT und %FOUND.

```
BEGIN
  OPEN c_emp_cursor;
  FETCH c_emp_cursor INTO v_sal;
  WHILE c_emp_cursor%ROWCOUNT <= v_num AND c_emp_cursor%FOUND LOOP
    INSERT INTO top_salaries (salary)
    VALUES (v_sal);
    FETCH c_emp_cursor INTO v_sal;
  END LOOP;
  CLOSE c_emp_cursor;
END;
```

- d. Zeigen Sie die Zeilen nach dem Einfügen in die `top_salaries`-Tabelle mit einer `SELECT`-Anweisung an. Die Ausgabe zeigt die fünf höchsten Gehälter in der `employees`-Tabelle an.

```
/  
SELECT * FROM top_salaries;
```

SALARY

24000
17000
14000
13500
13000

- e. Testen Sie eine Reihe von Sonderfällen wie `v_num = 0` oder Fälle, in denen `v_num` größer ist als die Anzahl der Mitarbeiter in der `employees`-Tabelle. Leeren Sie die `top_salaries`-Tabelle nach jedem Test.
2. Erstellen Sie einen PL/SQL-Block für die folgenden Aufgaben:

- a. Deklarieren Sie im deklarativen Bereich eine `v_deptno`-Variable des NUMBER-Typs, und ordnen Sie einen Wert zu, der die Abteilungsnummer enthält.

```
DECLARE  
v_deptno NUMBER := 10;
```

- b. Deklarieren Sie den Cursor `c_emp_cursor`, der `last_name`, `salary` und `manager_id` der Mitarbeiter abruft, die in der unter `v_deptno` angegebenen Abteilung arbeiten.

```
CURSOR c_emp_cursor IS  
SELECT last_name, salary, manager_id  
FROM employees  
WHERE department_id = v_deptno;
```

- c. Bearbeiten Sie die abgerufenen Daten im ausführbaren Bereich mit der Cursor FOR-Schleife. Wenn das Gehalt des Mitarbeiters unter 5000 liegt und die Managernummer entweder 101 oder 124 ist, zeigen Sie die Meldung `<<last_name>> Due for a raise` an. Zeigen Sie andernfalls die Meldung `<<last_name>> Not due for a raise` an.

```
BEGIN  
FOR emp_record IN c_emp_cursor  
LOOP  
IF emp_record.salary < 5000 AND (emp_record.manager_id=101 OR  
emp_record.manager_id=124) THEN
```

```

        DBMS_OUTPUT.PUT_LINE (emp_record.last_name || ' Due for a raise');
ELSE
    DBMS_OUTPUT.PUT_LINE (emp_record.last_name || ' Not Due for a
raise');
END IF;
END LOOP;
END;

```

- d. Testen Sie den PL/SQL-Block mit folgenden Werten:

Department ID	Message
10	Whalen Due for a raise
20	Hartstein Not Due for a raise Fay Not Due for a raise
50	Weiss Not Due for a raise Fripp Not Due for a raise Kaufling Not Due for a raise Vollman Not Due for a raise. OConnell Due for a raise Grant Due for a raise
80	Russell Not Due for a raise Partners Not Due for a raise Errazuriz Not Due for a raise Cambrault Not Due for a raise . . . Livingston Not Due for a raise Johnson Not Due for a raise

3. Erstellen Sie einen PL/SQL-Block, der Cursor mit Parametern deklariert und verwendet. Um aus der departments-Tabelle die Nummer und den Namen einer Abteilung abzurufen, deren department_id kleiner als 100 ist, verwenden Sie einen Cursor in einer Schleife. Um aus der employees-Tabelle die Details zu Nachname, Tätigkeit, Einstellungsdatum und Gehalt der Mitarbeiter abzurufen, die in einer Abteilung mit einer employee_id unter 120 arbeiten, übergeben Sie die Abteilungsnummer als Parameter an einen anderen Cursor.

- a. Um für die Abteilungen mit einer department_id unter 100 die department_id und den department_name abzurufen, deklarieren Sie im deklarativen Bereich einen dept_cursor-Cursor. Sortieren Sie nach department_id.

```
DECLARE
  CURSOR c_dept_cursor IS
    SELECT department_id, department_name
    FROM departments
    WHERE department_id < 100
    ORDER BY department_id;
```

- b. Deklarieren Sie einen weiteren Cursor c_emp_cursor, der die Abteilungsnummer als Parameter annimmt und last_name, job_id, hire_date und salary der Abteilungsmitarbeiter abruft, deren employee_id kleiner als 120 ist.

```
CURSOR c_emp_cursor(v_deptno NUMBER) IS
  SELECT last_name, job_id, hire_date, salary
  FROM employees
  WHERE department_id = v_deptno
  AND employee_id < 120;
```

- c. Deklarieren Sie Variablen, die von den einzelnen Cursorn abgerufenen Werte aufnehmen. Deklarieren Sie Variablen mit dem %TYPE-Attribut.

```
v_current_deptno departments.department_id%TYPE;
v_current_dname departments.department_name%TYPE;
v_ename employees.last_name%TYPE;
v_job employees.job_id%TYPE;
v_hiredate employees.hire_date%TYPE;
v_sal employees.salary%TYPE;
```

- d. Öffnen Sie c_dept_cursor, verwenden Sie eine einfache Schleife, und lesen Sie Werte in die deklarierten Variablen ein. Zeigen Sie die Abteilungsnummer und den Abteilungsnamen an.

```
BEGIN
  OPEN c_dept_cursor;
  LOOP
    FETCH c_dept_cursor INTO v_current_deptno, v_current_dname;
    EXIT WHEN c_dept_cursor%NOTFOUND;
    DBMS_OUTPUT.PUT_LINE ('Department Number: ' ||
  v_current_deptno || ' Department Name : ' || v_current_dname);
```

- e. Öffnen Sie c_emp_cursor für jede Abteilung, indem Sie die aktuelle Abteilungsnummer als Parameter übergeben. Starten Sie eine weitere Schleife, und lesen Sie die Werte von emp_cursor in Variablen ein. Geben Sie alle aus der employees-Tabelle abgerufenen Details aus.

Hinweis: Sie können nach den Details zu den einzelnen Abteilungen eine Zeile ausgeben. Verwenden Sie geeignete Attribute für die exit-Bedingung. Prüfen Sie außerdem vor dem Öffnen des Cursors, ob bereits ein Cursor geöffnet ist.

```
IF c_emp_cursor%ISOPEN THEN
  CLOSE c_emp_cursor;
END IF;
OPEN c_emp_cursor (v_current_deptno);
LOOP
  FETCH c_emp_cursor INTO v_ename,v_job,v_hiredate,v_sal;
  EXIT WHEN c_emp_cursor%NOTFOUND;
  DBMS_OUTPUT.PUT_LINE (v_ename || ' ' || v_job
                        || ' ' || v_hiredate || ' ' || v_sal);
END LOOP;
DBMS_OUTPUT.PUT_LINE ('-----');
CLOSE c_emp_cursor;
```

- f. Schließen Sie alle Schleifen und Cursor, und beenden Sie den ausführbaren Bereich.
Führen Sie das Skript aus.

```
END LOOP;
CLOSE c_dept_cursor;
END;
```

Ausgabe:

```
anonymous block completed
Department Number : 10 Department Name : Administration
-----
Department Number : 20 Department Name : Marketing
-----
Department Number : 30 Department Name : Purchasing
Raphaely    PU_MAN    07-DEC-94    11000
Khoo       PU_CLERK   18-MAY-95    3100
Baida      PU_CLERK   24-DEC-97    2900
Tobias     PU_CLERK   24-JUL-97    2800
Himuro     PU_CLERK   15-NOV-98    2600
Colmenares PU_CLERK   10-AUG-99    2500
-----
Department Number : 40 Department Name : Human Resources
-----
Department Number : 50 Department Name : Shipping
-----
Department Number : 60 Department Name : IT
Hunold     IT_PROG    03-JAN-90    9000
Ernst      IT_PROG    21-MAY-91    6000
Austin     IT_PROG    25-JUN-97    4800
Pataballa  IT_PROG    05-FEB-98    4800
Lorentz    IT_PROG    07-FEB-99    4200
-----
Department Number : 70 Department Name : Public Relations
-----
Department Number : 80 Department Name : Sales
-----
Department Number : 90 Department Name : Executive
King       AD_PRES    17-JUN-87    24000
Kochhar    AD_VP      21-SEP-89    17000
De Haan    AD_VP      13-JAN-93    17000
```

Übungen zu Lektion 8 – Exceptions behandeln

1. In dieser Übung wird die Verwendung vordefinierter Exceptions erläutert. Um den Namen des Mitarbeiters mit einem gegebenen Gehaltswert auszuwählen, erstellen Sie einen PL/SQL-Block.

- a. Löschen Sie alle Datensätze in der messages-Tabelle.

```
DELETE FROM MESSAGES;  
SET VERIFY OFF
```

- b. Deklarieren Sie im deklarativen Bereich zwei Variablen: v_ename des Typs employees.last_name und v_emp_sal des employees.salary-Typs. Initialisieren Sie letztere Variable mit 6000.

```
DECLARE  
  v_ename  employees.last_name%TYPE;  
  v_emp_sal    employees.salary%TYPE := 6000;
```

- c. Rufen Sie im ausführbaren Bereich die Nachnamen der Mitarbeiter ab, deren Gehalt dem Wert in v_emp_sal entspricht.

Hinweis: Verwenden Sie keine expliziten Cursor.

Wenn das eingegebene Gehalt nur eine Zeile zurückgibt, geben Sie in die messages-Tabelle den Namen des Mitarbeiters und die Höhe seines Gehalts ein.

```
BEGIN  
  SELECT last_name  
    INTO   v_ename  
   FROM   employees  
  WHERE   salary = v_emp_sal;  
  INSERT INTO messages (results)  
VALUES (v_ename || ' - ' || v_emp_sal);
```

- d. Wenn das eingegebene Gehalt keine Zeilen zurückgibt, behandeln Sie die Exception mit dem entsprechenden Exception-Handler, und fügen Sie in die messages-Tabelle die Meldung "No employee with a salary of <salary>" ein.

```
EXCEPTION  
  WHEN no_data_found THEN  
    INSERT INTO messages (results)  
VALUES ('No employee with a salary of '|| TO_CHAR(v_emp_sal));
```

- e. Wenn das eingegebene Gehalt mehrere Zeilen zurückgibt, behandeln Sie die Exception mit dem entsprechenden Exception Handler, und fügen Sie in die messages-Tabelle die Meldung "More than one employee with a salary of <salary>" ein.

```
WHEN too_many_rows THEN  
  INSERT INTO messages (results)  
VALUES ('More than one employee with a salary of '||  
      TO_CHAR(v_emp_sal));
```

- f. Behandeln Sie beliebige weitere Exceptions mit einem entsprechenden Exception Handler, und fügen Sie in die messages-Tabelle die Meldung "Some other error occurred" ein.

```

WHEN others THEN
  INSERT INTO messages (results)
    VALUES ('Some other error occurred.');
END;

```

- g. Um zu prüfen, ob der PL/SQL-Block erfolgreich ausgeführt wurde, zeigen Sie die Zeilen aus der messages-Tabelle an. Die Ausgabe kann beispielsweise wie folgt lauten:

```

/
SELECT * FROM messages;

```

RESULTS
----- More than one employee with a salary of 6000

2. In dieser Übung wird gezeigt, wie Sie Exceptions mit einem Standardfehler des Oracle-Servers deklarieren. Verwenden Sie den Oracle-Server-Fehler ORA-02292 (integrity constraint violated - child record found).
- Deklarieren Sie im deklarativen Bereich die Exception e_childrecord_exists. Ordnen Sie dem Standardfehler -02292 des Oracle-Servers die deklarierte Exception zu.

```

DECLARE
  e_childrecord_exists EXCEPTION;
  PRAGMA EXCEPTION_INIT(e_childrecord_exists, -02292);

```

- Zeigen Sie im ausführbaren Bereich "Deleting department 40..." an. Um die Abteilung mit der department_id 40 zu löschen, nehmen Sie eine DELETE-Anweisung auf.

```

BEGIN
  DBMS_OUTPUT.PUT_LINE(' Deleting department 40.....');
  delete from departments where department_id=40;

```

- Um die Exception e_childrecord_exists zu behandeln und die entsprechende Meldung anzuzeigen, nehmen Sie einen Exception-Abschnitt auf. Die Ausgabe kann beispielsweise wie folgt lauten:

```

EXCEPTION
  WHEN e_childrecord_exists THEN
    DBMS_OUTPUT.PUT_LINE(' Cannot delete this department. There are
employees in this department (child records exist.) ');
END;

```

```
anonymous block completed
Deleting department 40.....
Cannot delete this department.
There are employees in this department (child records exist.)
```

Oracle Internal & Oracle Academy
Use Only

Übungen zu Lektion 9 – Stored Procedures und Stored Functions erstellen

1. Laden Sie das Skript lab_02_04_soln.sql, das Sie in der 4. Übung der 2. Lektion erstellt haben.
 - a. Bearbeiten Sie das Skript, um den anonymen Block in eine Prozedur namens greet zu konvertieren.

```
CREATE PROCEDURE greet IS
    today DATE:=SYSDATE;
    tomorrow today%TYPE;
...
```

- b. Führen Sie das Skript aus, um die Prozedur zu erstellen.
 - c. Speichern Sie das Skript unter dem Namen lab_09_01_soln.sql.
 - d. Um den Workspace zu leeren, klicken Sie auf die Schaltfläche **Clear**.
 - e. Um die greet-Prozedur aufzurufen, erstellen Sie einen anonymen Block und führen ihn aus. Die Ausgabe kann beispielsweise wie folgt lauten:

```
BEGIN
    greet;
END;
```

```
anonymous block completed
Hello World
TODAY IS : 30-MAY-07
TOMORROW IS : 31-MAY-07
```

2. Laden Sie das Skript lab_09_01_soln.sql.
 - a. Löschen Sie die greet-Prozedur mit dem folgenden Befehl:

```
DROP PROCEDURE greet
```

- b. Bearbeiten Sie die Prozedur so, dass sie ein Argument des VARCHAR2-Typs annimmt. Nennen Sie das Argument p_name.

```
CREATE PROCEDURE greet(p_name VARCHAR2) IS
    today DATE:=SYSDATE;
    tomorrow today%TYPE;
```

- c. Geben Sie statt Hello <name> die Zeichenfolge Hello World aus.

```
BEGIN
    tomorrow:=today +1;
    DBMS_OUTPUT.PUT_LINE(' Hello'|| p_name);
```

- d. Speichern Sie Ihr Skript unter dem Namen lab_09_02_soln.sql.
- e. Führen Sie das Skript aus, um die Prozedur zu erstellen.
- f. Um die greet-Prozedur mit einem Parameter aufzurufen, erstellen Sie einen anonymen Block und führen ihn aus. Die Ausgabe kann beispielsweise wie folgt lauten:

```
BEGIN
  greet('Neema');
END;
```

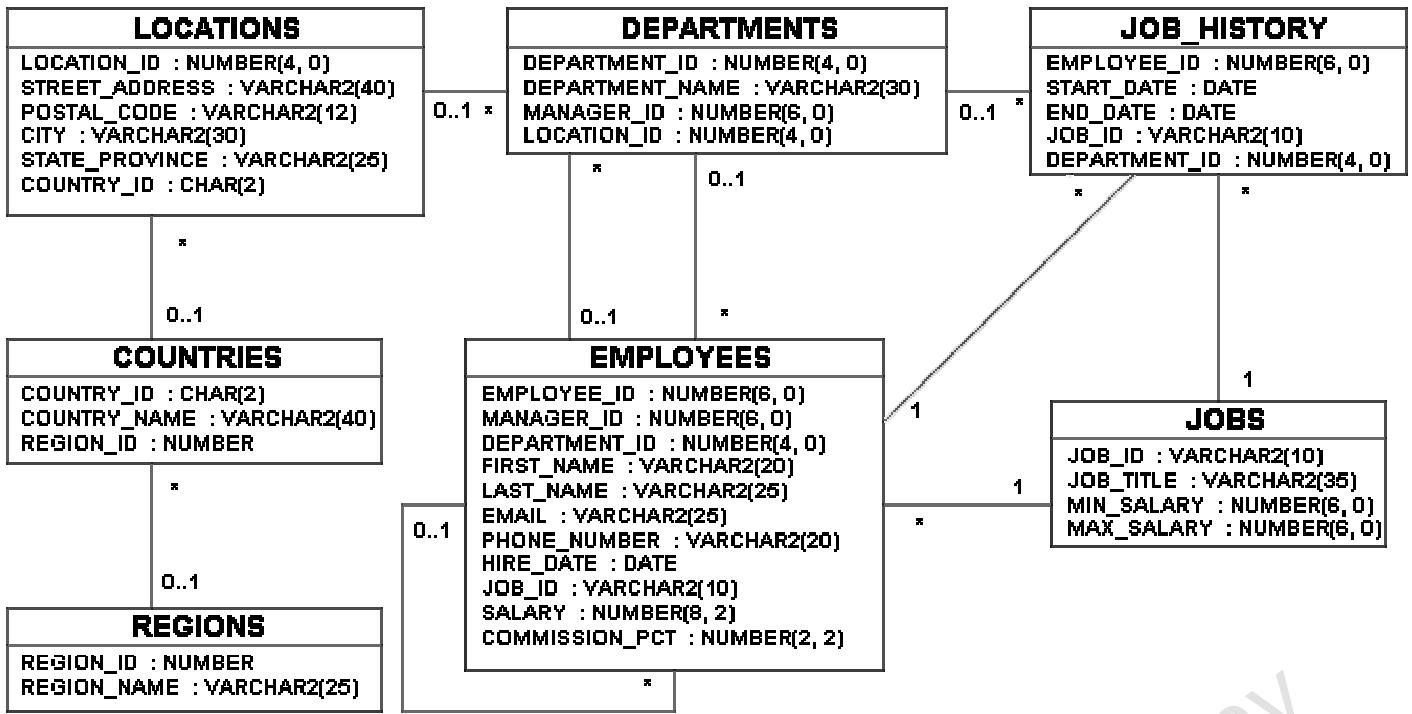
```
anonymous block completed
Hello Neema
TODAY IS : 30-MAY-07
TOMORROW IS : 31-MAY-07
```

B

Tabellenbeschreibungen und -daten

Oracle Internal & Oracle Academy
Use Only

ENTITY RELATIONSHIP-DIAGRAMM



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.

regions-Tabelle

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

countries-Tabelle

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.

locations-Tabelle

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	Y5W 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	80926	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.

departments-Tabelle

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.

jobs-Tabelle

```
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.

employees-Tabelle

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)

Oracle Internal & Oracle Academy
Use Only

employees-Tabelle (Fortsetzung)

Als Überschriften der Spalten commission_pct, manager_id und department_id wurde in der folgenden Abbildung jeweils comm, mgrid und deptid festgelegt, damit die 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
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	25-JUN-97	IT_PROG	4800		103	60
106	Valli	Pataballa	VPATABAL	590.423.4560	06-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	Sciarrা	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

employees-Tabelle (Fortsetzung)

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	650.124.8234	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	PHILHTAN	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	OTUVAU	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	011.44.1345.729268	10-MAR-97	SA REP	8000	.3	146	80
160	Louise	Doran	LDORAN	011.44.1345.629268	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

employees-Tabelle (Fortsetzung)

EMPLOYEE_ID	FIRST_NAME	LAST_NAME	EMAIL	PHONE_NUMBER	HIRE_DATE	JOB_ID	SALARY	comm	mgrid	deptid
170	Taylor	Fox	TFOX	011.44.1343.729268	24-JAN-98	SA_REP	9600	.2	148	80
171	William	Smith	WSMITH	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	JLIVINGSTON	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	Fleaur	JFLEAUR	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	GGEONI	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-96	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.

job_history-Tabelle

```
DESCRIBE job_history
```

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

```
SELECT * FROM job_history;
```

EMPLOYEE_ID	START_DATE	END_DATE	JOB_ID	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.



ORACLE

Copyright © 2007, Oracle. Alle Rechte vorbehalten.

Oracle Internal & Oracle Academy
Use Only

Ziele

Nach Beendigung dieses Anhangs haben Sie folgende Ziele erreicht:

- **Wichtige Features von Oracle SQL Developer auflisten**
- **Oracle SQL Developer installieren**
- **Menüoptionen von Oracle SQL Developer angeben**
- **Datenbankverbindungen erstellen**
- **Datenbankobjekte verwalten**
- **SQL Worksheet verwenden**
- **SQL-Anweisungen und SQL-Skripts ausführen**
- **PL/SQL-Anweisungen bearbeiten und debuggen**
- **Berichte erstellen und speichern**

ORACLE®

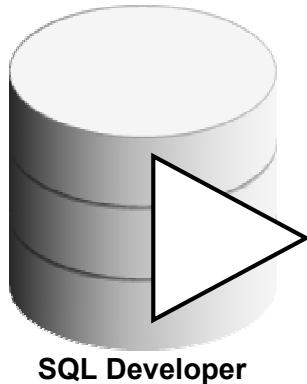
Copyright © 2007, Oracle. Alle Rechte vorbehalten.

Ziele

In diesem Anhang wird das grafische Tool SQL Developer eingeführt, mit dem Sie Datenbankentwicklungsaufgaben vereinfachen können. Sie lernen, wie Sie SQL-Anweisungen und SQL-Skripts mit SQL Worksheet ausführen. Außerdem lernen Sie, wie Sie PL/SQL bearbeiten und debuggen.

Oracle SQL Developer

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



ORACLE®

Copyright © 2007, Oracle. Alle Rechte vorbehalten.

Oracle SQL Developer

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

Mit SQL Developer, dem grafischen Tool zur Datenbankentwicklung, werden folgende Aufgaben vereinfacht:

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

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

Wichtige Features

- **In Java entwickelt**
- **Unterstützt Windows-, Linux- und Mac OS X-Plattformen**
- **Default-Konnektivität durch JDBC-Thin-Treiber**
- **Erfordert kein Installationsprogramm**
- **Anmeldung bei allen Datenbanken der Oracle Database-Version 9.2.0.1 oder höher möglich**
- **Mit JRE 1.5 ausgeliefert**

ORACLE

Copyright © 2007, Oracle. Alle Rechte vorbehalten.

Wichtige Features

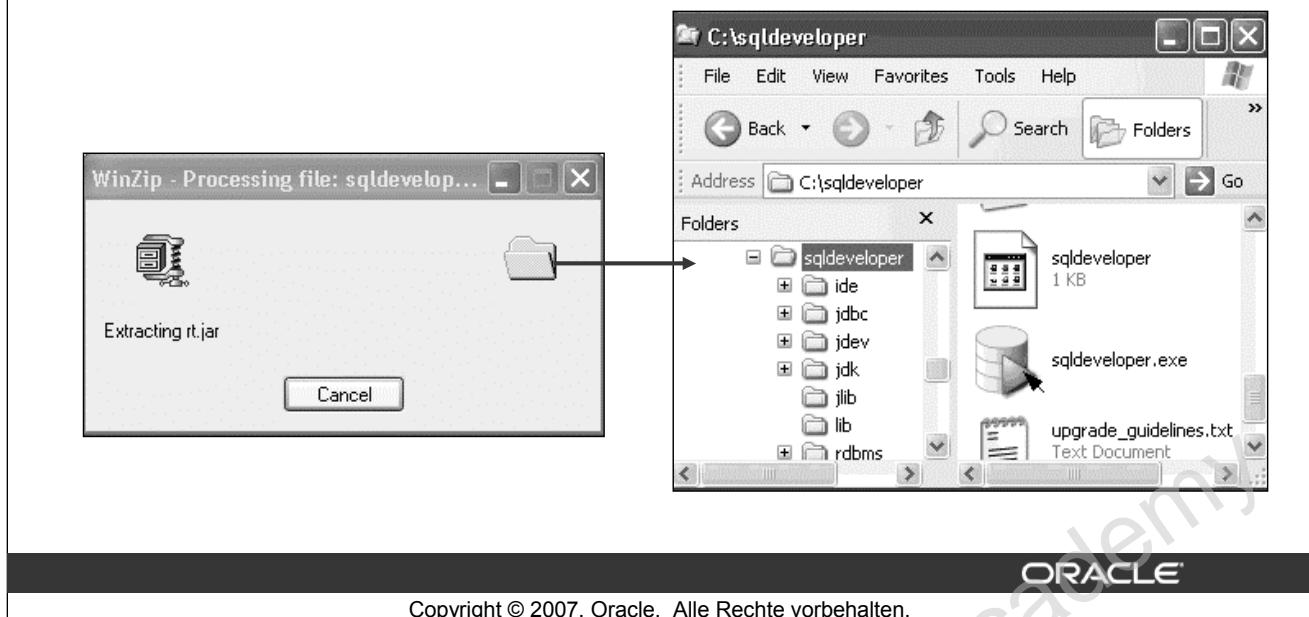
Oracle SQL Developer wurde in Java entwickelt, um die Oracle JDeveloper-IDE (Integrated Development Environment, integrierte Entwicklungsumgebung) zu nutzen. Das Tool kann auf Windows-, Linux- und Mac OS X-Plattformen ausgeführt werden. Sie können SQL Developer auf dem Datenbank-Server installieren und eine Remote-Vbindung von Ihrem Desktop aus herstellen. Auf diese Weise umgehen Sie den Verkehr im Client/Server-Netzwerk.

Die Konnektivität zur Datenbank implementiert standardmäßig der JDBC-(Java Database Connectivity-)Thin-Treiber, so dass 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 Datenbanken der Oracle Database-Version 9.2.0.1 oder höher sowie bei allen Oracle-Datenbankeditionen einschließlich der Express Edition anmelden. SQL Developer wird mit JRE (Java Runtime Environment) 1.5 einschließlich einer zusätzlichen tools.jar-Datei zur Unterstützung von Windows-Clients ausgeliefert. Clients, die keine Windows-Clients sind, benötigen nur JDK (Java Development Kit) 1.5.

SQL Developer installieren

Oracle SQL Developer Kit herunterladen und in einem beliebigen Verzeichnis auf dem Rechner dekomprimieren:



SQL Developer installieren

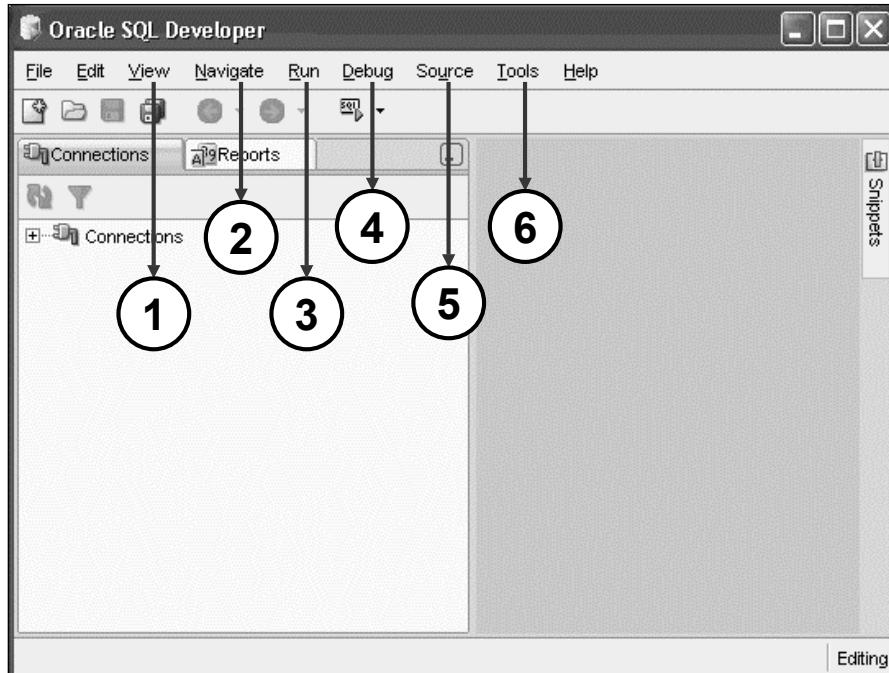
Für Oracle SQL Developer ist kein Installationsprogramm erforderlich. Um SQL Developer zu installieren, benötigen Sie ein Tool zum Dekomprimieren.

Gehen Sie wie folgt vor, um SQL Developer zu installieren:

1. Erstellen Sie auf dem lokalen Laufwerk den Ordner **<local drive>:\SQL Developer**.
2. Laden Sie das SQL Developer Kit unter folgender Adresse herunter:
<http://www.oracle.com/technology/software/products/sql/index.html>
3. Dekomprimieren Sie das heruntergeladene SQL Developer Kit in den Ordner, den Sie im 1. Schritt erstellt haben.

Um SQL Developer zu starten, wechseln Sie zu **<local drive>:\SQL Developer** und doppelklicken auf **sqldeveloper.exe**.

SQL Developer – Menüs



ORACLE®

Copyright © 2007, Oracle. Alle Rechte vorbehalten.

SQL Developer – Menüs

SQL Developer verfügt über zwei Hauptregisterkarten für die Navigation:

- **Connections Navigator:** In dieser Registerkarte können Sie Datenbankobjekte und Benutzer durchsuchen, auf die Sie Zugriff haben.
- **Registerkarte "Reports":** In dieser Registerkarte können Sie vordefinierte Berichte ausführen oder eigene Berichte erstellen und hinzufügen.

Die linke Seite in SQL Developer dient der Navigation zum Suchen und Auswählen von Objekten. Auf der rechten Seite werden Informationen über die ausgewählten Objekte angezeigt. Sie können viele Aspekte der Darstellung und des Verhaltens von SQL Developer anpassen, indem Sie Voreinstellungen festlegen.

Die Menüs im oberen Bildschirmteil enthalten Standardeinträge sowie zusätzliche Einträge für spezifische Features von SQL Developer.

1. **View:** Enthält Optionen, die bestimmen, was auf der Benutzeroberfläche von SQL Developer angezeigt wird
2. **Navigate:** Enthält Optionen für die Navigation zu anderen Bereichen und die Ausführung von Unterprogrammen
3. **Run:** Enthält die Optionen **Run File** und **Execution Profile**. Sie sind relevant, wenn Funktionen oder Prozeduren ausgewählt wurden.
4. **Debug:** Enthält Optionen, die für Funktionen oder Prozeduren relevant sind
5. **Source:** Enthält Optionen zur Bearbeitung von Funktionen und Prozeduren
6. **Tools:** Ruft SQL Developer-Tools wie SQL*Plus, Preferences und SQL Worksheet auf

Datenbankverbindungen erstellen

- Sie müssen über mindestens eine Datenbankverbindung verfügen, um SQL Developer verwenden zu können.
- Sie können Verbindungen erstellen und testen:
 - Für mehrere Datenbanken
 - Für mehrere Schemas
- SQL Developer importiert automatisch alle Verbindungen, die in der `tnsnames.ora`-Datei in Ihrem System definiert sind.
- Sie können Verbindungen in eine XML-Datei exportieren.
- Jede zusätzlich erstellte Datenbankverbindung wird in der Navigationshierarchie unter "Connections" aufgelistet.

ORACLE

Copyright © 2007, Oracle. Alle Rechte vorbehalten.

Datenbankverbindungen erstellen

Eine Verbindung ist ein SQL Developer-Objekt. Es gibt die Informationen an, die notwendig sind, um sich als bestimmter Benutzer bei einer bestimmten Datenbank anzumelden. Um SQL Developer zu verwenden, muss mindestens eine Datenbankverbindung vorhanden sein bzw. erstellt oder importiert werden.

Sie können Verbindungen für mehrere Datenbanken und mehrere Schemas erstellen und testen.

Die `tnsnames.ora`-Datei befindet sich standardmäßig im Verzeichnis `$ORACLE_HOME/network/admin`. Sie kann sich jedoch auch in einem Verzeichnis befinden, das durch die `TNS_ADMIN`-Umgebungsvariable oder den Registrierungswert angegeben wird.

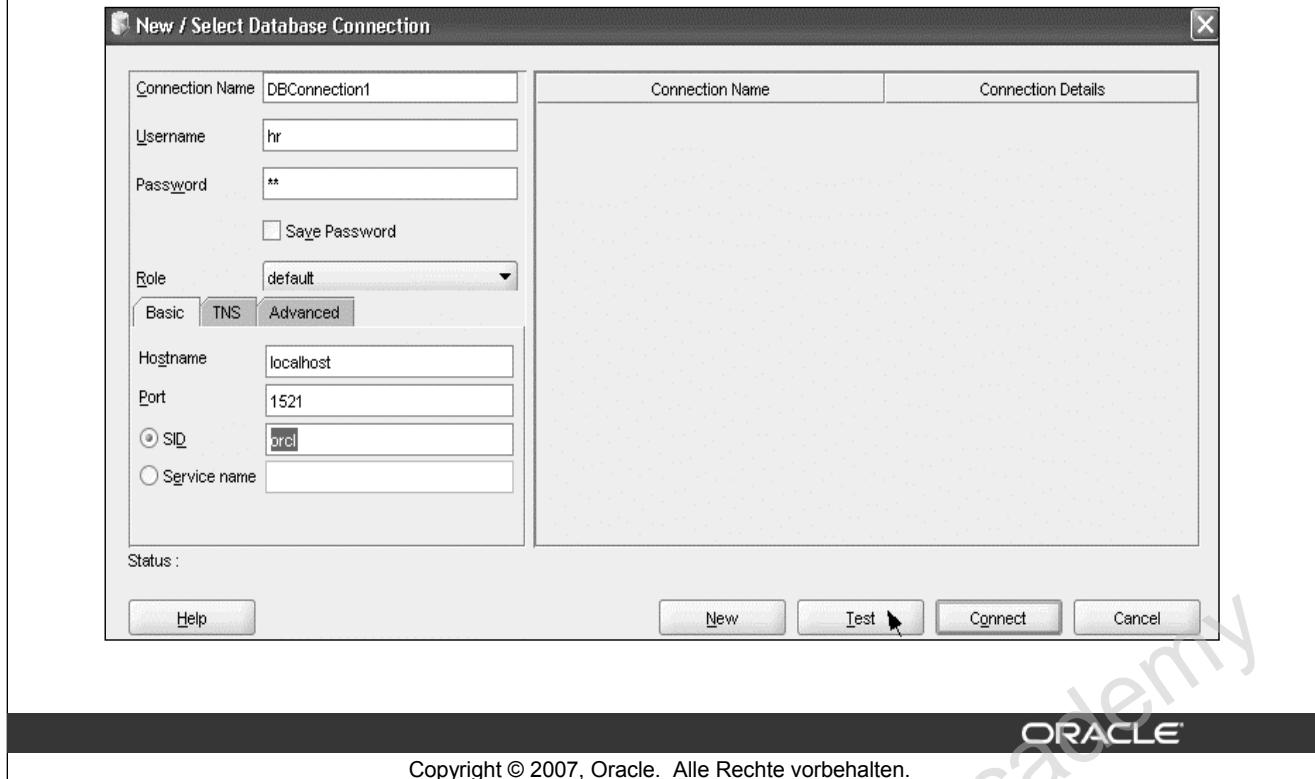
Wenn Sie SQL Developer starten und das Dialogfeld **Database Connections** anzeigen, importiert SQL Developer automatisch alle in der `tnsnames.ora`-Datei in Ihrem System definierten Verbindungen.

Hinweis: Wenn die in der `tnsnames.ora`-Datei definierten Verbindungen in Windows-Systemen 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 bei derselben Datenbank als verschiedene Benutzer anzumelden oder sich bei verschiedenen Datenbanken anzumelden.

Datenbankverbindungen erstellen



Datenbankverbindungen erstellen (Fortsetzung)

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

1. Doppelklicken Sie auf `<your_path>\sqldeveloper\sqldeveloper.exe`.
2. Klicken Sie in der Registerkarte **Connections** mit der rechten Maustaste auf **Connections**, und wählen Sie **New Database Connection**.
3. Geben Sie den Verbindungsnamen, den Benutzernamen, das Passwort, den Host-Namen und die SID für die Anmeldung bei der gewünschten Datenbank ein.
4. Um sicherzustellen, dass die Verbindung richtig eingerichtet wurde, klicken Sie auf **Test**.
5. Klicken Sie auf **Connect**.

Geben Sie in der Registerkarte **Basic** unten auf dem Bildschirm folgende Optionen an:

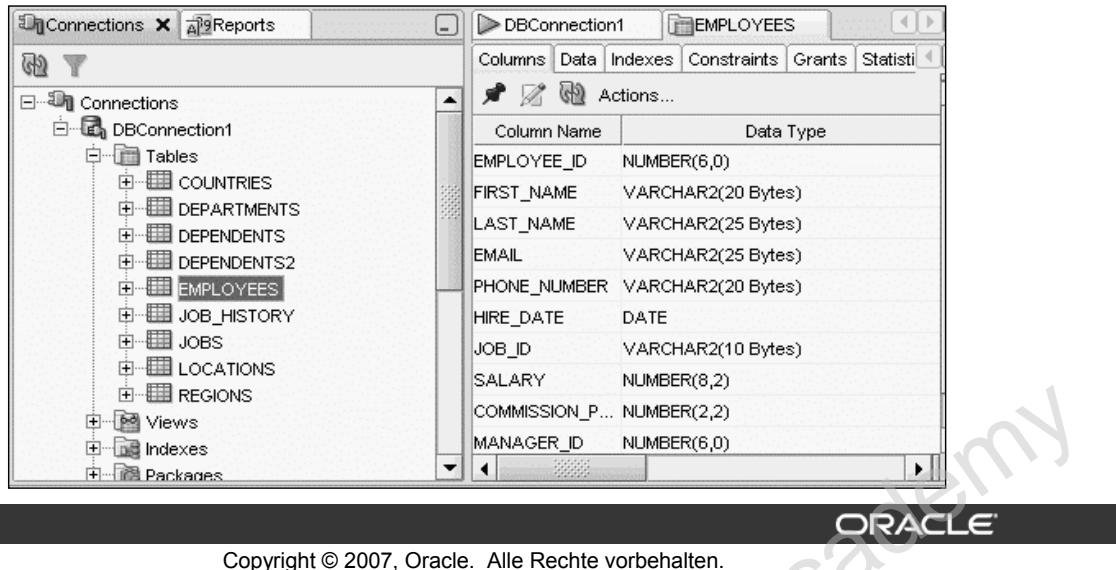
- **Hostname:** Host-System für die Oracle-Datenbank
- **Port:** Listener Port
- **SID:** Datenbankname
- **Service Name:** Netzwerk-Service-Name für eine Remote-Datenbankverbindung

Wenn Sie das Kontrollkästchen **Save Password** aktivieren, wird das Passwort in einer XML-Datei gespeichert. Wenn Sie dann die SQL Developer-Verbindung beenden und wieder öffnen, werden Sie nicht mehr aufgefordert, das Passwort einzugeben.

Datenbankobjekte durchsuchen

Mit dem Database Navigator können Sie:

- viele Objekte in einem Datenbankschema durchsuchen
- Definitionen von Objekten auf einen Blick prüfen



Copyright © 2007, Oracle. Alle Rechte vorbehalten.

Datenbankobjekte durchsuchen

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

Die linke Seite in SQL Developer dient der Navigation zum Suchen und Auswählen von Objekten. Auf der rechten Seite werden Informationen über die ausgewählten Objekte angezeigt. Sie können viele Aspekte der Darstellung von SQL Developer anpassen, indem Sie Voreinstellungen festlegen.

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

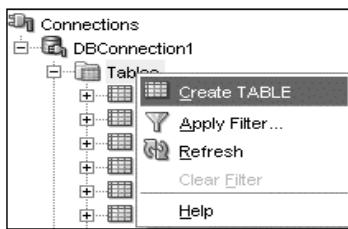
Um die Definition der EMPLOYEES-Tabelle wie auf der Folie anzuzeigen, gehen Sie wie folgt vor:

1. Blenden Sie im Connections Navigator den Knoten **Connections** ein.
2. Blenden Sie **Tables** ein.
3. Doppelklicken Sie auf **EMPLOYEES**.

Mit Hilfe der Registerkarte **Data** können Sie neue Zeilen eingeben, Daten aktualisieren und diese Änderungen in der Datenbank festschreiben.

Schemaobjekte erstellen

- **SQL Developer unterstützt die Erstellung beliebiger Schemaobjekte:**
 - SQL-Anweisungen können in SQL Worksheet ausgeführt werden.
 - Kontextmenüs sind verfügbar.
- Bearbeiten Sie Objekte mit Hilfe eines Dialogfeldes zum Bearbeiten oder mit einem der vielen Kontextmenüs.
- Zeigen Sie die DDL für Anpassungen wie das Erstellen eines neuen Objekts oder das Bearbeiten eines vorhandenen Schemaobjekts an.



ORACLE

Copyright © 2007, Oracle. Alle Rechte vorbehalten.

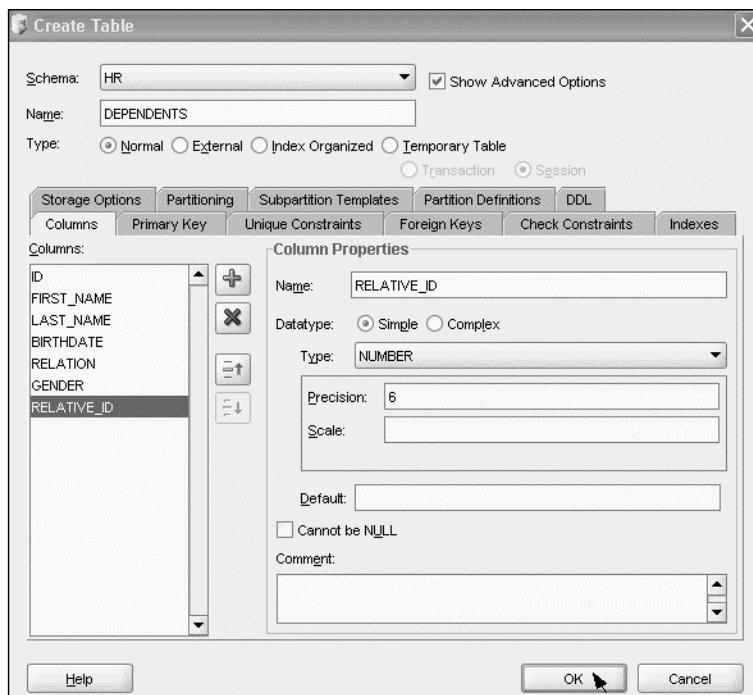
Schemaobjekte erstellen

SQL Developer unterstützt die Erstellung beliebiger Schemaobjekte durch die Ausführung einer SQL-Anweisung in SQL Worksheet. Alternativ können Sie Objekte mit Hilfe der Kontextmenüs erstellen. Nach ihrer Erstellung können Sie die Objekte mit Hilfe eines Dialogfeldes zum Bearbeiten oder mit einem der vielen Kontextmenüs bearbeiten.

Beim Erstellen neuer Objekte oder Bearbeiten vorhandener Objekte kann die DDL (Data Definition Language) für diese Anpassungen geprüft werden. Die Option **Export DDL** ist verfügbar, wenn Sie die vollständige DDL für ein oder mehrere Objekte im Schema erstellen möchten.

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

Neue Tabelle erstellen – Beispiel



ORACLE®

Copyright © 2007, Oracle. Alle Rechte vorbehalten.

Neue Tabelle erstellen – Beispiel

Wenn das Kontrollkästchen **Show Advanced Options** im Dialogfeld **Create Table** nicht aktiviert ist, können Sie Tabellen schnell erstellen, indem Sie Spalten und einige häufig verwendete Features angeben.

Bei aktiviertem Kontrollkästchen **Show Advanced Options** werden im Dialogfeld **Create Table** mehrere Registerkarten eingeblendet, in denen Sie beim Erstellen der Tabelle erweiterte Features angeben können.

Das Beispiel auf der Folie zeigt, wie die **DEPENDENTS**-Tabelle durch Auswahl des Kontrollkästchens **Show Advanced Options** erstellt wird.

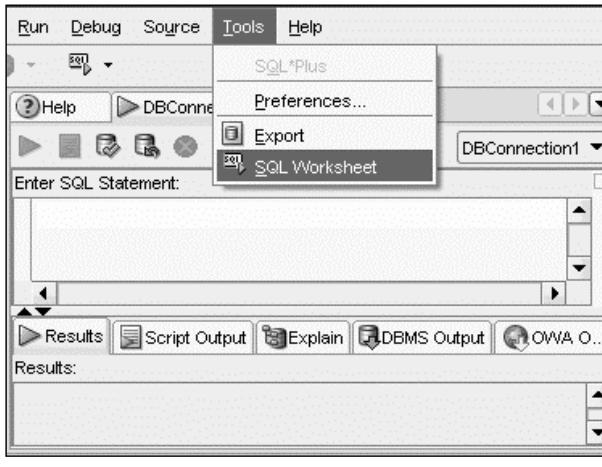
Um eine neue Tabelle zu erstellen, gehen Sie wie folgt vor:

1. Klicken Sie im Connections Navigator mit der rechten Maustaste auf **Tables**.
2. Wählen Sie **Create TABLE**.
3. Aktivieren Sie im Dialogfeld **Create Table** das Kontrollkästchen **Show Advanced Options**.
4. Geben Sie die Spalteninformationen an.
5. Klicken Sie auf **OK**.

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

SQL Worksheet

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



Copyright © 2007, Oracle. Alle Rechte vorbehalten.

SQL Worksheet

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

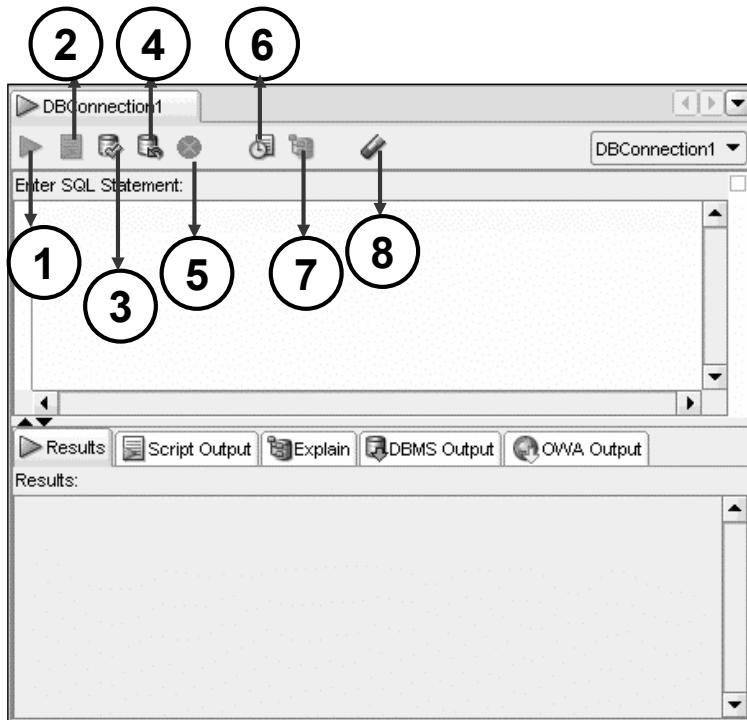
Sie können Aktionen angeben, die von der mit dem Worksheet verknüpften Datenbankverbindung verarbeitet werden können. Beispiel:

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

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

- Wählen Sie **Tools > SQL Worksheet**.
- Klicken Sie auf das Symbol **Open SQL Worksheet**.

SQL Worksheet



ORACLE®

Copyright © 2007, Oracle. Alle Rechte vorbehalten.

SQL Worksheet (Fortsetzung)

Sie können bestimmte Aufgaben auch über Tasturbefehle oder Symbole durchführen, zum Beispiel SQL-Anweisungen ausführen, Skripts ausführen und die Historie der bereits ausgeführten SQL-Anweisungen anzeigen. Die Symbolleiste von SQL Worksheet enthält Symbole zur Ausführung der folgenden Aufgaben:

1. **Anweisung ausführen:** Führt die Anweisung an der Cursor-Position im Feld **Enter SQL Statement** aus. Sie können Bind-Variablen, jedoch keine Austauschvariablen in den SQL-Anweisungen verwenden.
2. **Skript ausführen:** Führt mit Script Runner alle Anweisungen im Feld **Enter SQL Statement** aus. Sie können Austauschvariablen, jedoch keine Bind-Variablen in den SQL-Anweisungen verwenden.
3. **Festschreiben:** Schreibt alle Änderungen in der Datenbank fest und beendet die Transaktion.
4. **Rollback:** Verwirft alle Änderungen in der Datenbank, ohne sie festzuschreiben, und beendet die Transaktion.
5. **Abbrechen:** Bricht die Ausführung aller derzeit ausgeführten Anweisungen ab.
6. **SQL-Historie:** Zeigt ein Dialogfeld mit Informationen über bereits ausgeführte SQL-Anweisungen an.
7. **Explain-Plan ausführen:** Generiert den Ausführungsplan, den Sie durch Klicken auf die Registerkarte **Explain** anzeigen können.
8. **Löschen:** Löscht die Anweisungen im Feld **Enter SQL Statement**.

SQL-Anweisungen ausführen

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

The screenshot shows the Oracle SQL Worksheet window titled "DBConnection1". In the "Enter SQL Statement" pane, two SELECT statements are entered:

```
SELECT last_name, salary FROM employees  
WHERE salary > 10000;  
  
SELECT last_name "Name", salary*12 "Annual Salary"  
FROM employees;
```

Below the statements, the "Results" tab is selected, displaying the output:

LAST_NAME	SALARY
1 Hartstein	13000
2 Higgins	12000
3 King	24000

ORACLE®

Copyright © 2007, Oracle. Alle Rechte vorbehalten.

SQL-Anweisungen ausführen

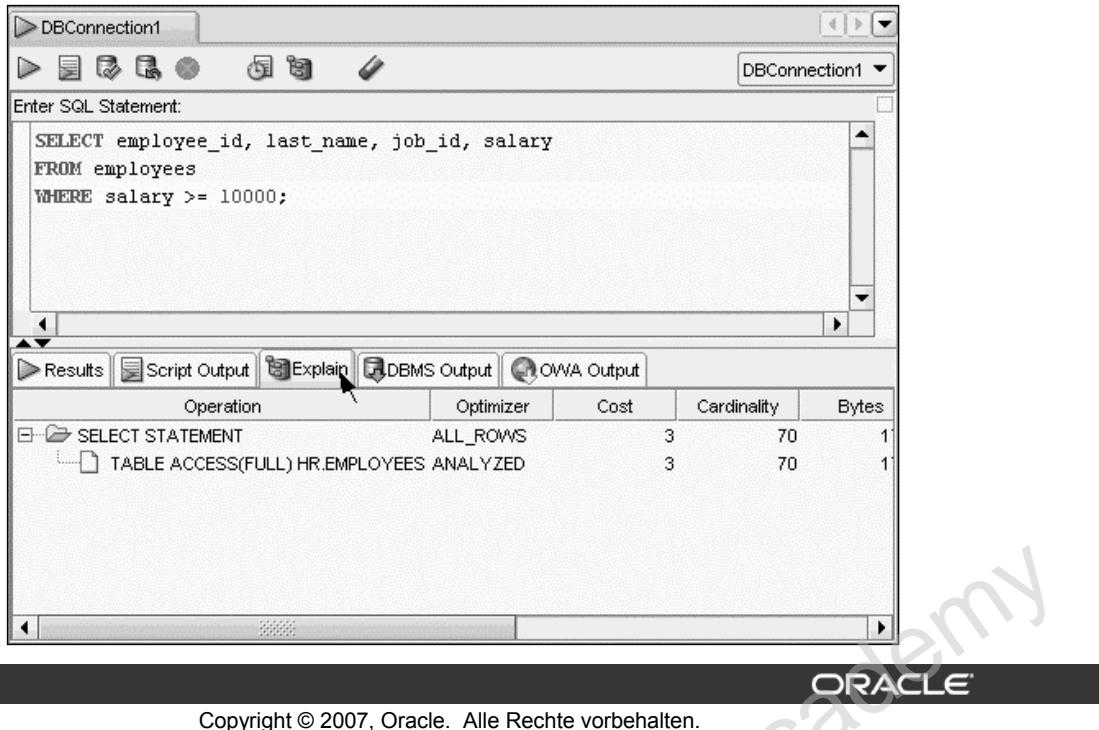
In SQL Worksheet können Sie im Bereich **Enter SQL Statement** eine einzelne oder mehrere SQL-Anweisungen eingeben. Wenn Sie nur eine Anweisung eingeben, ist das abschließende Semikolon optional.

Beim Eingeben der Anweisung werden die SQL-Schlüsselwörter automatisch hervorgehoben. Zum Ausführen einer SQL-Anweisung stellen Sie sicher, dass sich der Cursor innerhalb der Anweisung befindet, und klicken Sie auf das Symbol **Execute Statement**. Alternativ können Sie die Taste **F9** drücken.

Um mehrere SQL-Anweisungen auszuführen und die Ergebnisse anzuzeigen, klicken Sie auf das Symbol **Run Script**. Alternativ können Sie die Taste **F5** drücken.

Da das Beispiel auf der Folie mehrere SQL-Anweisungen enthält, endet die erste Anweisung mit einem Semikolon. Der Cursor befindet sich in der ersten Anweisung. Bei Ausführung der Anweisung werden im Feld **Results** nur die Ergebnisse der ersten Anweisung angezeigt.

Ausführungsplan anzeigen



Ausführungsplan anzeigen

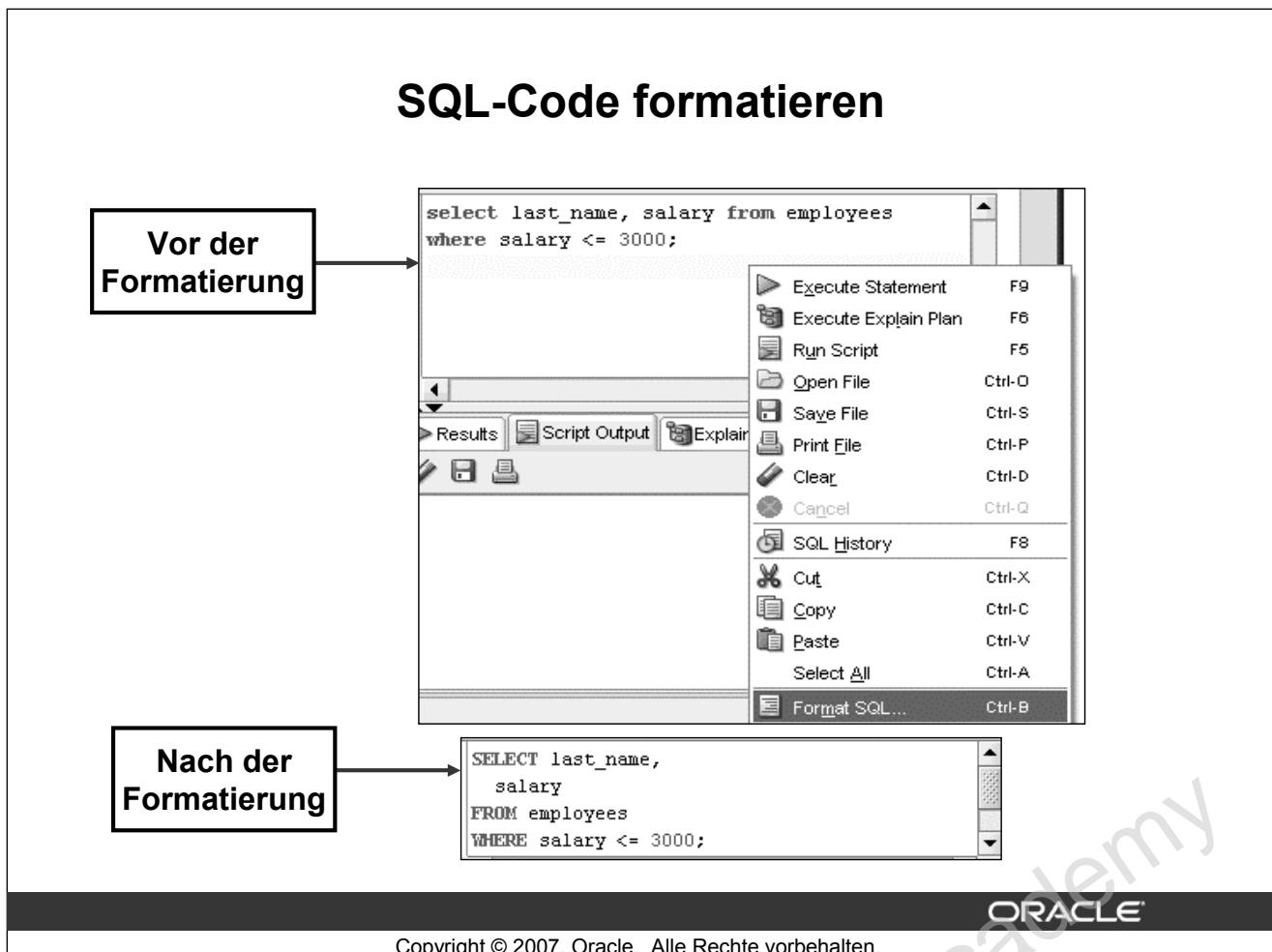
Sie können ein SQL-Skript ausführen und den Ausführungsplan anzeigen. Um eine SQL-Skriptdatei auszuführen, gehen Sie wie folgt vor:

1. Klicken Sie mit der rechten Maustaste im Bereich **Enter SQL Statement**, und wählen Sie im Dropdown-Menü die Option **Open File**.
2. Doppelklicken Sie im Dialogfeld **Open** auf die .sql-Datei.
3. Klicken Sie auf das Symbol **Run Script**.

Wenn Sie auf die .sql-Datei doppelklicken, werden die SQL-Anweisungen in das Feld **Enter SQL Statement** geladen. Sie können das Skript oder jede Zeile einzeln ausführen. Die Ergebnisse werden im Bereich **Script Output** angezeigt.

Im Beispiel auf der Folie wird der Ausführungsplan gezeigt. Mit dem Symbol **Execute Explain Plan** wird der Ausführungsplan generiert. Im Ausführungsplan wird die Folge der Operationen angegeben, mit denen die Anweisung ausgeführt wird. Sie können den Ausführungsplan anzeigen, indem Sie auf die Registerkarte **Explain** klicken.

SQL-Code formatieren



SQL-Code formatieren

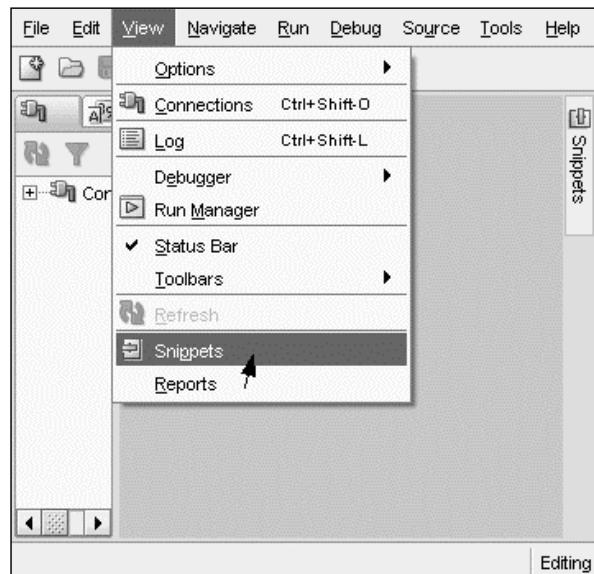
Es kann sinnvoll sein, die Einrückungen, Zeichenabstände, Groß-/Kleinschreibung und Zeilenabstände in SQL-Code übersichtlicher zu gestalten. SQL Developer bietet dafür ein Feature zur Formatierung von SQL-Code.

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

Aus dem Beispiel auf der Folie geht hervor, dass die Schlüsselwörter im SQL-Code vor der Formatierung nicht in Großbuchstaben dargestellt werden und die Anweisung nicht die richtigen Einrückungen aufweist. Nach der Formatierung ist der SQL-Code übersichtlicher, da die Schlüsselwörter groß geschrieben sind und die Anweisung richtig eingerückt ist.

Code-Auszüge

Enthalten nur Syntax oder Beispiele



ORACLE®

Copyright © 2007, Oracle. Alle Rechte vorbehalten.

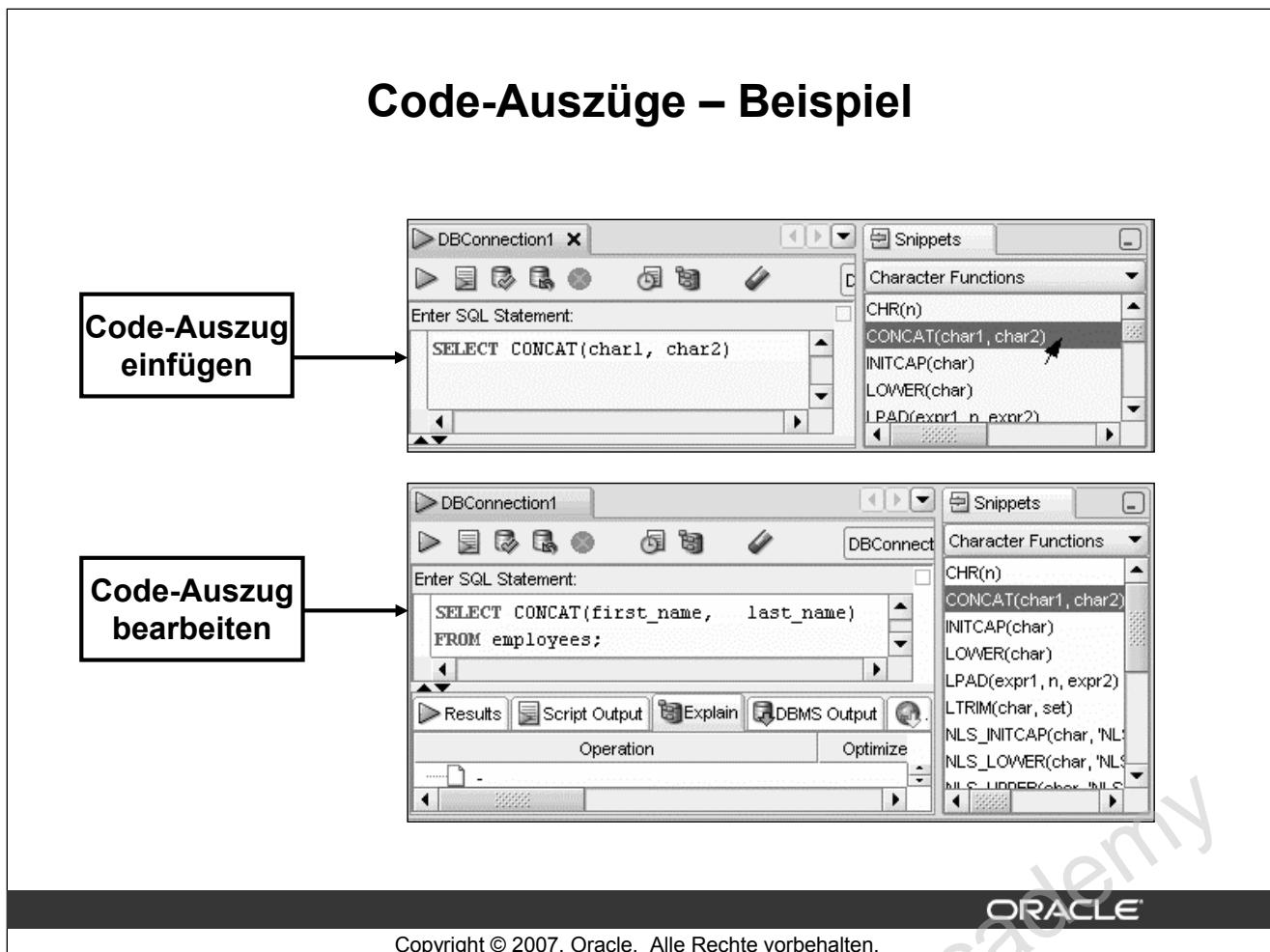
Code-Auszüge

Sie können bestimmte Code-Auszüge verwenden, wenn Sie mit SQL Worksheet arbeiten oder eine PL/SQL-Funktion oder -Prozedur erstellen oder bearbeiten. SQL Developer verfügt über ein Feature namens Snippets. Snippets sind Code-Auszüge wie SQL-Funktionen, Optimizer Hints und verschiedene PL/SQL-Programmiertechniken. Sie können Snippets in das Editor-Fenster ziehen.

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

Das Fenster **Snippets** wird auf der rechten Seite angezeigt. Verwenden Sie die Dropdown-Liste, um eine Gruppe auszuwählen. Über die Schaltfläche **Snippets** am rechten Fensterrand können Sie das Fenster **Snippets** einblenden.

Code-Auszüge – Beispiel



Code-Auszüge – Beispiel

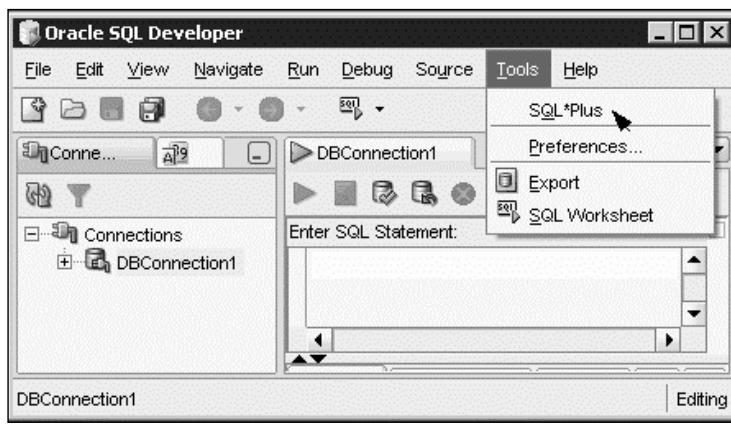
Um einen Code-Auszug in den Code in SQL Worksheet oder eine PL/SQL-Funktion oder -Prozedur einzufügen, ziehen Sie ihn aus dem Fenster **Snippets** an die gewünschte Stelle im Code. Danach können Sie die Syntax bearbeiten, so dass die SQL-Funktion im aktuellen Kontext gültig ist. Um eine kurze Beschreibung einer SQL-Funktion als QuickInfo anzeigen, platzieren Sie den Mauszeiger über den Funktionsnamen.

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

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

SQL*Plus

- **SQL Worksheet unterstützt nicht alle SQL*Plus-Anweisungen.**
- **Die SQL*Plus-Befehlszeilenschnittstelle kann über SQL Developer aufgerufen werden.**



ORACLE®

Copyright © 2007, Oracle. Alle Rechte vorbehalten.

SQL*Plus

SQL Worksheet unterstützt einige SQL*Plus-Anweisungen. SQL*Plus-Anweisungen müssen von SQL Worksheet interpretiert werden, bevor sie an die Datenbank übergeben werden. Nicht von SQL Worksheet unterstützte SQL*Plus-Anweisungen werden ignoriert und nicht an die Datenbank übergeben. Zu den SQL*Plus-Anweisungen, die nicht von SQL Worksheet unterstützt werden, gehören beispielsweise:

- append
- archive
- attribute
- break

Eine vollständige Liste der von SQL Worksheet unterstützten und nicht unterstützten SQL*Plus-Anweisungen finden Sie in der Online-Hilfe von SQL Developer.

Um das SQL*Plus-Befehlsfenster anzuzeigen, wählen Sie im Menü **Tools** die Option **SQL*Plus**. Für die Verwendung dieses Features muss das System, auf dem Sie SQL Developer verwenden, über ein Oracle-Home-Verzeichnis bzw. einen Oracle-Home-Ordner verfügen, in dem sich ein ausführbares SQL*Plus-Programm befindet. Wenn der Speicherort des ausführbaren SQL*Plus-Programms nicht bereits in Ihren Voreinstellungen für SQL Developer enthalten ist, werden Sie aufgefordert, den Speicherort anzugeben.

Anonyme Blöcke erstellen

Anonyme Blöcke erstellen und Ausgabe der Anweisungen des DBMS_OUTPUT Packages anzeigen

The screenshot shows the Oracle SQL Developer interface. In the main window, there is a code editor with the following PL/SQL code:

```
DECLARE
  Emp_name VARCHAR2(10);
  Cursor c1 IS SELECT last_name FROM Employees
  WHERE department_id = 20;
BEGIN
  OPEN c1;
  LOOP
    FETCH c1 INTO Emp_name;
    EXIT WHEN c1%NOTFOUND;
    DBMS_OUTPUT.PUT_LINE(Emp_name);
  END LOOP;
END.
```

Below the code editor, there is a toolbar with several icons. One of the icons is highlighted, labeled "DBMS Output". Underneath the toolbar, there is a status bar showing "Buffer Size: 20000".

In the bottom panel, there is a "DBMS Output" tab which contains the following text:

```
Hartstein
Fay
```

ORACLE

Copyright © 2007, Oracle. Alle Rechte vorbehalten.

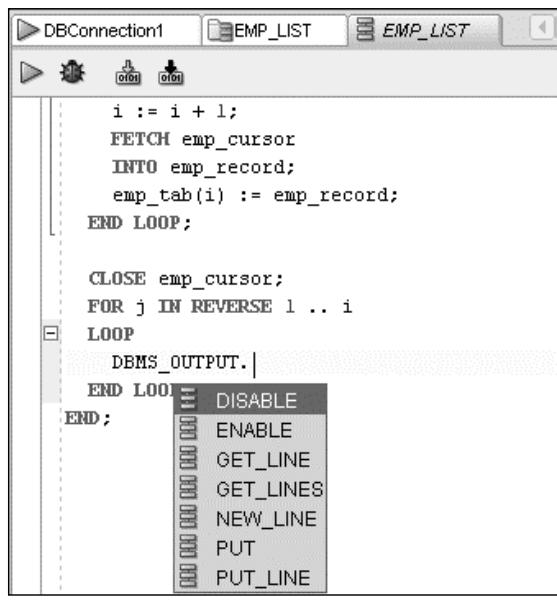
Anonyme Blöcke erstellen

Sie können einen anonymen Block erstellen und die Ausgabe der Anweisungen des DBMS_OUTPUT Packages anzeigen. Gehen Sie hierfür wie folgt vor:

1. Geben Sie im Feld **Enter SQL Statement** den PL/SQL-Code ein.
2. Klicken Sie auf den Bereich **DBMS Output**. Um die Server-Ausgabe zu aktivieren, klicken Sie auf das Symbol **Enable DBMS Output**.
3. Klicken Sie oberhalb des Feldes **Enter SQL Statement** auf das Symbol **Execute Statement**. Um die Ergebnisse anzuzeigen, klicken Sie anschließend auf den Bereich **DBMS Output**.

PL/SQL-Code bearbeiten

Editor mit umfangreicher Funktionalität für PL/SQL-Programmeinheiten verwenden:



ORACLE

Copyright © 2007, Oracle. Alle Rechte vorbehalten.

PL/SQL-Code bearbeiten

Sie können den PL/SQL-Code ändern. SQL Developer verfügt über einen Editor für PL/SQL-Programmeinheiten mit umfangreicher Funktionalität. Er ermöglicht die anpassbare Hervorhebung von PL/SQL-Syntax und umfasst gängige Editor-Funktionen wie:

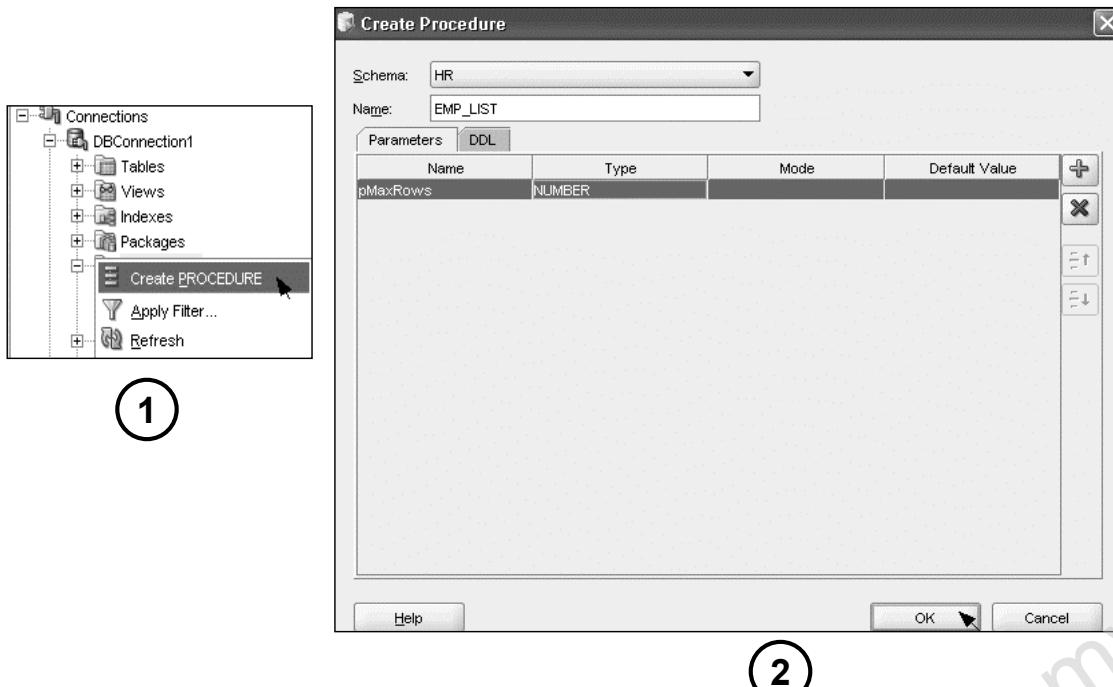
- Lesezeichen
- Code vervollständigen
- Code Folding
- Suchen und ersetzen

Um den PL/SQL-Code zu bearbeiten, klicken Sie im Connections Navigator auf den Objektnamen und dann auf das Symbol **Edit**. Alternativ können Sie auf den Objektnamen doppelklicken, um die Seite **Object Definition** mit den entsprechenden Registerkarten und die Seite **Edit** aufzurufen. Sie können nur in der Registerkarte **Edit** Aktualisierungen vornehmen.

Auf der Folie wird das Code Insight Feature gezeigt. Wenn Sie beispielsweise DBMS_OUTPUT. eingeben und die Tastenkombination STRG + LEERTASTE gedrückt halten, können Sie aus einer Member-Liste dieses Packages wählen. Beachten Sie, dass Code Insight standardmäßig automatisch aufgerufen wird, wenn Sie nach der Eingabe eines Punktes (.) länger als eine Sekunde warten.

Wenn Sie PL/SQL-Code mit dem Code-Editor bearbeiten, können Sie zwischen **Compile** und **Compile for Debug** wählen.

PL/SQL-Prozeduren erstellen



PL/SQL-Prozeduren erstellen

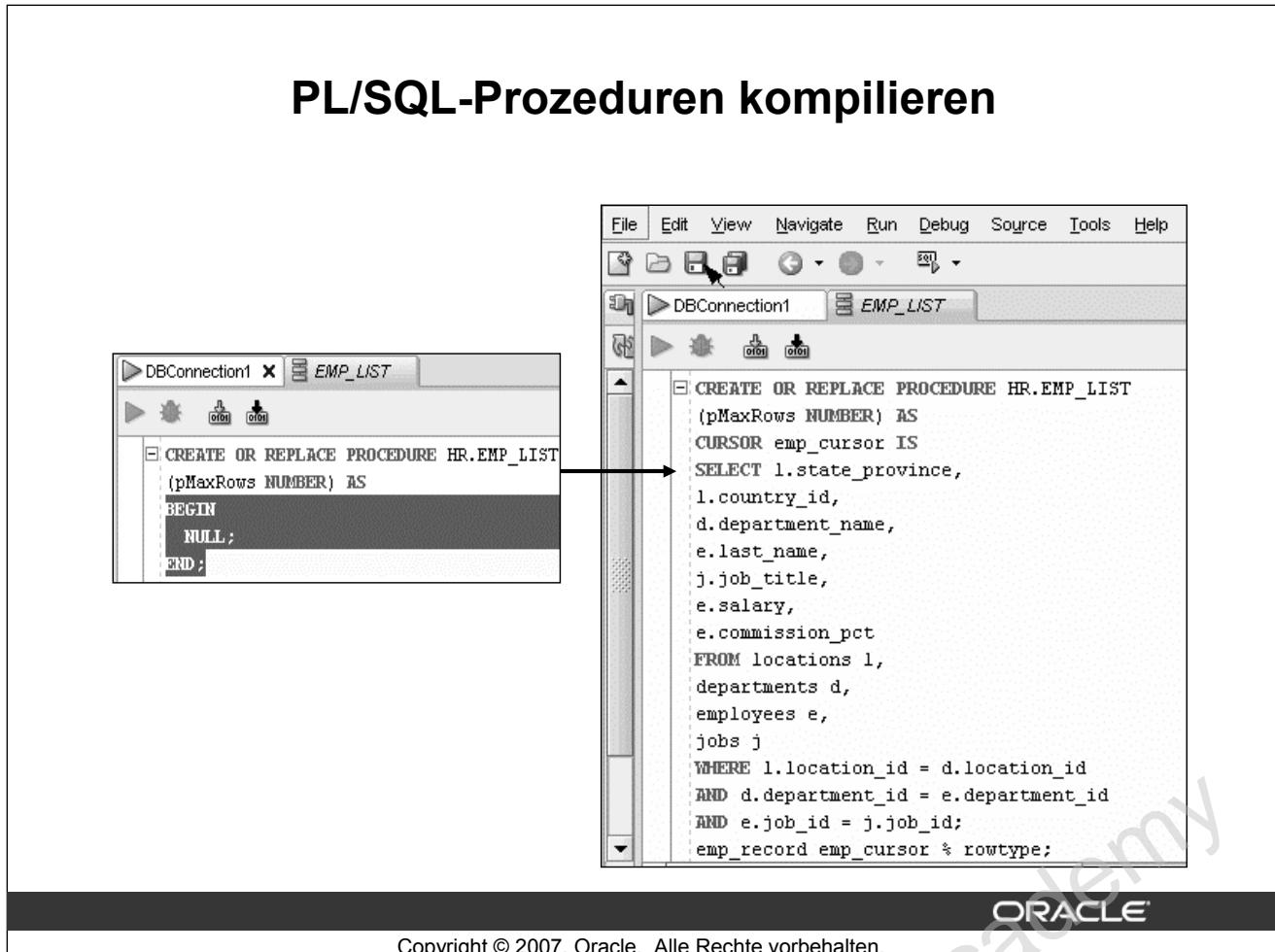
Mit SQL Developer können Sie PL/SQL-Funktionen, -Prozeduren und -Packages erstellen. Um eine PL/SQL-Prozedur zu erstellen, gehen Sie wie folgt vor:

1. Um das Kontextmenü aufzurufen, klicken Sie mit der rechten Maustaste im Connections Navigator auf den Knoten **Procedures**. Wählen Sie dann die Option **Create Procedure**.
2. Geben Sie im Dialogfeld **Create Procedure** die Prozedurinformationen ein, und klicken Sie auf **OK**.

Hinweis: Drücken Sie die Eingabetaste und dann **OK**.

Im Beispiel auf der Folie wird die **EMP_LIST**-Prozedur erstellt. Die Default-Werte für den Parameternamen und den Parametertyp werden durch **pMaxRows** beziehungsweise **NUMBER** ersetzt.

PL/SQL-Prozeduren kompilieren



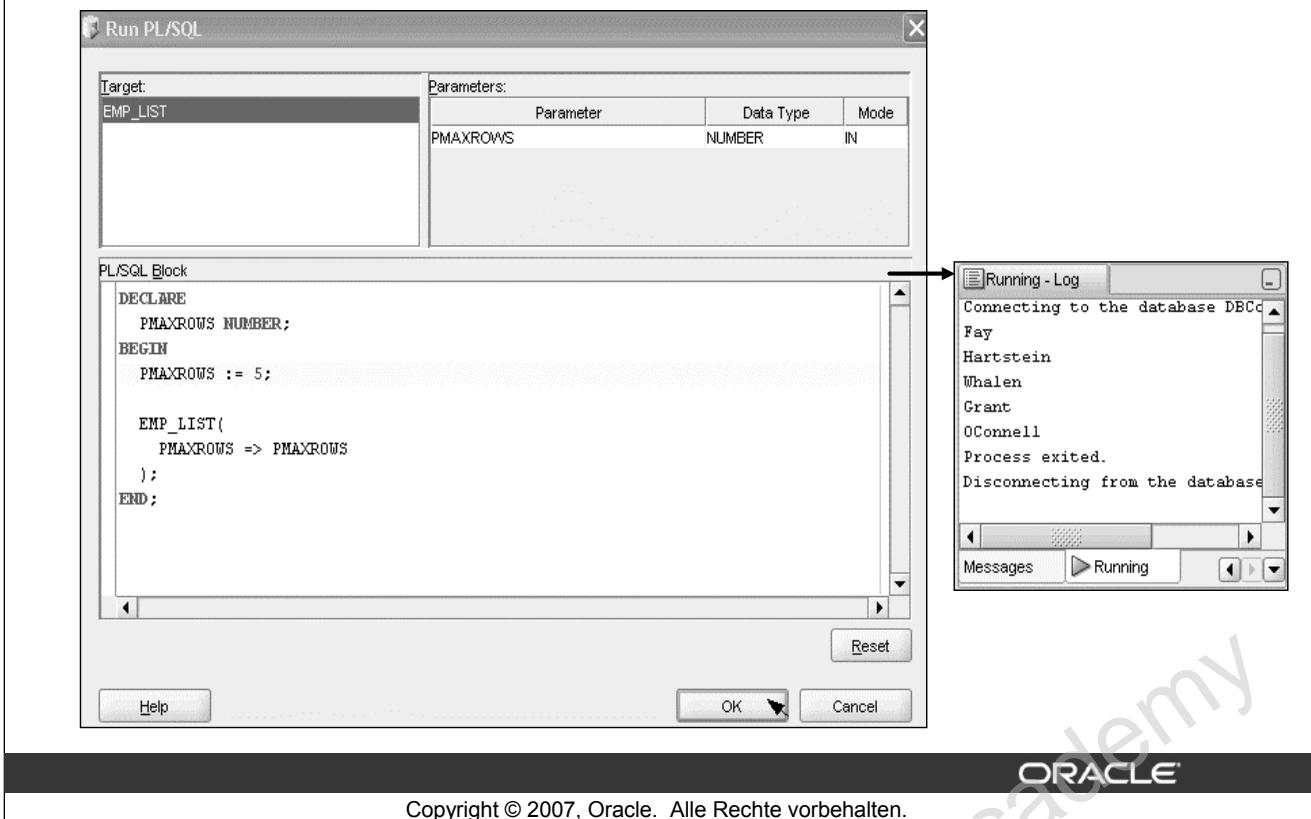
PL/SQL-Prozeduren kompilieren

Wenn Sie im Dialogfeld **Create Procedure** die Parameterinformationen angegeben und auf **OK** geklickt haben, wird im rechten Fenster die Registerkarte für die Prozedur angezeigt. Dann können Sie den anonymen Block mit Ihrem PL/SQL-Code ersetzen.

Um das PL/SQL-Unterprogramm zu kompilieren, klicken Sie in der Symbolleiste auf die Schaltfläche **Save**. Wenn Sie im Connections Navigator den Knoten **Procedures** einblenden, sehen Sie den für die Prozedur hinzugefügten Knoten.

Wenn SQL Developer ein ungültiges PL/SQL-Unterprogramm erkennt, wird der Status im Connections Navigator mit einem roten X über dem Symbol für das Unterprogramm angezeigt. Kompilierungsfehler werden im Log-Fenster angezeigt. Sie können einfach zu der als fehlerhaft gemeldeten Zeile gelangen, indem Sie auf den Fehler doppelklicken. SQL Developer zeigt in der Leiste auf der rechten Seite Fehler und Hinweise an. Wenn Sie den Mauszeiger auf einem der roten Balken in der Leiste platzieren, wird die entsprechende Fehlermeldung angezeigt. Beispiel: Wenn die Fehlermeldung auf einen Formatierungsfehler hinweist, ändern Sie den Code entsprechend und klicken auf das Symbol **Compile**.

PL/SQL-Prozeduren ausführen



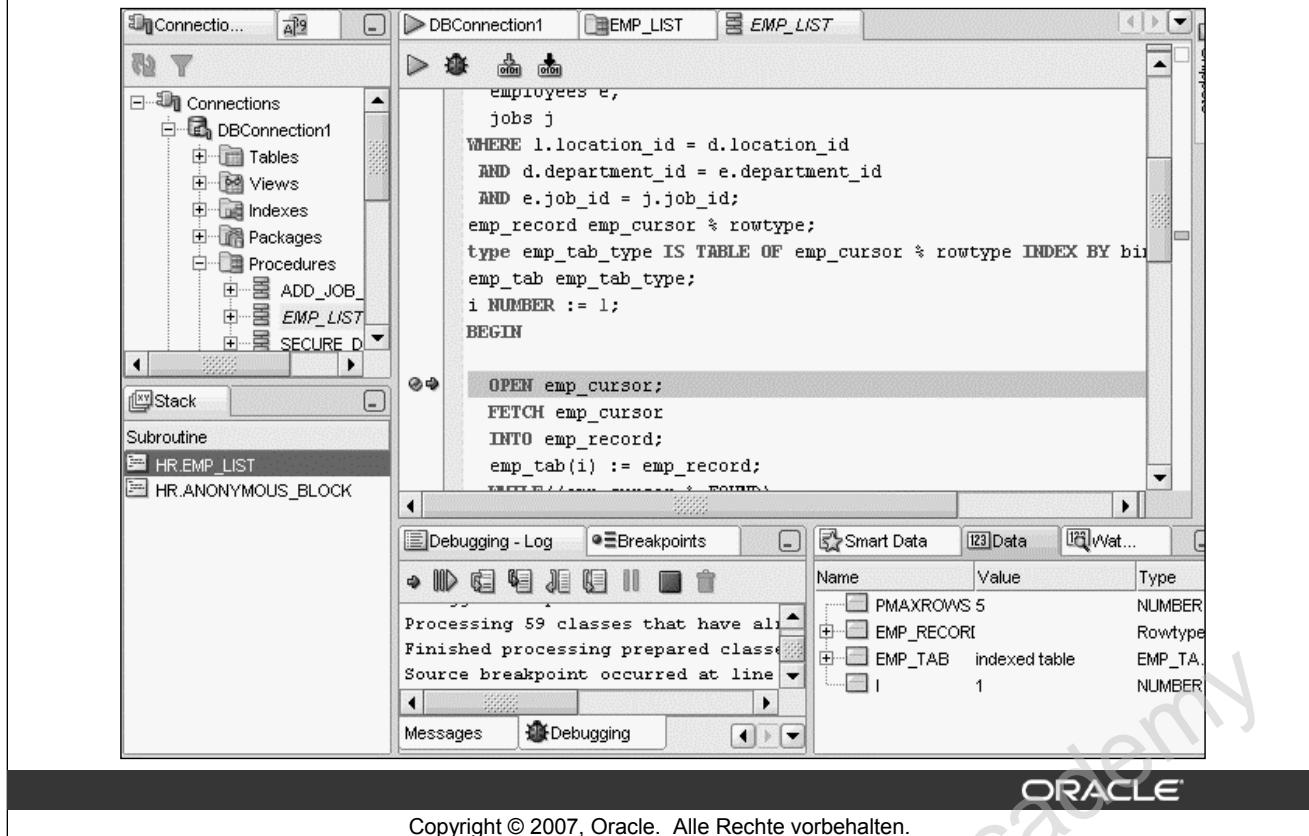
PL/SQL-Prozeduren ausführen

Wenn Sie eine PL/SQL-Prozedur erstellt und kompiliert haben, können Sie sie mit SQL Developer ausführen. Um eine PL/SQL-Prozedur auszuführen, klicken Sie mit der rechten Maustaste links im Navigator auf den Prozedurnamen und wählen **Run**. Optional können Sie auch die Schaltfläche **Run** im rechten Fenster verwenden. Das Dialogfeld **Run PL/SQL** wird aufgerufen. Im Dialogfeld **Run PL/SQL** können Sie die gewünschte Prozedur oder Funktion (**Target**) wählen. Für das ausgewählte Ziel wird eine Liste von Parametern angezeigt.

Sie können den Bereich **PL/SQL Block** verwenden, um Parameter zu füllen, die an die Programmeinheit übergeben werden sollen, oder um komplexe Rückgabetypen zu verarbeiten. Wenn Sie die erforderlichen Änderungen im Dialogfeld **Run PL/SQL** vorgenommen haben, klicken Sie auf **OK**. Im Fenster **Running-Log** werden die erwarteten Ergebnisse angezeigt.

Im Beispiel auf der Folie wird `PMAXROWS := NULL;` in `PMAXROWS := 5;` geändert. Die Ergebnisse der fünf zurückgegebenen Zeilen werden im Fenster **Running-Log** angezeigt.

PL/SQL debuggen



PL/SQL debuggen

Sie können PL/SQL-Funktionen, -Prozeduren oder -Packages debuggen. SQL Developer bietet volle Unterstützung für das PL/SQL-Debugging. Um eine Funktion oder Prozedur zu debuggen, gehen Sie wie folgt vor:

1. Klicken Sie im Connections Navigator auf den Objektnamen.
2. Klicken Sie mit der rechten Maustaste auf das Objekt, und wählen Sie **Compile for debug**.
3. Klicken Sie auf das Symbol **Edit**. Klicken Sie dann oberhalb der Quellliste auf das Symbol **Debug**.

Wenn die Zahlen zum Ein-/Ausschalten vor jeder Code-Zeile noch nicht angezeigt werden, klicken Sie mit der rechten Maustaste auf den Rand des Code Editors und wählen **Toggle Line Numbers**.

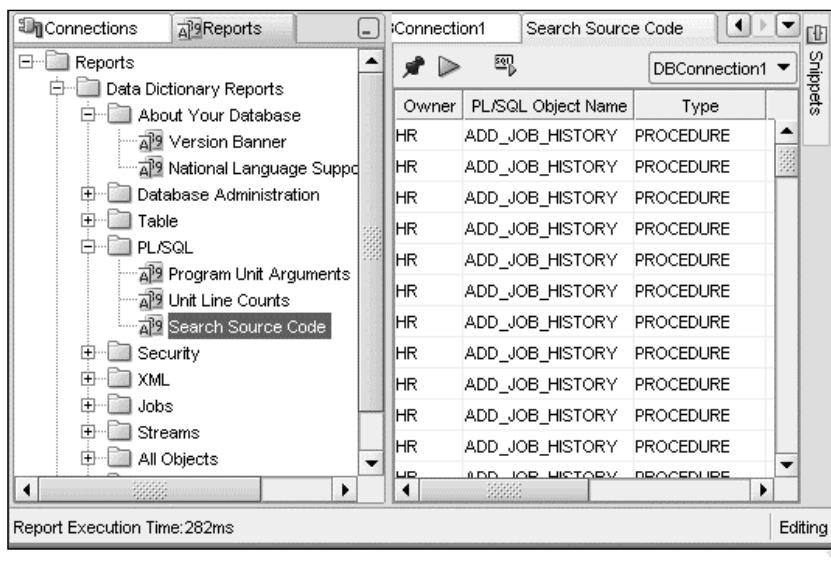
Der PL/SQL-Debugger verfügt über viele Befehle zur Steuerung der Programmausführung, darunter **Step Into**, **Step Over**, **Step Out**, **Run to Cursor** und andere. Wenn der Debugger angehalten wird, können Sie in den Fenstern **Smart Data**, **Watches** oder **Inspector** die Werte von Variablen prüfen und ändern.

Im Fenster **Breakpoints** werden die definierten Breakpoints aufgelistet. In diesem Fenster können Sie neue Breakpoints hinzufügen oder das Verhalten vorhandener Breakpoints anpassen.

Hinweis: Für das PL/SQL-Debugging benötigen Sie die Privilegien `debug any procedure` und `debug connect session`.

Datenbankberichte

SQL Developer stellt mehrere vordefinierte Berichte über die Datenbank und ihre Objekte bereit.



ORACLE

Copyright © 2007, Oracle. Alle Rechte vorbehalten.

Datenbankberichte

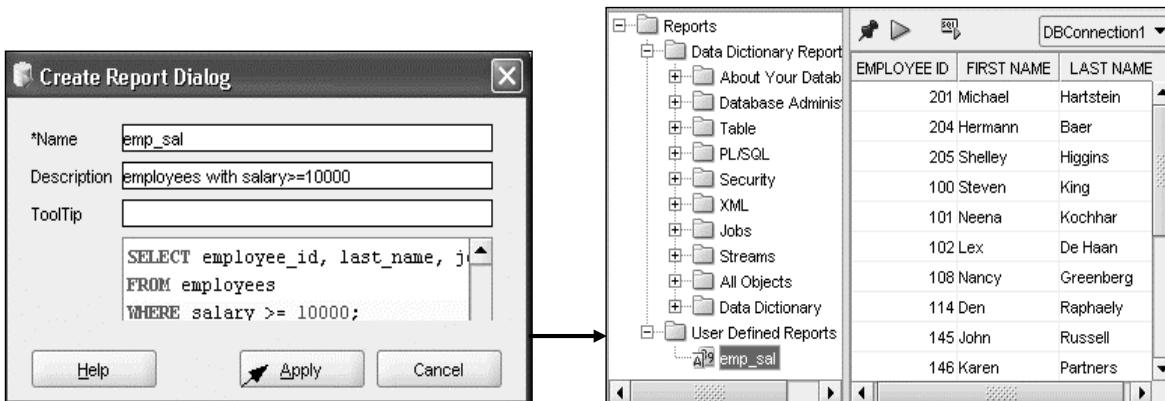
SQL Developer stellt viele Berichte über die Datenbank und ihre Objekte bereit. Diese Berichte werden in folgende Kategorien unterteilt:

- Berichte über die Datenbank (**About Your Database**)
- Berichte zur Datenbankadministration (**Database Administration**)
- Tabellenberichte (**Table**)
- PL/SQL-Berichte (**PL/SQL**)
- Sicherheitsberichte (**Security**)
- XML-Berichte (**XML**)
- Job-Berichte (**Jobs**)
- Stream-Berichte (**Streams**)
- Berichte zu allen Objekten (**All Objects**)
- Data Dictionary-Berichte (**Data Dictionary**)
- Benutzerdefinierte Berichte (**User Defined**)

Um Berichte anzuzeigen, klicken Sie auf der linken Seite des Fensters in die Registerkarte **Reports**. Die einzelnen Berichte werden in Tabbed Panes auf der rechten Seite des Fensters 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 mit der ausgewählten Datenbankverbindung verknüpfte Datenbankbenutzer sehen kann. Die Zeilen werden normalerweise nach dem Eigentümer (**Owner**) sortiert angezeigt.

Benutzerdefinierte Berichte erstellen

Benutzerdefinierte Berichte zur Wiederverwendung erstellen und speichern



Copyright © 2007, Oracle. Alle Rechte vorbehalten.

Benutzerdefinierte Berichte erstellen

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

Im Beispiel auf der Folie wird der Berichtsnname `emp_sal` angegeben. Dabei wurde eine optionale Beschreibung eingegeben, die angibt, dass der Bericht die Details von Angestellten mit einem Gehalt ≥ 10.000 enthält. Die vollständige SQL-Anweisung zum Abrufen der Informationen, die im benutzerdefinierten Bericht angezeigt werden sollen, wird im Bereich für den SQL-Code angegeben. Sie können auch eine optionale QuickInfo aufnehmen, die eingeblendet wird, wenn der Mauszeiger in der Reports Navigator-Anzeige auf den Berichtsnamen bewegt 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 einer Datei namens `UserReports.xml` gespeichert.

Zusammenfassung

In diesem Anhang haben Sie gelernt, wie Sie SQL Developer für folgende Aufgaben verwenden:

- **Datenbankobjekte durchsuchen, erstellen und bearbeiten**
- **SQL-Anweisungen und -Skripts in SQL Worksheet ausführen**
- **PL/SQL-Anweisungen bearbeiten und debuggen**
- **Benutzerdefinierte Berichte erstellen und speichern**

ORACLE

Copyright © 2007, Oracle. Alle Rechte vorbehalten.

Zusammenfassung

SQL Developer ist ein kostenloses grafisches Tool zur Vereinfachung von Aufgaben der Datenbankentwicklung. Mit SQL Developer können Sie Datenbankobjekte durchsuchen, erstellen und bearbeiten. Mit SQL Worksheet können Sie SQL-Anweisungen und -Skripts ausführen. Mit SQL Developer können Sie PL/SQL-Code bearbeiten und debuggen. Darüber hinaus können Sie mit SQL Developer eigene spezielle Berichte erstellen und speichern, die Sie immer wieder verwenden können.



ORACLE

Copyright © 2007, Oracle. Alle Rechte vorbehalten.

Oracle Internal & Oracle Academy
Use Only

Ziele

Nach Beendigung dieses Anhangs haben Sie folgende Ziele erreicht:

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

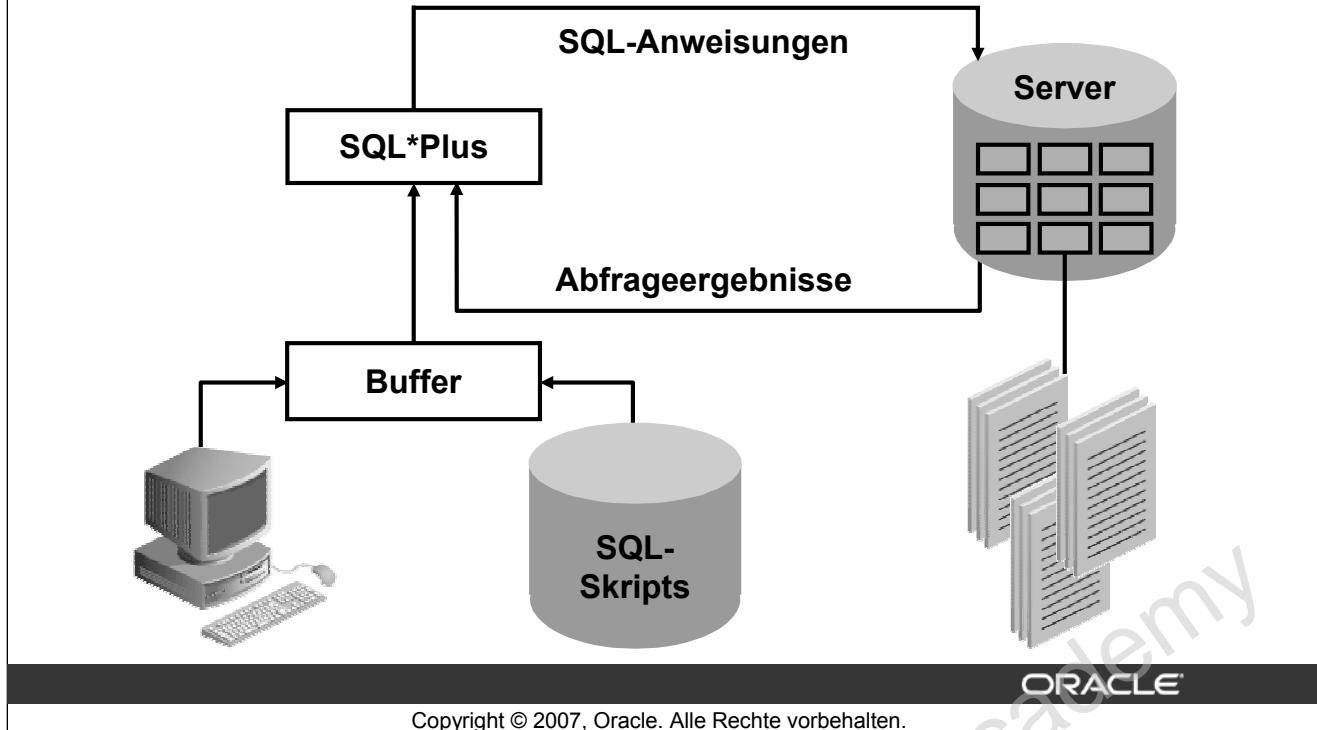
ORACLE®

Copyright © 2007, Oracle. Alle Rechte vorbehalten.

Ziele

Es wird erläutert, wie Sie SELECT-Anweisungen erstellen, die Sie immer wieder verwenden können. Darüber hinaus wird in diesem Anhang die Ausführung von SQL-Anweisungen mit Hilfe von SQL*Plus-Befehlen behandelt. Außerdem wird beschrieben, wie Sie die Ausgabe mit SQL*Plus-Befehlen formatieren, SQL-Befehle bearbeiten und Skripts in SQL*Plus speichern.

SQL und SQL*Plus – Interaktion



Copyright © 2007, Oracle. Alle Rechte vorbehalten.

SQL und SQL*Plus

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

SQL – Features

- Kann von allen Benutzern verwendet werden, auch von Benutzern ohne oder mit geringer Programmiererfahrung
- Ist eine nicht prozedurale Sprache
- Reduziert den Zeitaufwand für das Erstellen und Verwalten von Systemen
- Ist an das Englische angelehnt

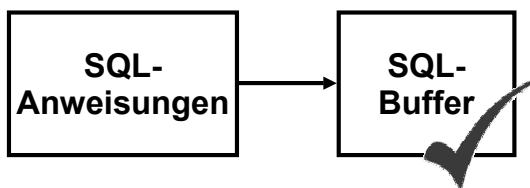
SQL*Plus – Features

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

SQL-Anweisungen und SQL*Plus-Befehle – Vergleich

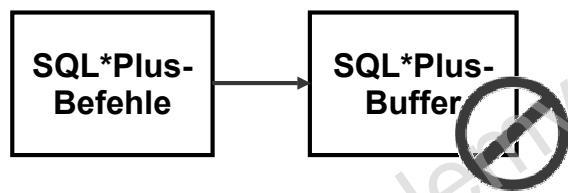
SQL

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



SQL*Plus

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



Copyright © 2007, Oracle. Alle Rechte vorbehalten.

SQL und SQL*Plus (Fortsetzung)

In der folgenden Tabelle werden die Merkmale von SQL und SQL*Plus gegenübergestellt:

SQL	SQL*Plus
Ist eine Sprache für die Kommunikation mit dem Oracle-Server, mit dem Ziel, auf Daten zuzugreifen	Erkennt SQL-Anweisungen und sendet sie an den Server
Basiert auf Standard-SQL des ANSI (American National Standards Institute)	Ist die Oracle-firmeneigene Oberfläche zum Ausführen von SQL-Anweisungen
Bearbeitet Daten und Tabellendefinitionen in der Datenbank	Erlaubt keine Bearbeitung von Werten in der Datenbank
Wird in einer oder mehreren Zeilen in den SQL-Buffer eingegeben	Wird Zeile für Zeile eingegeben und nicht im SQL-Buffer gespeichert
Enthält kein Fortsetzungszeichen	Verwendet den Gedankenstrich (-) als Fortsetzungszeichen, wenn der Befehl länger als eine Zeile ist
Kann nicht abgekürzt werden	Kann abgekürzt werden
Verwendet ein Beendigungszeichen, um Befehle unmittelbar auszuführen	Erfordert kein Beendigungszeichen und führt alle Befehle unmittelbar aus
Verwendet Funktionen, um bestimmte Formatierungen vorzunehmen	Verwendet Befehle, um Daten zu formatieren

SQL*Plus – Überblick

- Bei SQL*Plus anmelden
- Die Tabellenstruktur beschreiben
- SQL-Anweisung bearbeiten
- SQL in SQL*Plus ausführen
- SQL-Anweisungen in Dateien speichern und SQL-Anweisungen an Dateien anhängen
- Gespeicherte Dateien ausführen
- Befehle aus Dateien in den Buffer laden, um sie zu bearbeiten

ORACLE

Copyright © 2007, Oracle. Alle Rechte vorbehalten.

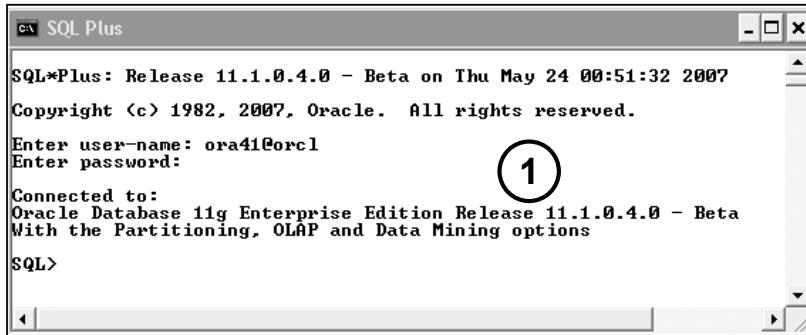
SQL*Plus

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

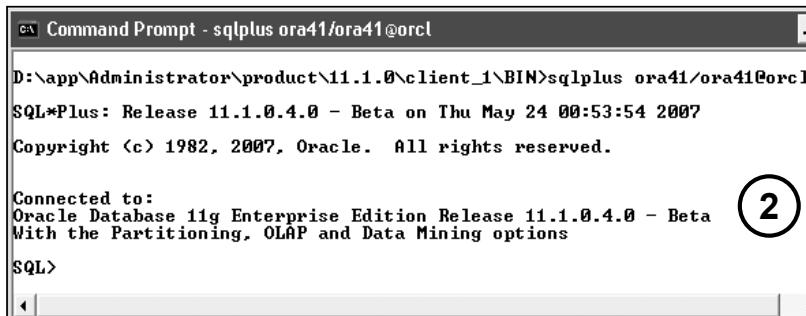
- SQL-Anweisungen ausführen, um Daten aus einer Datenbank abzurufen, zu ändern, hinzuzufügen und zu entfernen
 - Abfrageergebnisse als Berichte formatieren, speichern und drucken sowie Berechnungen mit den Abfrageergebnissen ausführen
 - Skriptdateien erstellen, um SQL-Anweisungen für die spätere erneute Verwendung zu speichern
- SQL*Plus-Befehle können in die folgenden Hauptkategorien unterteilt werden:

Kategorie	Zweck
Umgebung	Allgemeines Verhalten von SQL-Anweisungen für die Session beeinflussen
Format	Abfrageergebnisse formatieren
Dateibearbeitung	Skriptdateien speichern, laden und ausführen
Ausführung	SQL-Anweisungen aus dem SQL-Buffer an den Oracle-Server senden
Bearbeitung	SQL-Anweisungen im Buffer ändern
Interaktion	Variablen erstellen und an SQL-Anweisungen übergeben, Variablenwerte drucken und Meldungen auf dem Bildschirm anzeigen
Verschiedenes	Bei der Datenbank anmelden, SQL*Plus-Umgebung bearbeiten und Spaltendefinitionen anzeigen

Bei SQL*Plus anmelden



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



ORACLE

Copyright © 2007, Oracle. Alle Rechte vorbehalten.

Bei SQL*Plus anmelden

Wie Sie SQL*Plus aufrufen, hängt von Ihrem Betriebssystemtyp oder Ihrer Windows-Umgebung ab.

So melden Sie sich aus der Windows-Umgebung an:

1. Wählen Sie **Start > Programs > Oracle > Application Development > SQL*Plus**.
2. Geben Sie den Benutzernamen, das Passwort und den Datenbanknamen ein.

So melden Sie sich aus einer Befehlszeilenumgebung an:

1. Melden Sie sich bei Ihrem Rechner an.
2. Geben Sie den auf der Folie dargestellten `sqlplus`-Befehl ein.

Für die Syntax gilt:

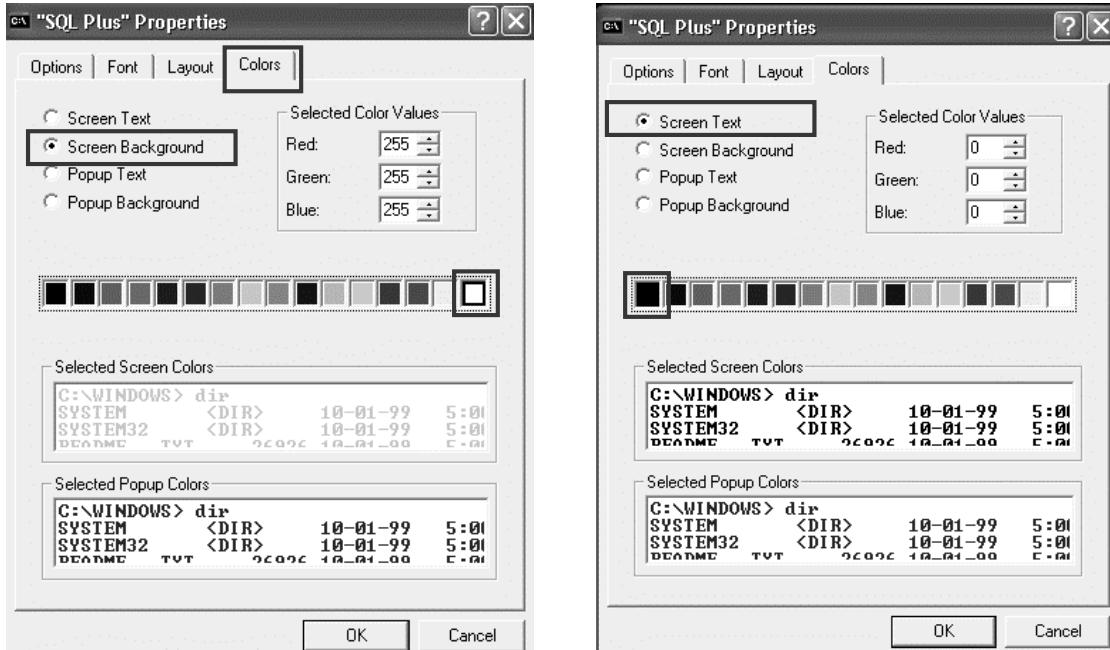
`username` Ihr Benutzername für die Datenbank

`password` Ihr Passwort für die Datenbank. (Das Passwort wird bei der Eingabe sichtbar.)

`@database` Der Connect String für die Datenbank

Hinweis: Um die Integrität Ihres Passwortes zu wahren, dürfen Sie es nicht in der Eingabeaufforderung des Betriebssystems eingeben. Geben Sie stattdessen nur Ihren Benutzernamen ein. Geben Sie das Passwort in der Passwort-Eingabeaufforderung ein.

Einstellungen der SQL*Plus-Umgebung ändern



ORACLE

Copyright © 2007, Oracle. Alle Rechte vorbehalten.

Einstellungen der SQL*Plus-Umgebung ändern

Sie können das Erscheinungsbild der SQL*Plus-Umgebung ändern. Verwenden Sie hierfür das Dialogfeld **SQL*Plus Properties**.

Klicken Sie im Fenster **SQL*Plus** mit der rechten Maustaste auf die Titelleiste. Wählen Sie im aufgerufenen Kontextmenü **Properties**. In der Registerkarte **Colors** des Dialogfeldes **SQL*Plus Properties** können Sie **Screen Background** und **Screen Text** aktivieren.

Tabellenstrukturen anzeigen

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

```
DESC[RIBE] tablename
```

ORACLE®

Copyright © 2007, Oracle. Alle Rechte vorbehalten.

Tabellenstrukturen anzeigen

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

Für die Syntax gilt:

tablename Der Name einer beliebigen vorhandenen für Benutzer zugänglichen Tabelle, View oder eines Synonyms

Zeigen Sie die DEPARTMENTS-Tabelle mit folgendem Befehl an:

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

Tabellenstrukturen anzeigen

```
DESCRIBE departments
```

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

ORACLE

Copyright © 2007, Oracle. Alle Rechte vorbehalten.

Tabellenstrukturen anzeigen (Fortsetzung)

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

Null? : Gibt an, ob eine Spalte Daten enthalten muss. (NOT NULL zeigt an, dass eine Spalte Daten enthalten muss.)

Type : Zeigt den Datentyp einer Spalte an

SQL*Plus-Bearbeitungsbefehle

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

ORACLE

Copyright © 2007, Oracle. Alle Rechte vorbehalten.

SQL*Plus-Bearbeitungsbefehle

SQL*Plus-Befehle werden Zeile für Zeile eingegeben und nicht im SQL-Buffer 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-Buffer
DEL	Löscht die aktuelle Zeile
DEL <i>n</i>	Löscht die Zeile <i>n</i>
DEL <i>m</i> <i>n</i>	Löscht die Zeilen <i>m</i> bis einschließlich <i>n</i>

Richtlinien

- Wenn Sie die EINGABETASTE drücken, bevor Sie einen Befehl abgeschlossen haben, zeigt SQL*Plus die Nummer der entsprechenden Befehlszeile an.
- Sie beenden den SQL-Buffer, indem Sie eines der Beendigungszeichen (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***
- **O *text***

ORACLE®

Copyright © 2007, Oracle. Alle Rechte vorbehalten.

SQL*Plus-Bearbeitungsbefehle (Fortsetzung)

Befehl	Beschreibung
I [NPUT]	Fügt eine unbestimmte Anzahl von Zeilen ein
I [NPUT] <i>text</i>	Fügt eine Zeile mit dem Text (<i>text</i>) ein
L [IST]	Listet alle Zeilen im SQL-Buffer auf
L [IST] <i>n</i>	Listet eine Zeile auf (durch <i>n</i> angegeben)
L [IST] <i>m n</i>	Listet einen Bereich an Zeilen (<i>m</i> bis einschließlich <i>n</i>) auf
R [UN]	Zeigt die aktuell im Buffer befindliche SQL-Anweisung an und führt sie aus
<i>n</i>	Macht <i>n</i> zur aktuellen Zeile
<i>n text</i>	Ersetzt die Zeile <i>n</i> durch den Text (<i>text</i>)
O <i>text</i>	Fügt eine Zeile vor Zeile 1 ein

Hinweis: Sie können nur jeweils einen SQL*Plus-Befehl in einer SQL-Eingabeaufforderung eingeben. SQL*Plus-Befehle werden nicht im SQL-Buffer 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 © 2007, Oracle. Alle Rechte vorbehalten.

LIST, n und APPEND

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

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

CHANGE-Befehl

```
LIST
```

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

```
c/employees/departments
```

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

```
LIST
```

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

ORACLE

Copyright © 2007, Oracle. Alle Rechte vorbehalten.

CHANGE-Befehl

- Zeigen Sie mit L [IST] den Inhalt des Buffers an.
- Ändern Sie mit dem C [HANGE]-Befehl den Inhalt der aktuellen Zeile im SQL-Buffer. Ersetzen Sie in diesem Fall die employees- Tabelle durch die departments-Tabelle. Die neue aktuelle Zeile wird angezeigt.
- Prüfen Sie mit dem L [IST]-Befehl den neuen Inhalt des SQL-Buffers.

SQL*Plus-Dateibefehle

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

ORACLE

Copyright © 2007, Oracle. Alle Rechte vorbehalten.

SQL*Plus-Dateibefehle

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

Befehl	Beschreibung
SAV[E] <i>filename</i> [.ext] [REP[LACE]APP[END]]	Speichert den aktuellen Inhalt des SQL-Buffers in einer Datei. Verwenden Sie APPEND, um den Buffer-Inhalt an eine vorhandene Datei anzuhängen, oder REPLACE, um eine vorhandene Datei zu ersetzen. Die Default-Erweiterung ist .sql.
GET <i>filename</i> [.ext]	Schreibt den Inhalt einer zuvor gespeicherten Datei in den SQL-Buffer. Die Default-Erweiterung für den Dateinamen ist .sql.
STA[RT] <i>filename</i> [.ext]	Führt eine zuvor gespeicherte Befehlsdatei aus
@ <i>filename</i>	Führt eine zuvor gespeicherte Befehlsdatei aus (identisch mit START)
ED[IT]	Ruft den Editor auf und speichert den Buffer-Inhalt in einer Datei namens afiedt.buf
ED[IT] [<i>filename</i> [.ext]]	Ruft den Editor auf, um den Inhalt einer gespeicherten Datei zu bearbeiten
SPO[OL] [<i>filename</i> [.ext]] OFF OUT	Speichert Abfrageergebnisse in einer Datei. OFF schließt die Spool-Datei. OUT schließt die Spool-Datei und sendet die Dateiergebnisse an den Drucker.
EXIT	Beendet SQL*Plus

Befehle SAVE, START und EDIT

```
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 © 2007, Oracle. Alle Rechte vorbehalten.

Befehle SAVE, START und EDIT

SAVE

Mit dem SAVE-Befehl speichern Sie den aktuellen Inhalt des Buffers in einer Datei. So können Sie häufig benutzte Skripts später wieder verwenden.

START

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

```
@my_query
```

Befehle SAVE, START und EDIT

EDIT my_query



ORACLE

Copyright © 2007, Oracle. Alle Rechte vorbehalten.

Befehle SAVE, START und EDIT (Fortsetzung)

EDIT

Mit dem EDIT-Befehl bearbeiten Sie ein vorhandenes Skript. Der Befehl öffnet einen Editor und lädt die Skriptdatei. Nachdem Sie die Änderungen vorgenommen haben, beenden Sie den Editor, um zur SQL*Plus-Befehlszeile zurückzukehren.

Hinweis: Das Begrenzungszeichen "/" gibt das Ende der Anweisung an. In einer Datei führt das Begrenzungszeichen dazu, dass SQL*Plus die vor dem Trennzeichen vorhandene Anweisung ausführt. Das Begrenzungszeichen muss das erste Zeichen einer neuen Zeile sein, die der Anweisung direkt folgt.

SERVEROUTPUT-Befehl

- Mit dem Befehl **SET SERVEROUT [PUT]** können Sie 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 Default-Größe ist nun unbegrenzt.
- Die Ressourcen werden nicht reserviert, wenn SERVEROUTPUT festgelegt ist.
- Verwenden Sie UNLIMITED, da es nicht zu Performance-Einbußen kommt. Ausnahme: Sie möchten physischen Speicher einsparen.

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

ORACLE®

Copyright © 2007, Oracle. Alle Rechte vorbehalten.

SERVEROUTPUT-Befehl

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-I/Os erfolgen über APIs, die mit anderen Programmen interagieren. Beispiel: Das DBMS_OUTPUT Package verfügt über Prozeduren wie PUT_LINE. Um das Ergebnis außerhalb von PL/SQL anzuzeigen, ist ein anderes Programm (wie SQL*Plus) erforderlich, um die an DBMS_OUTPUT übergebenen Daten zu lesen und anzuzeigen.

SQL*Plus zeigt DBMS_OUTPUT-Daten erst an, wenn Sie den SQL*Plus-Befehl SET SERVEROUTPUT ON wie folgt zum ersten Mal absetzen:

```
SET SERVEROUTPUT ON
```

Hinweise:

- SIZE legt die Anzahl der Byte der Ausgabe fest, die im Oracle-Datenbank-Server gepuffert werden können. Der Default ist UNLIMITED. *n* darf nicht unter 2000 oder über 1.000.000 liegen.
- Weitere Informationen über SERVEROUTPUT finden Sie im *Oracle Database PL/SQL User's Guide and Reference 11g*.

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 die Inhalte einer vorhandenen Datei. Ist die Datei nicht vorhanden, wird sie durch REPLACE erstellt
APP[END]	Fügt die Buffer-Inhalte am Ende der angegebenen Datei ein
OFF	Stoppt das Spool-Verfahren
OUT	Stoppt das Spool-Verfahren und sendet die Datei an den Standarddrucker (Default) des Computers

ORACLE

Copyright © 2007, Oracle. Alle Rechte vorbehalten.

SQL*Plus-Befehl SPOOL

Der SPOOL-Befehl speichert die Abfrageergebnisse in einer Datei oder sendet die Datei optional an einen Drucker. Der SPOOL-Befehl wurde weiterentwickelt. 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 Default.

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

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

AUTOTRACE-Befehl

- Zeigt einen Bericht an, wenn SQL DML-Anweisungen wie SELECT, INSERT, UPDATE oder DELETE erfolgreich ausgeführt wurden
- Der Bericht kann Statistiken zur Ausführung und den Pfad zur Ausführungsstatistik angeben.

```
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 © 2007, Oracle. Alle Rechte vorbehalten.

AUTOTRACE-Befehl

EXPLAIN zeigt den Pfad der Abfrageausführung, indem EXPLAIN PLAN durchgeführt wird. STATISTICS zeigt Statistiken zu SQL-Anweisungen an. Die Formatierung des AUTOTRACE-Berichts kann abhängig von der Version des Servers, mit dem Sie verbunden sind, und der Server-Konfiguration variieren. Das DBMS_XPLAN-Package ermöglicht Ihnen die einfache Anzeige der Ausgabe des EXPLAIN PLAN-Befehls in verschiedenen, vordefinierten Formaten.

Hinweise:

- Weitere Informationen über das Package und die Unterprogramme finden Sie im *Oracle Database PL/SQL Packages and Types Reference 11g Guide*.
- Weitere Informationen zu EXPLAIN PLAN finden Sie in *Oracle Database SQL Reference 11g*.
- Weitere Informationen zu Ausführungsplänen und Statistiken finden Sie im *Oracle Database Performance Tuning Guide 11g*.

Zusammenfassung

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

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

ORACLE

Copyright © 2007, Oracle. Alle Rechte vorbehalten.

Zusammenfassung

SQL*Plus ist eine Ausführungsumgebung, mit der Sie SQL-Befehle an den Datenbank-Server senden sowie SQL-Befehle bearbeiten und speichern können. Sie können die Befehle in der SQL-Eingabeaufforderung oder in einer Skriptdatei ausführen.

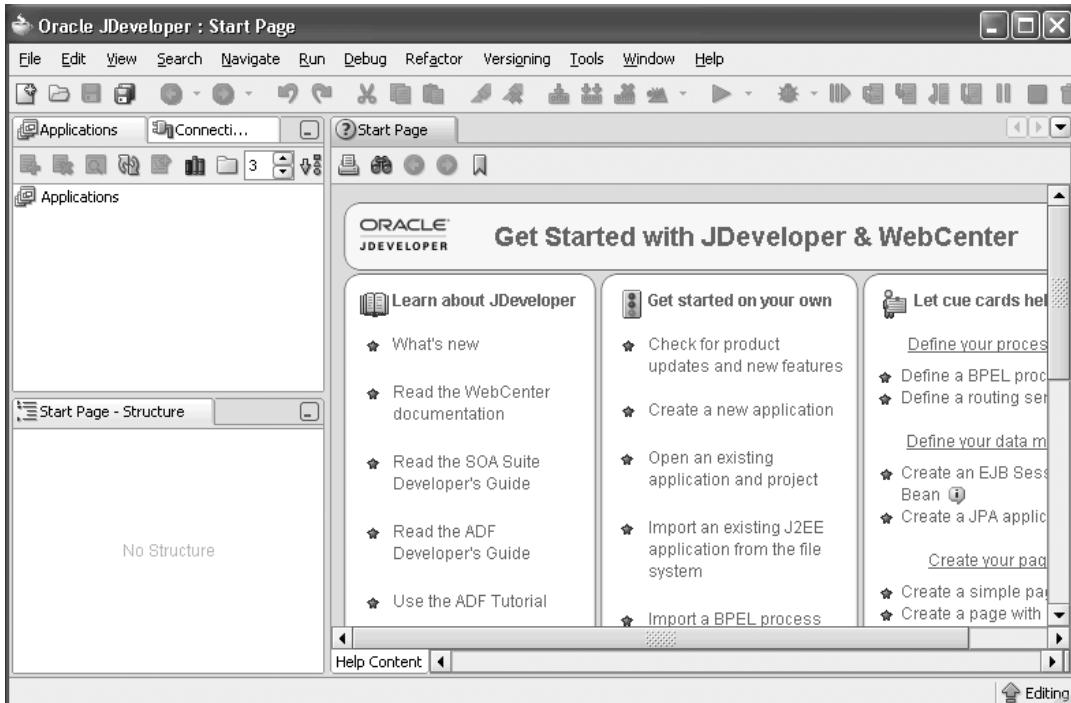


ORACLE

Copyright © 2007, Oracle. Alle Rechte vorbehalten.

Oracle Internal & Oracle Academy
Use Only

Oracle JDeveloper



ORACLE®

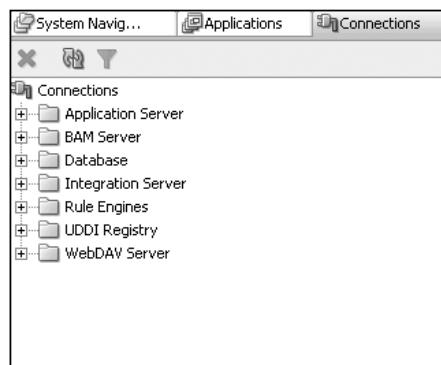
Copyright © 2007, Oracle. Alle Rechte vorbehalten.

Oracle JDeveloper

Oracle JDeveloper ist eine integrierte Entwicklungsumgebung (Integrated Development Environment, IDE) für die Entwicklung und das Deployment von Java-Anwendungen und Web Services. Es unterstützt jede Lebenszyklusphase der Software-Entwicklung (Software Development Life Cycle, SDLC) von der Modellierung bis zum Deployment. Mit der bereitgestellten Funktionalität verwenden Sie bei der Anwendungsentwicklung die neuesten Industriestandards für Java, XML und SQL.

Oracle JDeveloper 10g stellt mit Hilfe dieser Funktionalität einen neuen Ansatz der J2EE-Entwicklung vor, der visuell und deklarativ ist. Dieser innovative Ansatz macht die J2EE-Entwicklung einfach und effizient.

Connections Navigator



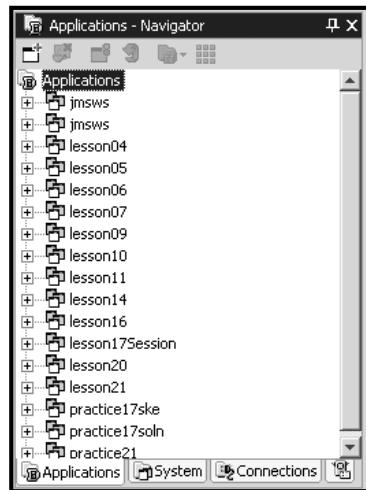
ORACLE®

Copyright © 2007, Oracle. Alle Rechte vorbehalten.

Connections Navigator

In Oracle JDeveloper können Sie die für die Anmeldung bei einer Datenbank benötigten Informationen in einem Objekt namens "Connection" speichern. Dieses Objekt wird als Teil der IDE-Einstellungen gespeichert und lässt sich exportieren sowie importieren, so dass sie in einer Benutzergruppe problemlos gemeinsam verwendet werden kann. Eine Verbindung dient mehreren Zwecken, die vom Durchsuchen von Datenbanken und der Erstellung von Anwendungen bis hin zum Deployment reichen.

Applications Navigator



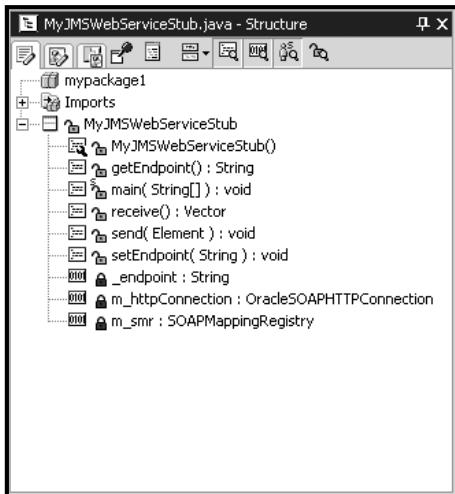
ORACLE®

Copyright © 2007, Oracle. Alle Rechte vorbehalten.

Applications Navigator

Der Applications Navigator zeigt Ihre Anwendung und die darin enthaltenen Daten in logischer Strukturierung an. Er stellt eine Infrastruktur bereit, in die die verschiedenen Erweiterungen als Plug-Ins eingefügt werden können. Die Erweiterungen verwenden diese Infrastruktur, um ihre Daten und Menüs konsistent und abstrakt zu organisieren. Der Applications Navigator kann zwar einzelne Dateien (wie etwa Java-Quelldateien) enthalten, wurde aber eigentlich entwickelt, um komplexe Daten zu konsolidieren. Komplexe Datenarten wie Entity-Objekte, UML-Diagramme, EJB oder Web Services werden in diesem Navigator als einzelne Knoten angezeigt. Die Raw-Dateien, aus denen sich diese abstrakten Knoten zusammensetzen, werden im Fenster **Structure** angezeigt.

Fenster "Structure"



ORACLE

Copyright © 2007, Oracle. Alle Rechte vorbehalten.

Fenster "Structure"

Das Fenster **Structure** zeigt die Daten des Dokuments in einer Strukturansicht an, das im aktiven Fenster eines der Strukturfenster (Navigatoren, Editoren, Viewer und Property Inspector) ausgewählt ist.

Im Fenster **Structure** können Sie Dokumentendaten auf vielfältige Weise anzeigen. Welche Strukturen angezeigt werden können, hängt vom Dokumenttyp ab. Für Java-Dateien können Sie die Code-Struktur, die UI-Struktur oder die UI-Modelldaten anzeigen. Für XML-Dateien können Sie die XML-Struktur, die Entwurfsstruktur oder die UI-Modelldaten anzeigen.

Das Fenster **Structure** ist dynamisch und verfolgt durchgehend die im aktiven Fenster aktuelle und für den derzeit aktiven Editor relevante Auswahl (der Inhalt des Fensters darf nicht in einer bestimmten Ansicht fixiert sein). Handelt es sich bei der aktuellen Auswahl um einen Knoten im Navigator, wird der Default-Editor verwendet. Um die Ansicht der Struktur für die aktuelle Auswahl zu ändern, wählen Sie im Fenster **Structure** eine andere Registerkarte.

Editor-Fenster

```
SHOW_CUST_CALL
PROCEDURE show_cust_call (
  custid IN NUMBER default 101) AS
  BEGIN NULL;
  http.prn('');
  http.prn('');
  http.prn('');
  <HTML>
  <BODY>
  <form method="POST" action="show_cust">
  <p>Enter the Customer ID:</p>
  <input type="text" name="custid">
  <input type="submit" value="Submit">
  </form>
  </BODY>
  </HTML>
  );
  END;
```

ORACLE

Copyright © 2007, Oracle. Alle Rechte vorbehalten.

Editor-Fenster

Sie können alle Projektdateien in einem einzigen Editor-Fenster anzeigen. Sie können mehrere Ansichten derselben Datei oder mehrere Ansichten verschiedener Dateien öffnen.

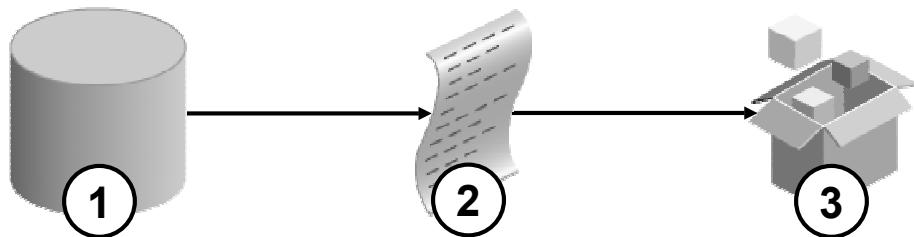
Die Registerkarten am oberen Rand des Editor-Fensters sind Dokumentregisterkarten. Wenn Sie auf eine Dokumentregisterkarte klicken, wird diese Datei zur aktuellen Datei und im Fenster des aktuellen Editors im Vordergrund angezeigt.

Die für eine gegebene Datei am unteren Rand des Editor-Fensters angeordneten Registerkarten sind die Editor-Registerkarten. Wenn Sie auf eine Editor-Registerkarte klicken, wird die Datei in diesem Editor geöffnet.

Java Stored Procedures bereitstellen

Bevor Sie Java Stored Procedures bereitstellen, gehen Sie wie folgt vor:

- 1. Melden Sie sich bei der Datenbank an.**
- 2. Erstellen Sie ein Deployment-Profil.**
- 3. Stellen Sie die Objekte bereit.**



ORACLE

Copyright © 2007, Oracle. Alle Rechte vorbehalten.

Java Stored Procedures bereitstellen

Erstellen Sie ein Deployment-Profil für Java Stored Procedures, und stellen Sie anschließend die Klassen und optional alle öffentlichen statischen Methoden für JDeveloper bereit. Verwenden Sie hierzu die Profileinstellungen.

Wenn Sie das Deployment für eine Datenbank ausführen, werden die im Deployment Profile Wizard eingegebenen Informationen und zwei Oracle-Datenbank-Utilitys verwendet:

- `loadjava` lädt die Java-Klasse, die die Stored Procedures enthält, in eine Oracle-Datenbank.
- `publish` generiert die für PL/SQL-Aufrufe spezifischen Wrapper für die geladenen öffentlichen statischen Methoden. Durch die Veröffentlichung können die Java-Methoden als PL/SQL-Funktionen oder -Prozeduren aufgerufen werden.

Java für PL/SQL veröffentlichen

FormatCreditCardNo.java CCFORMAT

```
public class FormatCreditCardNo
{
    public static final void formatCard(String[] cardno)
    {
        int count=0, space=0;
        String oldcc=cardno[0];
        // System.out.println("Printing the card no initially "+oldcc);
        String[] newcc= {" "};
        while (count<16)
        {
            newcc[0]+= oldcc.charAt(count);
            space++;
            if (space ==4)
            {   newcc[0]+=" "; space=0; }
            count++;
        }
        cardno[0]=newcc [0];
    }
}
```

FormatCreditCardNo.java CCFORMAT

```
PROCEDURE ccformat (x IN OUT varchar2)
AS LANGUAGE JAVA
NAME 'FormatCreditCardNo.formatCard(java.lang.String[])';
```

ORACLE

Copyright © 2007, Oracle. Alle Rechte vorbehalten.

Java für PL/SQL veröffentlichen

Die Folie zeigt den Java-Code und verdeutlicht, wie er in einer PL/SQL-Prozedur veröffentlicht wird.

Programmeinheiten erstellen



```
TEST_JDEV
FUNCTION "TEST_JDEV" RETURN VARCHAR2
AS
BEGIN
    RETURN(' ');
END;
```

Skeleton der Funktion

ORACLE

Copyright © 2007, Oracle. Alle Rechte vorbehalten.

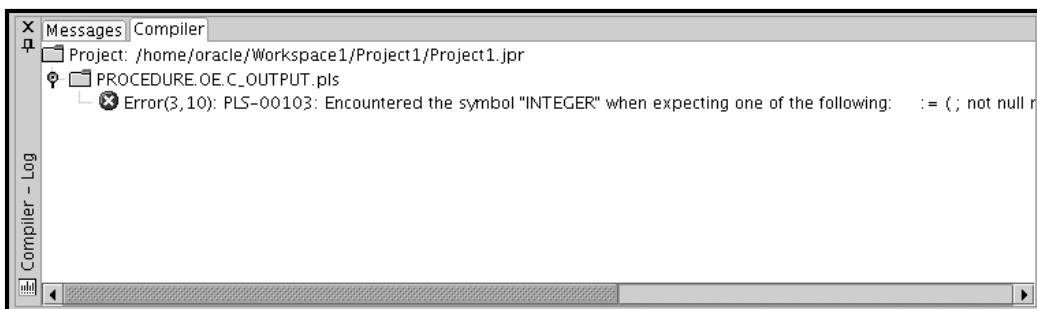
Programmeinheiten erstellen

So erstellen Sie eine PL/SQL-Programmeinheit:

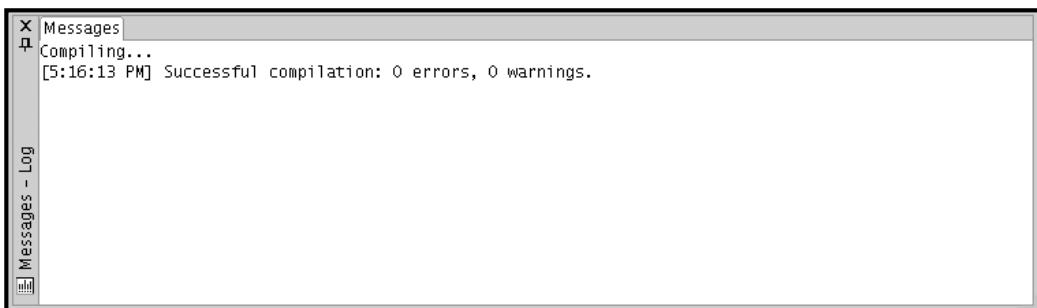
1. Klicken Sie auf **View > Connections Navigator**.
2. Blenden Sie die Datenbank ein, und wählen Sie eine Datenbankverbindung.
3. Blenden Sie in der Verbindung ein Schema ein.
4. Klicken Sie mit der rechten Maustaste auf einen Ordner, der dem Objekttyp (Prozeduren, Packages, Funktionen) entspricht.
5. Klicken Sie auf **New PL/SQL object_type**. Für die Funktion, das Package oder die Prozedur wird das Dialogfeld **Create PL/SQL** angezeigt.
6. Geben Sie für die Funktion, das Package oder die Prozedur einen gültigen Namen ein, und klicken Sie auf **OK**.

Eine Skeleton-Definition wird erstellt und im Code Editor geöffnet. Sie können anschließend das Unterprogramm bearbeiten, um es an Ihre Anforderungen anzupassen.

Kompilieren



Kompilierung mit Fehlern



Kompilierung ohne Fehler

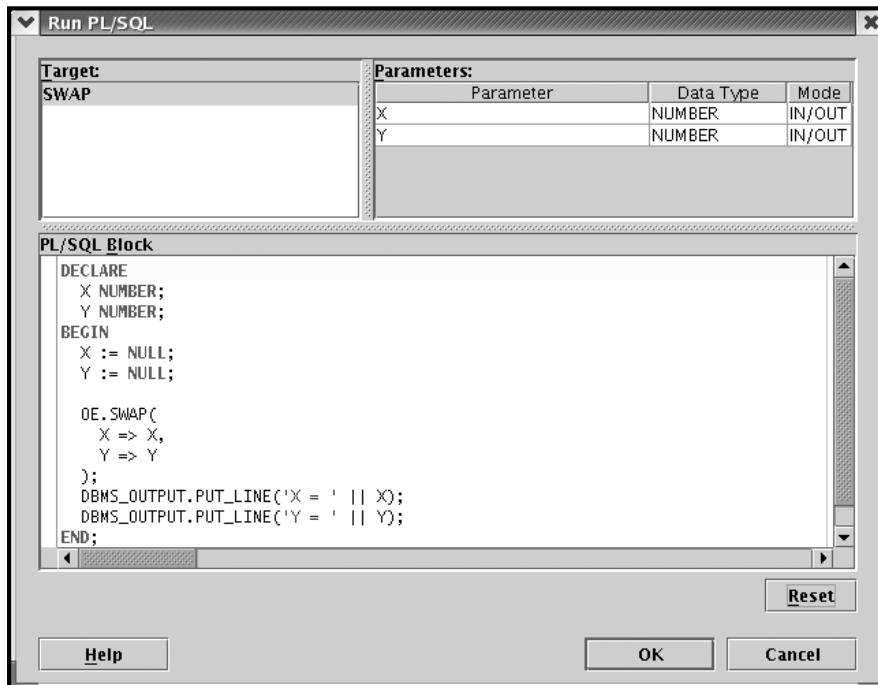
ORACLE

Copyright © 2007, Oracle. Alle Rechte vorbehalten.

Kompilieren

Nachdem Sie die Skeleton-Definition bearbeitet haben, müssen Sie die Programmeinheit kompilieren. Klicken Sie im Connections Navigator mit der rechten Maustaste auf das PL/SQL-Objekt, das kompiliert werden soll, und klicken Sie anschließend auf **Compile**. Alternativ können Sie auch die Tasten STRG + UMSCHALT + F9 drücken, um das Objekt zu kompilieren.

Programmeinheiten ausführen



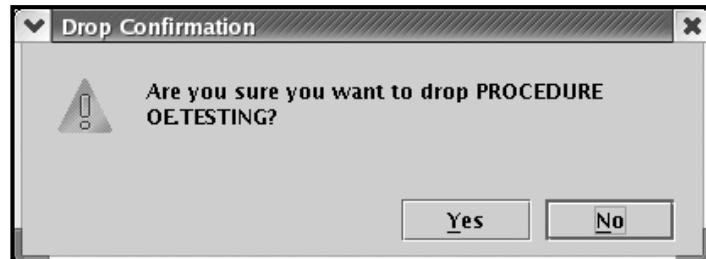
ORACLE®

Copyright © 2007, Oracle. Alle Rechte vorbehalten.

Programmeinheiten ausführen

Um die Programmeinheit auszuführen, klicken Sie zunächst mit der rechten Maustaste auf das Objekt und anschließend auf **Run**. Das Dialogfeld **Run PL/SQL** wird angezeigt. Möglicherweise müssen Sie die NULL-Werte durch sinnvolle Werte ersetzen, die an die Programmeinheit übergeben werden. Nachdem Sie die Werte geändert haben, klicken Sie auf **OK**. Die Ausgabe wird im Fenster **Message-Log** angezeigt.

Programmeinheiten löschen



ORACLE®

Copyright © 2007, Oracle. Alle Rechte vorbehalten.

Programmeinheiten löschen

Um eine Programmeinheit zu löschen, klicken Sie mit der rechten Maustaste auf das Objekt und anschließend auf **Drop**. Das Dialogfeld **Drop Confirmation** wird geöffnet. Klicken Sie auf **Yes**. Das Objekt wird aus der Datenbank gelöscht.

PL/SQL-Programme debuggen

JDeveloper unterstützt zwei Debugging-Typen:

- **Lokal**
- **Remote**

Sie benötigen folgende Privilegien, um das PL/SQL-Debugging ausführen zu können:

- **DEBUG ANY PROCEDURE**
- **DEBUG CONNECT SESSION**

ORACLE®

Copyright © 2007, Oracle. Alle Rechte vorbehalten.

PL/SQL-Programme debuggen

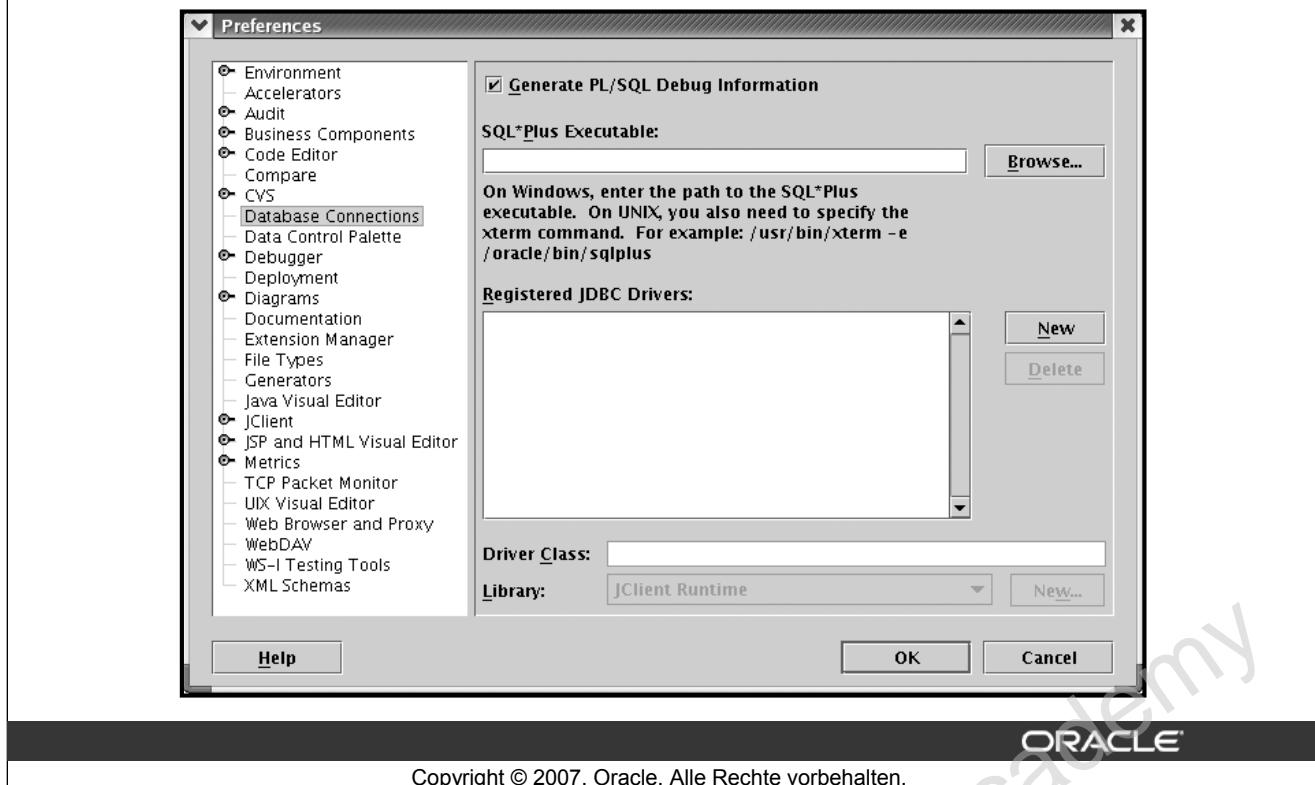
JDeveloper ermöglicht sowohl lokales als auch Remote Debugging. Eine lokale Debugging Session wird gestartet, wenn Sie in Quelldateien Breakpoints festlegen und anschließend den Debugger starten. Für das Remote Debugging sind zwei JDeveloper-Prozesse erforderlich: `debugger` und `debuggee`, die auf verschiedenen Plattformen ausgeführt werden können.

Um ein PL/SQL-Programm zu debuggen, muss es im `INTERPRETED`-Modus kompiliert werden. Sie können kein PL/SQL-Programm debuggen, das im `NATIVE`-Modus kompiliert wurde. Dieser Modus wird in der `init.ora`-Datei der Datenbank eingestellt.

PL/SQL-Programme müssen mit aktivierter `DEBUG`-Option kompiliert werden. Sie haben mehrere Möglichkeiten, diese Option zu aktivieren. Führen Sie in SQL*Plus den Befehl `ALTER SESSION SET PLSQL_DEBUG = true` aus, um die `DEBUG`-Option zu aktivieren. Anschließend können Sie das PL/SQL-Programm, das Sie debuggen möchten, erstellen oder erneut kompilieren. Eine weitere Möglichkeit für die Aktivierung der `DEBUG`-Option ist die Verwendung des folgenden Befehls in SQL*Plus:

```
ALTER <procedure, function, package> <name> COMPILE DEBUG;
```

PL/SQL-Programme debuggen



PL/SQL-Programme debuggen (Fortsetzung)

Stellen Sie vor dem Debuggen des Programms sicher, dass das Kontrollkästchen **Generate PL/SQL Debug Information** aktiviert ist. Sie können das Dialogfeld öffnen, indem Sie auf **Tools > Preferences > Database Connections** klicken.

Von SQL*Plus sind Sie es wahrscheinlich gewohnt, PL/SQL-Funktionen und Prozeduren manuell oder durch Ausführen einer Dummy-Prozedur in der Datenbank zu testen. JDeveloper bietet Ihnen jetzt die Möglichkeit, diese Objekte automatisch zu testen. In diesem Release von JDeveloper können Sie PL/SQL-Programmeinheiten ausführen und debuggen. Beispiel: Sie können übergebene Parameter oder Rückgabewerte einer Funktion angeben. Hierdurch erhalten Sie mehr Kontrolle darüber, was ausgeführt wird, und es werden Ausgabedetails zu den ausgeführten Tests bereitgestellt.

Hinweis: Die Prozeduren oder Funktionen in der Oracle-Datenbank können entweder eigenständig oder Bestandteil eines Packages sein.

PL/SQL-Programme debuggen (Fortsetzung)

So führen Sie Funktionen, Prozeduren und Packages aus oder debuggen sie:

1. Melden Sie sich mit dem Database Wizard bei der Datenbank an.
2. Blenden Sie im Navigator den Knoten **Database** ein, um den Benutzer- und Schemanamen der Datenbank anzuzeigen.
3. Blenden Sie den Knoten **Schema** ein.
4. Blenden Sie je nachdem, was Sie debuggen, den entsprechenden Knoten ein: **Procedure**, **Function** oder **Package body**.
5. (Optional nur für Debugging) Wählen Sie die Funktion, Prozedur oder das Package für das Debugging, und doppelklicken Sie auf die Auswahl, um sie im Code Editor zu öffnen.
6. (Optional nur für Debugging) Legen Sie einen Breakpoint im PL/SQL-Code fest, indem Sie links neben den Rand klicken.

Hinweis: Der Breakpoint muss in einer Zeile mit ausführbarem Code festgelegt werden. Falls der Debugger nicht angehalten wird, haben Sie den Breakpoint möglicherweise nicht in einer Zeile mit ausführbarem Code festgelegt. (Prüfen Sie, ob Sie den Breakpoint korrekt festgelegt haben.) Prüfen Sie zudem, ob die Voraussetzungen zum Debuggen von PL/SQL-Code erfüllt sind. Stellen Sie insbesondere sicher, dass das PL/SQL-Programm im `INTERPRETED`-Modus kompiliert wurde.

7. Vergewissern Sie sich, dass entweder der Code Editor oder die Prozedur im Navigator aktuell ausgewählt ist.
8. Klicken Sie in der Symbolleiste auf die Schaltfläche **Debug**. Wenn Sie das Programm ausführen möchten, ohne es zu debuggen, klicken Sie in der Symbolleiste auf die Schaltfläche **Run**.
9. Das Dialogfeld **Run PL/SQL** wird angezeigt.
 - Wählen Sie ein Ziel, das heißt den Namen der Prozedur oder Funktion, die Sie debuggen möchten. Der Inhalt in den Bereichen **Parameters** und **PL/SQL Block** ändert sich dynamisch, wenn Sie das Ziel ändern.

Hinweis: Sie können Ziele nur dann wählen, wenn Sie ein Package ausführen oder debuggen, das mehr als eine Programmeinheit enthält.

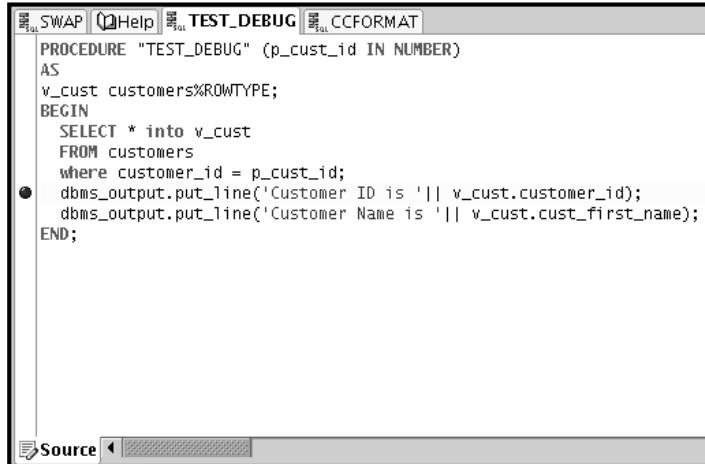
Im Bereich **Parameters** sind die Argumente des Ziels (sofern vorhanden) aufgelistet.

Im Bereich **PL/SQL Block** wird der Code angezeigt, der von JDeveloper für das ausgewählte Ziel generiert wurde. Abhängig davon, was die Funktion oder Prozedur ausführt, müssen Sie die `NULL`-Werte möglicherweise durch sinnvolle Werte ersetzen, so dass sie an die Prozedur, Funktion oder das Package übergeben werden. In einigen Fällen müssen Sie zusätzlichen Code erstellen, um Werte zu initialisieren, die als Argumente übergeben werden. In diesem Fall können Sie den im Bereich **PL/SQL Block** angezeigten Code entsprechend bearbeiten.

10. Klicken Sie auf **OK**, um das Ziel auszuführen oder zu debuggen.
11. Analysieren Sie die ausgegebenen Informationen, die im Log-Fenster angezeigt werden.

Bei Funktionen wird der Rückgabewert angezeigt. Außerdem werden `DBMS_OUTPUT`-Meldungen angezeigt.

Breakpoints festlegen



The screenshot shows the Oracle SQL Developer interface with the PL/SQL Editor open. The window title is "TEST_DEBUG". The code displayed is:

```
PROCEDURE "TEST_DEBUG" (p_cust_id IN NUMBER)
AS
v_cust customers%ROWTYPE;
BEGIN
    SELECT * into v_cust
    FROM customers
    where customer_id = p_cust_id;
    dbms_output.put_line('Customer ID is'|| v_cust.customer_id);
    dbms_output.put_line('Customer Name is'|| v_cust.cust_first_name);
END;
```

A black dot, representing a breakpoint, is placed on the second line of the code. The status bar at the bottom left shows "Source".

ORACLE®

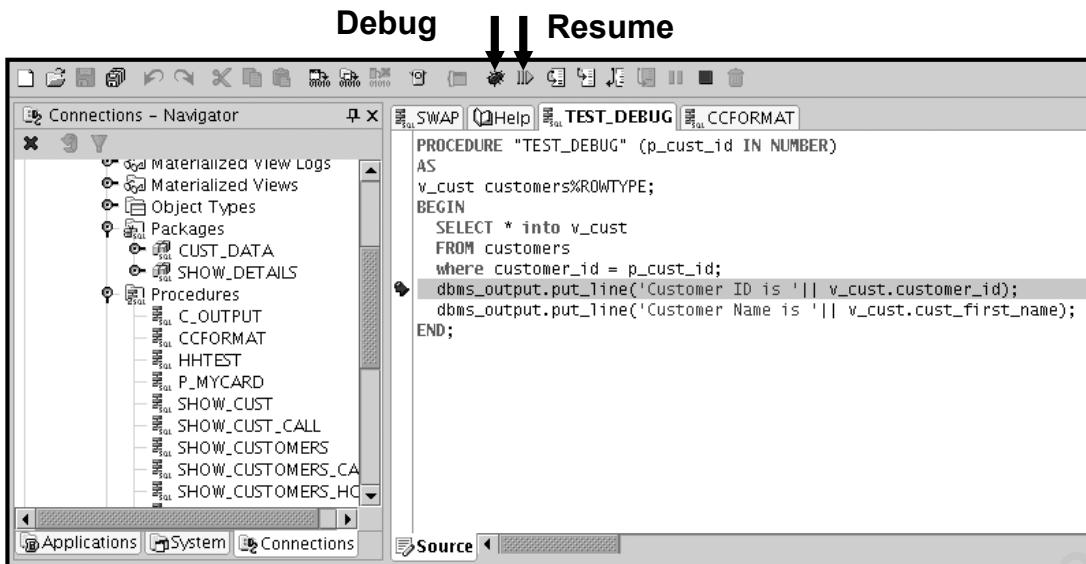
Copyright © 2007, Oracle. Alle Rechte vorbehalten.

Breakpoints festlegen

Breakpoints sind nützlich, um die Werte der Programmvariablen zu untersuchen. Breakpoints sind Trigger in einem Programm, die die Programmausführung am jeweiligen Punkt anhalten, so dass Sie die Werte einiger oder aller Programmvariablen untersuchen können. Wenn Sie Breakpoints in potenziellen Problembereichen des Quell-Codes festlegen, können Sie das Programm so lange ausführen, bis der Punkt erreicht ist, an dem Sie das Debugging starten möchten. Wenn die Programmausführung einen Breakpoint erreicht, wird das Programm angehalten, und der Debugger zeigt die Zeile mit dem Breakpoint im Code Editor an. Sie können anschließend den Programmstatus mit dem Debugger anzeigen. Breakpoints sind flexibel, da Sie sie vor der Programmausführung festlegen können oder während Sie das Programm debuggen.

Um im Code Editor einen Breakpoint festzulegen, klicken Sie neben einer Zeile mit ausführbarem Code auf den linken Rand. Breakpoints, die sich in Kommentarzeilen, Leerzeilen, im Deklarationsbereich oder in anderen Zeilen ohne ausführbaren Code befinden, werden vom Debugger nicht geprüft und als ungültig behandelt.

Code schrittweise ablaufen lassen



ORACLE

Copyright © 2007, Oracle. Alle Rechte vorbehalten.

Code schrittweise ablaufen lassen

Nachdem Sie den Breakpoint festgelegt haben, starten Sie den Debugger, indem Sie auf das Symbol **Debug** klicken. Der Debugger hält die Programmausführung am festgelegten Breakpoint an. Dort können Sie die Variablenwerte prüfen. Sie können die Programmausführung fortsetzen, indem Sie auf das Symbol **Resume** klicken. Der Debugger führt das Programm anschließend bis zum nächsten Breakpoint aus. Nachdem alle Breakpoints ausgeführt worden sind, hält der Debugger die Programmausführung an und zeigt die Ergebnisse im Bereich **Debugging – Log** an.

Variablen untersuchen und ändern

Data		
Name	Value	Type
P_CUST_ID	103	NUMBER
V_CUST		Rowtype

Fenster "Data"

ORACLE

Copyright © 2007, Oracle. Alle Rechte vorbehalten.

Variablen untersuchen und ändern

Wenn der Debugger aktiviert ist, können Sie in den Fenstern **Data**, **Smart Data** und **Watches** Variablenwerte untersuchen und ändern. Sie können die Programmvariablen während einer Debugging Session ändern und haben damit die Möglichkeit, hypothetische Bug-Korrekturen während der Programmausführung zu testen. Wenn Sie feststellen, dass ein Programmfehler durch eine Korrektur behoben werden kann, können Sie die Debugging Session beenden, den Programm-Code entsprechend korrigieren und das Programm erneut kompilieren, um die Korrektur dauerhaft zu speichern.

Im Fenster **Data** können Sie Informationen zu Programmvariablen anzeigen. Im Fenster **Data** werden die Argumente, lokalen Variablen und statischen Felder für den aktuellen Kontext angezeigt. Diese Anzeige wird durch die Auswahl im Fenster **Stack** gesteuert. Wenn Sie einen neuen Kontext verwenden, wird das Fenster **Data** aktualisiert, um die Daten für den neuen Kontext anzuzeigen. Falls das aktuelle Programm ohne Debug-Informationen kompiliert wurde, werden die lokalen Variablen nicht angezeigt.

Variablen untersuchen und ändern

Name	Value	Type
v_cust		Rowtype
v_cust.customer_id	103	NUMBER(6,0)

Fenster "Smart Data"

ORACLE®

Copyright © 2007, Oracle. Alle Rechte vorbehalten.

Variablen untersuchen und ändern (Fortsetzung)

Anders als das Fenster **Data**, in dem alle Programmvariablen angezeigt werden, zeigt das Fenster **Smart Data** nur die Daten an, die für den schrittweise ablaufenden Quell-Code relevant sind.

Variablen untersuchen und ändern

Watches		
Name	Value	Type
v_cust.customer_103		NUMBER(6,0)

Fenster "Watches"

ORACLE®

Copyright © 2007, Oracle. Alle Rechte vorbehalten.

Variablen untersuchen und ändern (Fortsetzung)

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

Variablen untersuchen und ändern

Class	Method
<input type="checkbox"/> TEST_DEBUG	TEST_DEBUG
<input type="checkbox"/> ANONYMOUS	BLOCK

Fenster "Stack"

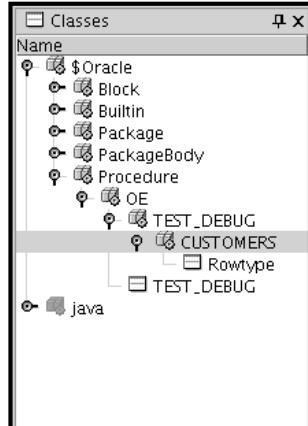
ORACLE

Copyright © 2007, Oracle. Alle Rechte vorbehalten.

Variablen untersuchen und ändern (Fortsetzung)

Sie können das Fenster **Stack** aktivieren, indem Sie auf View > Debugger > Stack klicken. Der Aufruf-Stack für den aktuellen Thread wird angezeigt. Wenn Sie im Fenster **Stack** eine Zeile markieren, werden das Fenster **Data**, das Fenster **Watches** und alle anderen Fenster aktualisiert, um die Daten für die ausgewählte Klasse anzuzeigen.

Variablen untersuchen und ändern



Fenster "Classes"

ORACLE

Copyright © 2007, Oracle. Alle Rechte vorbehalten.

Variablen untersuchen und ändern (Fortsetzung)

Im Fenster **Classes** werden alle Klassen angezeigt, die aktuell zum Ausführen des Programms geladen werden. Wenn Oracle Java Virtual Machine (OJVM) aktiviert ist, werden in diesem Fenster auch die Anzahl der Instanzen einer Klasse sowie der von diesen Instanzen verwendete Speicher angezeigt.

REF-Cursor

ORACLE®

Copyright © 2007, Oracle. Alle Rechte vorbehalten.

Oracle Internal & Oracle Academy
Use Only

Cursor-Variablen

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

ORACLE

Copyright © 2007, Oracle. Alle Rechte vorbehalten.

Cursor-Variablen

Cursor-Variablen sind mit Zeigern in C und Pascal vergleichbar. Sie enthalten anstelle des eigentlichen Elements die Speicheradresse des Elements. Daher wird kein Element, sondern ein Zeiger erstellt, wenn Sie eine Cursor-Variable deklarieren. In PL/SQL haben Zeiger den Datentyp REF X. REF ist die Abkürzung von REFERENCE, X steht für eine Klasse von Objekten. Eine Cursor-Variable hat den Datentyp REF CURSOR.

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

Cursor-Variablen sind für jeden PL/SQL-Client verfügbar. Sie können eine Cursor-Variable beispielsweise in einer PL/SQL-Host-Umgebung wie einem OCI- oder Pro*C-Programm deklarieren und dann als Eingabe-Host-Variable (Bind-Variable) an PL/SQL übergeben. Darüber hinaus können Tools zur Anwendungsentwicklung (z. B. Oracle Forms und Oracle Reports), die eine PL/SQL-Engine enthalten, Cursor-Variablen vollständig auf der Client-Seite verwenden. Der Oracle-Server verfügt ebenfalls über eine PL/SQL-Engine. Mit Hilfe von Remote Procedure Calls (RPCs) können Sie Cursor-Variablen zwischen Anwendungen und Servern übergeben.

Cursor-Variablen

- **Mit Hilfe von Cursor-Variablen können Sie Abfrageergebnisse zwischen gespeicherten PL/SQL-Unterprogrammen und verschiedenen Clients übergeben.**
- **PL/SQL kann einen Zeiger auf den Arbeitsbereich, in dem die Ergebnismenge der Abfrage gespeichert ist, gemeinsam nutzen.**
- **Sie können den Wert einer Cursor-Variablen nach Belieben aus einem Gültigkeitsbereich an einen anderen übergeben.**
- **Sie können den Netzwerkverkehr reduzieren, indem ein PL/SQL-Block mehrere Host-Cursor-Variablen in einem Roundtrip öffnet oder schließt.**

ORACLE

Copyright © 2007, Oracle. Alle Rechte vorbehalten.

Cursor-Variablen

Mit Hilfe von Cursor-Variablen können Sie Abfrageergebnisse zwischen gespeicherten PL/SQL-Unterprogrammen und verschiedenen Clients übergeben. PL/SQL und die zugehörigen Clients sind nicht die Eigentümer der Ergebnismenge. Sie teilen sich lediglich einen Zeiger auf den Arbeitsbereich der Abfrage, in dem die Ergebnismenge gespeichert ist. Ein OCI-Client, eine Oracle Forms-Anwendung und der Oracle-Server beispielsweise können auf denselben Arbeitsbereich verweisen.

Auf den Arbeitsbereich einer Abfrage kann so lange zugegriffen werden, wie eine Cursor-Variable auf ihn zeigt. Sie können daher den Wert einer Cursor-Variablen nach Belieben aus einem Gültigkeitsbereich an einen anderen übergeben. Wenn Sie z. B. eine Host-Cursor-Variable 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 Cursor-Variable zeigt.

Wenn auf Client-Seite eine PL/SQL-Engine vorhanden ist, sind Aufrufe vom Client an den Server keinen Einschränkungen unterworfen. Sie können beispielsweise eine Cursor-Variable auf Client-Seite deklarieren, sie auf Server-Seite öffnen und lesen und den Lesevorgang auf Client-Seite fortsetzen. Darüber hinaus können Sie den Netzwerkverkehr reduzieren, indem ein PL/SQL-Block mehrere Host-Cursor-Variablen in einem Roundtrip öffnet oder schließt.

Cursor-Variablen adressieren die PGA nicht mit einem statischen Namen, sondern enthalten Verweise auf Cursor-Arbeitsbereiche in der PGA. Da Sie die den Bereich mit einem Verweis adressieren, 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];
```

Cursor-Variable diesen Typs deklarieren:

```
ref_cv ref_type_name;
```

Beispiel:

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

ORACLE

Copyright © 2007, Oracle. Alle Rechte vorbehalten.

REF CURSOR-Typen definieren

Um einen REF CURSOR zu definieren, führen Sie zwei Schritte aus. Zuerst definieren Sie einen REF CURSOR-Typ, anschließend deklarieren Sie Cursor-Variablen 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];
```

Hierbei gilt:

ref_type_name Typspezifikation, die in nachfolgenden Deklarationen von Cursor-Variablen verwendet wird
return_type Record oder Zeile in einer Datenbanktabelle

Im folgenden Beispiel geben Sie einen Rückgabetyp an, der eine Zeile in der DEPARTMENT-Datenbanktabelle repräsentiert.

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 jedoch nicht:

```
DECLARE
```

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

```
  TYPE GenericCurTyp IS REF CURSOR; -- weak
```

REF CURSOR-Typen definieren (Fortsetzung)

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

Cursor-Variablen deklarieren

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

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

Hinweis: Sie können Cursor-Variablen nicht in einem Package deklarieren. Im Gegensatz zu Variablen, die in einem Package integriert sind, haben Cursor-Variablen keinen beständigen Status. Daher wird kein Element, sondern ein Zeiger erstellt, wenn Sie eine Cursor-Variable deklarieren. Cursor-Variablen 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 im folgenden Beispiel mit %ROWTYPE einen Record-Typ angeben, der für eine Zeile steht, die von einer Cursor-Variablen 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
```

Gleichermaßen können Sie %TYPE verwenden, um entsprechend dem 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
```

Cursor-Variablen als Parameter

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

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

Anweisungen OPEN-FOR, FETCH und CLOSE

- Die **OPEN-FOR-Anweisung** ordnet einer Multiple Row-Abfrage eine Cursor-Variable zu, führt die Abfrage aus, identifiziert die Ergebnismenge und positioniert den Cursor so, dass er auf die erste Zeile in der Ergebnismenge zeigt.
- 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.
- Die **CLOSE-Anweisung** deaktiviert eine Cursor-Variable.

ORACLE

Copyright © 2007, Oracle. Alle Rechte vorbehalten.

Anweisungen OPEN-FOR, FETCH und CLOSE

Dynamische Multiple Row-Abfragen verarbeiten Sie mit drei Anweisungen: OPEN-FOR, FETCH und CLOSE. Zunächst "öffnen" (OPEN) Sie eine Cursor-Variable "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 Cursor-Variable.

Cursor-Variablen öffnen

Die OPEN-FOR-Anweisung ordnet einer Multiple Row-Abfrage eine Cursor-Variable 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 in %ROWCOUNT enthaltenen Zähler der verarbeiteten Zeilen auf null zurück. Im Gegensatz zum statischen Format von OPEN-FOR enthält das dynamische Format eine USING-Klausel. Zur Laufzeit ersetzen Bind-Argumente in der USING-Klausel entsprechende Platzhalter in der dynamischen SELECT-Anweisung. Die Syntax lautet:

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

Dabei gilt: CURSOR_VARIABLE ist eine Cursor-Variable mit schwacher Typdefinition (ohne Rückgabetyp), HOST_CURSOR_VARIABLE eine in einer PL/SQL-Host-Umgebung wie einem OCI-Programm deklarierte Cursor-Variable und dynamic_string ein Zeichenfolgenausdruck, der eine Abfrage repräsentiert, die mehrere Zeilen liefert.

Anweisungen OPEN-FOR, FETCH und CLOSE (Fortsetzung)

Die Syntax im folgenden Beispiel deklariert eine Cursor-Variable, die anschließend einer dynamischen SELECT-Anweisung zugeordnet wird, die Zeilen aus der employees-Tabelle 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 Cursor-Variable geöffnet ist. Um also Zeilen mit Hilfe verschiedener Bind-Werte aus dem Cursor zu lesen, müssen Sie die Cursor-Variable erneut öffnen, nachdem die Bind-Argumente auf neue Werte festgelegt wurden.

Daten aus einer Cursor-Variablen abrufen

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 in %ROWCOUNT gespeicherten Zähler und rückt den Cursor in die nächste Zeile vor. Verwenden Sie die folgende Syntax:

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

Lesen Sie als Fortsetzung des Beispiels Zeilen aus der Cursor-Variablen 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 von der Abfrage im Zusammenhang mit dem zugeordneten Cursor zurückgegeben wird, muss eine entsprechende, typkompatible Variable oder ein entsprechendes Feld in der INTO-Liste vorhanden sein. Sie können für verschiedene FETCH-Anweisungen mit derselben Cursor-Variablen 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 Cursor-Variable zu lesen, löst PL/SQL die vordefinierte Exception INVALID_CURSOR aus.

Anweisungen OPEN-FOR, FETCH und CLOSE (Fortsetzung)

Cursor-Variablen schließen

Die CLOSE-Anweisung deaktiviert eine Cursor-Variable. Die zugehörige Ergebnismenge ist dann undefiniert. Verwenden Sie die folgende Syntax:

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

In vorliegenden Beispiel soll nach Verarbeitung der letzten Zeile die Cursor-Variable `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 Cursor-Variable zu schließen, löst PL/SQL `INVALID_CURSOR` aus.

Fetch-Vorgang – 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 © 2007, Oracle. Alle Rechte vorbehalten.

Fetch-Vorgang – Beispiel

Das Beispiel auf der Folie zeigt, dass Sie Zeilen aus der Ergebnismenge einer dynamischen Multiple Row-Abfrage in einen Record einlesen können. Zunächst müssen Sie den REF CURSOR-Typ EmpCurTyp definieren. Anschließend definieren Sie eine Cursor-Variable emp_cv des Typs EmpcurTyp. Im ausführbaren Bereich des PL/SQL-Blockes ordnet die OPEN-FOR-Anweisung die Cursor-Variable 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 in der INTO-Klausel der EMP_REC-Variablen zu. Schließen Sie nach Verarbeitung der letzten Zeile die Cursor-Variable EMP_CV.

Zusätzliche Übungen

Oracle Internal & Oracle Academy
Use Only

Zusätzliche Übungen – Überblick

Diese zusätzlichen Übungen sind eine Ergänzung zum Kurs *Oracle Database 11g: PL/SQL-Grundlagen*. Sie wenden in den Übungen die im Kurs beschriebenen Konzepte an und erwerben zusätzliche praktische Erfahrungen im Deklarieren von Variablen, Erstellen von ausführbaren Anweisungen, Interagieren mit dem Oracle-Server, Erstellen von Kontrollstrukturen, Arbeiten mit zusammengesetzten Datentypen und Cursors und Behandeln von Exceptions. In diesem Übungsteil werden u. a. die Tabellen `employees`, `jobs`, `job_history` und `departments` verwendet.

Oracle Internal & Oracle Academy
Use Only

Zusätzliche Übungen 1 und 2

Hinweis: Diese Übungen vermitteln zusätzliche praktische Erfahrungen beim Deklarieren von Variablen und Erstellen von ausführbaren Anweisungen.

1. Prüfen Sie folgende Deklarationen. Bestimmen Sie, welche Deklarationen nicht zulässig sind, und erläutern Sie warum.

- a.

```
DECLARE
    name,dept      VARCHAR2(14);
```
- b.

```
DECLARE
    test           NUMBER(5);
```
- c.

```
DECLARE
    MAXSALARY      NUMBER(7,2) = 5000;
```
- d.

```
DECLARE
    JOINDATE       BOOLEAN := SYSDATE;
```

2. Bestimmen Sie in den folgenden Zuweisungen jeweils den Datentyp des resultierenden Ausdrucks.

- a.

```
email := firstname || to_char(empno);
```
- b.

```
confirm := to_date('20-JAN-1999', 'DD-MON-YYYY');
```
- c.

```
sal := (1000*12) + 500
```
- d.

```
test := FALSE;
```
- e.

```
temp := temp1 < (temp2/ 3);
```
- f.

```
var := sysdate;
```

Zusätzliche Übung 3

```
3. DECLARE
    v_custid      NUMBER(4) := 1600;
    v_custname     VARCHAR2(300) := 'Women Sports Club';
    v_new_custid   NUMBER(3) := 500;

BEGIN
DECLARE
    v_custid      NUMBER(4) := 0;
    v_custname     VARCHAR2(300) := 'Shape up Sports Club';
    v_new_custid   NUMBER(3) := 300;
    v_new_custname VARCHAR2(300) := 'Jansports Club';

BEGIN
    v_custid := v_new_custid;
    v_custname := v_custname || ' ' || v_new_custname;
    END;
    v_custid := (v_custid *12) / 10;
    END;
    /

```

1

END;

2

END;

Prüfen Sie den oben angegebenen PL/SQL-Block, und bestimmen Sie den Datentyp und den Wert jeder der folgenden Variablen entsprechend den Regeln für Gültigkeitsbereiche:

- Wert von v_custid an der 1. Position:
- Wert von v_custname an der 1. Position:
- Wert von v_new_custid an der 2. Position:
- Wert von v_new_custname an der 1. Position:
- Wert von v_custid an der 2. Position:
- Wert von v_custname an der 2. Position:

Zusätzliche Übungen 4 und 5

Hinweis: In diesen Übungen erwerben Sie zusätzliche praktische Erfahrungen beim Interagieren mit dem Oracle-Server und beim Erstellen von Kontrollstrukturen.

- Erstellen Sie einen PL/SQL-Block, der ein Jahr annimmt, und prüfen Sie, ob es sich um ein Schaltjahr handelt. Wenn Sie beispielsweise 1990 als Jahr eingeben, sollten Sie die Ausgabe "1990 is not a leap year" erhalten.

Tipp: Schaltjahre sind alle Jahre, deren zwei Endziffern durch 4 teilbar sind, außer den nicht durch 400 teilbaren Jahrhundertjahren (1700, 1800, 1900).

Zusätzliche Übungen 4 und 5

Testen Sie Ihre Lösung mit den folgenden Jahren:

1990	Kein Schaltjahr
2000	Schaltjahr
1996	Schaltjahr
1886	Kein Schaltjahr
1992	Schaltjahr
1824	Schaltjahr

```
anonymous block completed  
1990 is not a leap year|
```

5. a. Erstellen Sie für die folgende Übungen eine temporäre Tabelle, in der die Ergebnisse gespeichert werden. Sie können die Tabelle entweder manuell oder mit dem Skript lab_ap_05.sql automatisch erstellen. Erstellen Sie eine TEMP-Tabelle mit den folgenden drei Spalten:

Spaltenname	NUM_STORE	CHAR_STORE	DATE_STORE
Schlüsseltyp			
Null/Unique-Werte			
FK-Tabelle			
FK-Spalte			
Datentyp	Number	VARCHAR2	Date
Länge	7,2	35	

- b. Erstellen Sie einen PL/SQL-Block, der die zwei Variablen V_MESSAGE und V_DATE_WRITTEN enthält. Deklarieren Sie V_MESSAGE als VARCHAR2-Datentyp mit einer Länge von 35 und V_DATE_WRITTEN als DATE-Datentyp. Weisen Sie den Variablen die folgenden Werte zu:

Variable	Inhalt
V_MESSAGE	This is my first PL/SQL program
V_DATE_WRITTEN	Aktuelles Datum

Speichern Sie die Werte in den entsprechenden Spalten der TEMP-Tabelle. Prüfen Sie Ihre Ergebnisse, indem Sie die TEMP-Tabelle abfragen.

NUM_STORE	CHAR_STORE	DATE_STORE
(null)	This is my first PLSQL Program	27-JUN-07

Zusätzliche Übungen 6 und 7

6. a. Speichern Sie eine Abteilungsnummer in einer Austauschvariablen.
b. Erstellen Sie einen PL/SQL-Block, der die Anzahl der Mitarbeiter dieser Abteilung ausgibt.

```
anonymous block completed  
6 employee(s) work for department number 30
```

7. Erstellen Sie einen PL/SQL-Block, um eine Variable namens `sal` zu deklarieren, in der das Gehalt eines Mitarbeiters gespeichert wird. Gehen Sie im ausführbaren Teil des Programms wie folgt vor:
 - a. Speichern Sie einen Mitarbeiternamen in einer Austauschvariablen.
 - b. Speichern Sie das Gehalt des Mitarbeiters in der `v_sal`-Variablen.
 - c. Wenn das Gehalt unter 3000 liegt, erhöhen Sie den Betrag um 500, und zeigen Sie im Fenster die Meldung "<Employee Name>'s salary updated" an.
 - d. Wenn das Gehalt über 3000 liegt, geben Sie das Gehalt des Mitarbeiters in diesem Format an: "<Employee Name> earns".
 - e. Testen Sie den PL/SQL-Block mit folgenden Nachnamen:

LAST_NAME	SALARY
Pataballa	4800
Greenberg	12000
Ernst	6000

Zusätzliche Übungen 8 und 9

8. Erstellen Sie einen PL/SQL-Block, der das Gehalt eines Mitarbeiters in einer Austauschvariablen speichert.

Gehen Sie im ausführbaren Teil des Programms wie folgt vor:

- Berechnen Sie das Jahresgehalt als "Gehalt mal 12".
- Berechnen Sie die Prämie gemäß den folgenden Angaben:

Jahresgehalt	Prämie
>= 20,000	2,000
19,999 - 10,000	1,000
<= 9,999	500

Zeigen Sie die Höhe der Prämie im folgenden Format im Fenster an:

"The bonus is \$....."

- Testen Sie den PL/SQL-Block mit folgenden Werten:

SALARY	BONUS
5000	2000
1000	1000
15000	2000

Hinweis: In diesen Übungen sammeln Sie praktische Erfahrungen beim Arbeiten mit zusammengesetzten Datentypen und Cursorn sowie beim Behandeln von Exceptions.

9. a. Führen Sie das Skript lab_ap_09_a.sql aus, um eine temporäre Tabelle namens emp zu erstellen. Erstellen Sie einen PL/SQL-Block, um eine Personalnummer, die neue Abteilungsnummer und die Erhöhung des Gehalts in Prozent in Austauschvariablen zu speichern.
- b. Aktualisieren Sie die Abteilungsnummer des Mitarbeiters durch die neue Abteilungsnummer und das Gehalt durch das neue Gehalt. Speichern Sie die Aktualisierungen in der emp-Tabelle. Zeigen Sie nach Abschluss der Aktualisierung die Meldung "Update complete" im Fenster an. Wenn keine übereinstimmenden Blöcke gefunden werden, zeigen Sie die Meldung "No Data Found" an. Testen Sie den PL/SQL-Block mit folgenden Werten:

EMPLOYEE_ID	NEW_DEPARTMENT_ID	% INCREASE	MESSAGE
100	20	2	Update Complete
10	30	5	No Data found
126	40	3	Update Complete

Zusätzliche Übungen 10 und 11

10. Erstellen Sie einen PL/SQL-Block, um den `EMP_CUR`-Cursor zu deklarieren und den Mitarbeiternamen, das Gehalt und das Einstellungsdatum in der `employees`-Tabelle auszuwählen. Verarbeiten Sie alle Zeilen im Cursor. Wenn das Gehalt höher als 15000 ist und das Einstellungsdatum nach dem 01-FEB-1988 liegt, zeigen Sie den Mitarbeiternamen, das Gehalt und das Einstellungsdatum im Fenster an. Berücksichtigen Sie das in der Beispielausgabe unten gezeigte Format:

```
anonymous block completed
Kochhar earns 17000 and joined the organization on 21-SEP-89
De Haan earns 17000 and joined the organization on 13-JAN-93
```

11. Erstellen Sie einen PL/SQL-Block, um den Nachnamen und die Abteilungsnummer der Mitarbeiter, deren `EMPLOYEE_ID` unter 114 liegt, aus der `EMPLOYEES`-Tabelle abzurufen. Füllen Sie mit den abgerufenen Werten zwei PL/SQL-Tabellen. Speichern Sie in einer Tabelle die Records der Nachnamen und in der zweiten Tabelle die Records der Abteilungsnummern. Rufen Sie mit einer Schleife die Informationen zu Mitarbeiternamen und Gehalt aus den PL/SQL-Tabellen ab, und zeigen Sie sie im Fenster an. Verwenden Sie hierfür `DBMS_OUTPUT.PUT_LINE`. Zeigen Sie diese Details für die ersten 15 Mitarbeiter der PL/SQL-Tabellen an.

```
anonymous block completed
Employee Name: King Department_id: 90
Employee Name: Kochhar Department_id: 90
Employee Name: De Haan Department_id: 90
Employee Name: Hunold Department_id: 60
Employee Name: Ernst Department_id: 60
Employee Name: Austin Department_id: 60
Employee Name: Pataballa Department_id: 60
Employee Name: Lorentz Department_id: 60
Employee Name: Greenberg Department_id: 100
Employee Name: Faviet Department_id: 100
Employee Name: Chen Department_id: 100
Employee Name: Sciarra Department_id: 100
Employee Name: Urman Department_id: 100
Employee Name: Popp Department_id: 100
Employee Name: Raphaely Department_id: 30
```

Zusätzliche Übungen 12, 13 und 14

12. a. Erstellen Sie einen PL/SQL-Block, der einen Cursor namens DATE_CUR deklariert. Übergeben Sie an den Cursor einen Parameter vom Datentyp DATE, und geben Sie die Details aller Mitarbeiter aus, die nach diesem Datum eingestellt wurden.

```
DEFINE B_HIREDATE = 08-MAR-00
```

- b. Testen Sie den PL/SQL-Block mit folgenden Datumsangaben der Einstellung: 08-MAR-00, 25-JUN-97, 28-SEP-98, 07-FEB-99.

```
anonymous block completed
166 Ande 24-MAR-00
167 Banda 21-APR-00
173 Kumar 21-APR-00
```

13. Führen Sie das Skript lab_ap_09_a.sql aus, um die emp-Tabelle erneut zu erstellen. Erstellen Sie einen PL/SQL-Block, um Sachbearbeiter, die mehr als 3000 verdienen, der Tätigkeitskennung SR CLERK zuzuweisen und ihr Gehalt um 10 % zu erhöhen. Verwenden Sie für diese Übung die EMP-Tabelle. Prüfen Sie Ihre Ergebnisse, indem Sie die emp-Tabelle abfragen.

Tipp: Verwenden Sie einen Cursor mit der Syntax FOR UPDATE und CURRENT OF.

14. a. Für die folgende Übung benötigen Sie eine Tabelle, in der die Ergebnisse gespeichert werden. Sie können die analysis-Tabelle manuell oder mit dem Skript lab_ap_14_a.sql automatisch erstellen. Erstellen Sie eine analysis-Tabelle mit den folgenden drei Spalten:

Spaltenname	ENAME	YEARS	SAL
Schlüsseltyp			
Null/Unique-Werte			
FK-Tabelle			
FK-Spalte			
Datentyp	VARCHAR2	Number	Number
Länge	20	2	8,2

- b. Erstellen Sie einen PL/SQL-Block, um die analysis-Tabelle mit den Informationen aus der employees-Tabelle zu füllen. Speichern Sie den Nachnamen eines Mitarbeiters in einer Austauschvariablen.

Zusätzliche Übungen 12, 13 und 14 (Fortsetzung)

- c. Fragen Sie in der employees-Tabelle ab, ob der betreffende Mitarbeiter länger als fünf Jahre bei der Organisation tätig ist. Wenn das Gehalt gleichzeitig unter 3500 liegt, lösen Sie eine Exception aus. Behandeln Sie die Exception mit einem geeigneten Exception-Handler, der die folgenden Werte in die analysis-Tabelle einfügt: Nachname, Dienstjahre und aktuelles Gehalt des Mitarbeiters. Zeigen Sie andernfalls die Meldung Not due for a raise im Fenster an. Prüfen Sie die Ergebnisse, indem Sie die analysis-Tabelle abfragen. Testen Sie den PL/SQL-Block mit den folgenden Werten:

LAST_NAME	MESSAGE
Austin	Not due for a raise
Nayer	Due for a raise
Fripp	Not due for a raise
Khoo	Due for a raise

Oracle Internal & Oracle Academy
Use Only

Zusätzliche Übungen – Lösungen

Oracle Internal & Oracle Academy
Use Only

Zusätzliche Übungen 1 und 2 – Lösungen

1. Beurteilen Sie die folgenden Deklarationen. Ermitteln Sie, welche Deklarationen *nicht* zulässig sind, und erläutern Sie den Grund.

a. DECLARE

```
name,dept      VARCHAR2(14);
```

Unzulässig, da nur ein Identifier pro Deklaration zulässig ist.

b. DECLARE

```
test          NUMBER(5);
```

Zulässig

c. DECLARE

```
MAXSALARY     NUMBER(7,2) = 5000;
```

Unzulässig, da der Zuweisungsoperator falsch ist. Der richtige Operator ist :=.

d. DECLARE

```
JOINDATE      BOOLEAN := SYSDATE;
```

Unzulässig, da die Datentypen nicht übereinstimmen. Booleschen Datentypen kann kein Datumswert zugewiesen werden. Als Datentyp muss DATE verwendet werden.

2. Bestimmen Sie in den folgenden Zuweisungen jeweils den Datentyp des resultierenden Ausdrucks.

a. email := firstname || to_char(empno);

Char

b. confirm := to_date('20-JAN-1999', 'DD-MON-YYYY');

Date

c. sal := (1000*12) + 500

Number

d. test := FALSE;

Boolean

e. temp := temp1 < (temp2/3);

Boolean

f. var := sysdate;

Date

Zusätzliche Übung 3 – Lösungen

```
3. DECLARE
    v_custid    NUMBER(4) := 1600;
    v_custname      VARCHAR2(300) := 'Women Sports Club';
    v_new_custid    NUMBER(3) := 500;
BEGIN
DECLARE
    v_custid    NUMBER(4) := 0;
    v_custname      VARCHAR2(300) := 'Shape up Sports Club';
    v_new_custid    NUMBER(3) := 300;
    v_new_custname    VARCHAR2(300) := 'Jansports Club';
BEGIN
    v_custid := v_new_custid;
    v_custname := v_custname || ' ' || v_new_custname;
    END;
    v_custid := (v_custid *12) / 10;
END;
```

1 → v_custid := v_new_custid;
1 → v_custname := v_custname || ' ' || v_new_custname;
2 → v_custid := (v_custid *12) / 10;

Werten Sie den oben aufgeführten PL/SQL-Block aus, und bestimmen Sie Datentyp und Wert der einzelnen Variablen entsprechend den Regeln für Gültigkeitsbereiche:

- a. Wert von v_custid an 1. Position:

300. Der Datentyp ist NUMBER.

- b. Wert von v_custname an 1. Position:

Shape up Sports Club Jansports Club. Der Datentyp ist VARCHAR2 .

- c. Wert von v_new_custid an 1. Position:

500. Der Datentyp ist NUMBER (oder INTEGER).

- d. Wert von v_new_custname an 1. Position:

Jansports Club. Der Datentyp ist VARCHAR2.

- e. Wert von v_custid an 2. Position:

1920. Der Datentyp ist NUMBER.

- f. Wert von v_custname an 2. Position:

Women Sports Club. Der Datentyp ist VARCHAR2.

Zusätzliche Übung 4 – Lösungen

4. Erstellen Sie einen PL/SQL-Block, der ein Jahr annimmt, und prüfen Sie, ob es sich um ein Schaltjahr handelt. Wenn Sie beispielsweise 1990 als Jahr eingeben, sollten Sie die Ausgabe "1990 is not a leap year" erhalten.

Tipp: Schaltjahre sind alle Jahre, deren zwei Endziffern durch 4 teilbar sind, außer den nicht durch 400 teilbaren Jahrhundertdaten (1700, 1800, 1900).

Testen Sie Ihre Lösung mit den folgenden Jahren:

1990	Kein Schaltjahr
2000	Schaltjahr
1996	Schaltjahr
1886	Kein Schaltjahr
1992	Schaltjahr
1824	Schaltjahr

```
DECLARE
    v_YEAR NUMBER(4) := &P_YEAR;
    v_REMAINDER1 NUMBER(5,2);
    v_REMAINDER2 NUMBER(5,2);
    v_REMAINDER3 NUMBER(5,2);

BEGIN
    v_REMAINDER1 := MOD(v_YEAR,4);
    v_REMAINDER2 := MOD(v_YEAR,100);
    v_REMAINDER3 := MOD(v_YEAR,400);
    IF ((v_REMAINDER1 = 0 AND v_REMAINDER2 <> 0 ) OR
v_REMAINDER3 = 0) THEN
        DBMS_OUTPUT.PUT_LINE(v_YEAR || ' is a leap year');
    ELSE
        DBMS_OUTPUT.PUT_LINE (v_YEAR || ' is not a leap year');
    END IF;
END ;
/
```

Zusätzliche Übung 5 – Lösungen

5. a. Erstellen Sie für die folgenden Übungen eine temporäre Tabelle, in der die Ergebnisse gespeichert werden.

Sie können die Tabelle entweder manuell oder mit dem Skript lab_ap_05.sql automatisch erstellen. Erstellen Sie eine TEMP-Tabelle mit den folgenden drei Spalten:

Spaltenname	NUM_STORE	CHAR_STORE	DATE_STORE
Schlüsseltyp			
Null/Unique-Werte			
FK -Tabelle			
FK-Spalte			
Datentyp	Number	VARCHAR2	Date
Länge	7 , 2	35	

```
CREATE TABLE temp
(num_store NUMBER(7,2),
char_store VARCHAR2(35),
date_store DATE);
```

- b. Erstellen Sie einen PL/SQL-Block, der die zwei Variablen V_MESSAGE und V_DATE_WRITTEN enthält. Deklarieren Sie V_MESSAGE als VARCHAR2-Datentyp mit einer Länge von 35 und V_DATE_WRITTEN als DATE-Datentyp. Weisen Sie den Variablen folgende Werte zu:

Variable	Inhalt
V_MESSAGE	This is my first PL/SQL program
V_DATE_WRITTEN	Aktuelles Datum

Speichern Sie die Werte in den entsprechenden Spalten der TEMP-Tabelle. Prüfen Sie Ihre Ergebnisse, indem Sie die TEMP-Tabelle abfragen.

```
DECLARE
    V_MESSAGE VARCHAR2(35);
    V_DATE_WRITTEN DATE;
BEGIN
    V_MESSAGE := 'This is my first PLSQL Program';
    V_DATE_WRITTEN := SYSDATE;
    INSERT INTO temp(CHAR_STORE,DATE_STORE)
    VALUES (V_MESSAGE,V_DATE_WRITTEN);
END;
/
SELECT * FROM TEMP;
```

Zusätzliche Übungen 6 und 7 – Lösungen

6. a. Speichern Sie eine Abteilungsnummer in einer Austauschvariablen.

```
DEFINE P_DEPTNO = 30
```

- b. Erstellen Sie einen PL/SQL-Block, der die Anzahl der Mitarbeiter dieser Abteilung ausgibt.

Tipp: Aktivieren Sie DBMS_OUTPUT in SQL Developer.

```
DECLARE
    V_HOWMANY NUMBER(3);
    V_DEPTNO DEPARTMENTS.department_id%TYPE := &P_DEPTNO;
BEGIN
    SELECT COUNT(*) INTO V_HOWMANY FROM employees
        WHERE department_id = V_DEPTNO;
    DBMS_OUTPUT.PUT_LINE (V_HOWMANY || ' employee(s) work
        for department number ' || V_DEPTNO);
END;
/
```

7. Erstellen Sie einen PL/SQL-Block, um eine Variable namens sal zu deklarieren, in der das Gehalt eines Mitarbeiters gespeichert wird. Gehen Sie im ausführbaren Teil des Programms wie folgt vor:

- a. Speichern Sie einen Mitarbeiternamen in einer Austauschvariablen:

```
DEFINE P_LASTNAME = Pataballa
```

- b. Speichern Sie das Gehalt in der sal-Variablen.

- c. Liegt das Gehalt des Mitarbeiters unter 3000, erhöhen Sie den Betrag um 500, und zeigen Sie im Fenster die Meldung "<Employee Name>'s salary updated" an.

- d. Liegt das Gehalt über 3000, geben Sie den Betrag im Format "<Employee Name> earns" aus.

- e. Testen Sie den PL/SQL-Block mit weiteren Nachnamen.

Hinweis: Löschen Sie am Ende des Skripts die Variable, in der der Mitarbeitername gespeichert ist.

LAST_NAME	SALARY
Pataballa	4800
Greenberg	12000
Ernst	6000

Zusätzliche Übungen 7 und 8 – Lösungen

```
DECLARE
    V_SAL NUMBER(7,2);
    V_LASTNAME EMPLOYEES.LAST_NAME%TYPE;
BEGIN
    SELECT salary INTO V_SAL
    FROM employees WHERE last_name = INITCAP('&P_LASTNAME')
        FOR UPDATE OF salary;
    V_LASTNAME := INITCAP('&P_LASTNAME');
    IF V_SAL < 3000 THEN
        UPDATE employees SET salary = salary + 500
        WHERE last_name = INITCAP('&P_LASTNAME') ;
        DBMS_OUTPUT.PUT_LINE (V_LASTNAME || ''s salary updated');
    ELSE
        DBMS_OUTPUT.PUT_LINE (V_LASTNAME || ' earns ' ||
        TO_CHAR(V_SAL));
    END IF;
END;
/
```

8. Erstellen Sie einen PL/SQL-Block, der das Gehalt eines Mitarbeiters in einer Austauschvariablen speichert. Gehen Sie im ausführbaren Teil des Programms wie folgt vor:
 - Berechnen Sie das Jahresgehalt als "Gehalt mal 12".
 - Berechnen Sie die Prämie gemäß den folgenden Angaben:

ahresgehalt	Prämie
>= 20,000	2,000
19,999–10,000	1,000
<= 9,999	500

- Zeigen Sie die Höhe der Prämie im folgenden Format im Fenster an:
"The bonus is \$....."
- Testen Sie den PL/SQL-Block mit folgenden Werten:

SALARY	BONUS
5000	2000
1000	1000
15000	2000

Zusätzliche Übungen 8 und 9 – Lösungen

```
DEFINE B_SALARY = 5000
DECLARE
    V_SAL  NUMBER(7,2) := &B_SALARY;
    V_BONUS  NUMBER(7,2);
    V_ANN_SALARY NUMBER(15,2);
BEGIN
    V_ANN_SALARY := V_SAL * 12;
    IF V_ANN_SALARY >= 20000 THEN
        V_BONUS := 2000;
    ELSIF V_ANN_SALARY <= 19999 AND V_ANN_SALARY >=10000 THEN
        V_BONUS := 1000;
    ELSE
        V_BONUS := 500;
    END IF;
    DBMS_OUTPUT.PUT_LINE ('The Bonus is $ ' ||
    TO_CHAR(V_BONUS));
END;
/
```

9. a. Führen Sie das Skript `lab_ap_09_a.sql` aus, um eine temporäre Tabelle namens `emp` zu erstellen. Erstellen Sie einen PL/SQL-Block, um eine Personalnummer, die neue Abteilungsnummer und die Erhöhung des Gehalts in Prozent in *i*SQL*Plus-Austauschvariablen zu speichern.

```
DEFINE B_EMPNO = 100
DEFINE B_NEW_DEPTNO = 10
DEFINE B_PER_INCREASE = 2
```

- b. Aktualisieren Sie die Abteilungsnummer des Mitarbeiters durch die neue Abteilungsnummer und das Gehalt durch das neue Gehalt. Speichern Sie die Aktualisierungen in der `emp`-Tabelle. Zeigen Sie nach Abschluss der Aktualisierung die Meldung "Update complete" im Fenster an. Wenn keine übereinstimmenden Records gefunden werden, zeigen Sie die Meldung "No Data Found" an. Testen Sie den PL/SQL-Block mit folgenden Werten:

EMPLOYEE_ID	NEW_DEPARTMENT_ID	% INCREASE	MESSAGE
100	20	2	Update Complete
10	30	5	No Data found
126	40	3	Update Complete

Zusätzliche Übungen 9 und 10 – Lösungen

```
DECLARE
    V_EMPNO emp.EMPLOYEE_ID%TYPE := &B_EMPNO;
    V_NEW_DEPTNO emp.DEPARTMENT_ID%TYPE := & B_NEW_DEPTNO;
    V_PER_INCREASE NUMBER(7,2) := & B_PER_INCREASE;
BEGIN
    UPDATE emp
        SET department_id = V_NEW_DEPTNO,
            salary = salary + (salary *
V_PER_INCREASE/100)
        WHERE employee_id = V_EMPNO;
    IF SQL%ROWCOUNT = 0 THEN
        DBMS_OUTPUT.PUT_LINE ('No Data Found');
    ELSE
        DBMS_OUTPUT.PUT_LINE ('Update Complete');
    END IF;
END;
/
```

10. Erstellen Sie einen PL/SQL-Block, um den `EMP_CUR`-Cursor zu deklarieren und den Mitarbeiternamen, das Gehalt und das Einstellungsdatum in der `employees`-Tabelle zu wählen. Verarbeiten Sie die einzelnen Zeilen im Cursor. Wenn das Gehalt höher als 15000 ist und das Einstellungsdatum nach dem 01-FEB-1988 liegt, zeigen Sie den Mitarbeiternamen, das Gehalt und das Einstellungsdatum im Fenster an.

```
DECLARE
    CURSOR C_EMP_CUR IS
        SELECT last_name,salary,hire_date FROM EMPLOYEES;
        V_ENAME VARCHAR2(25);
        V_SAL NUMBER(7,2);
        V_HIREDATE DATE;
BEGIN
    OPEN C_EMP_CUR;
    FETCH C_EMP_CUR INTO V_ENAME,V_SAL,V_HIREDATE;
    WHILE C_EMP_CUR%FOUND
    LOOP
        IF V_SAL > 15000 AND V_HIREDATE >=
TO_DATE('01-FEB-1988','DD-MON-YYYY') THEN
            DBMS_OUTPUT.PUT_LINE (V_ENAME || ' earns '
|| TO_CHAR(V_SAL) || ' and joined the organization on '
|| TO_DATE(V_HIREDATE,'DD-Mon-YYYY'));
        END IF;
    END LOOP;
END;
```

Zusätzliche Übungen 10 und 11 – Lösungen

```
    FETCH C_EMP_CUR INTO V_ENAME,V_SAL,V_HIREDATE;
  END LOOP;
  CLOSE C_EMP_CUR;
END;
/
```

11. Erstellen Sie einen PL/SQL-Block, um den Nachnamen und die Abteilungsnummer der Mitarbeiter, deren EMPLOYEE_ID unter 114 liegt, aus der employees-Tabelle abzurufen. Füllen Sie mit den abgerufenen Werten zwei PL/SQL-Tabellen. Speichern Sie in einer Tabelle die Records des Nachnamens und in der zweiten Tabelle die Records der Abteilungsnummern. Rufen Sie mit einer Schleife die Informationen zu Mitarbeiternamen und Gehalt aus den PL/SQL-Tabellen ab, und zeigen Sie sie im Fenster an. Verwenden Sie hierfür DBMS_OUTPUT.PUT_LINE. Zeigen Sie diese Details für die ersten 15 Mitarbeiter in den PL/SQL-Tabellen an.

```
DECLARE
  TYPE Table_Ename IS TABLE OF employees.last_name%TYPE
  INDEX BY BINARY_INTEGER;
  TYPE Table_dept IS TABLE OF employees.department_id%TYPE
  INDEX BY BINARY_INTEGER;
  Tename Table_Ename;
  Tdept Table_dept;
  i BINARY_INTEGER := 0;
  CURSOR Namedept IS SELECT last_name,department_id FROM
  employees WHERE employee_id < 115;
  TRACK NUMBER := 15;
BEGIN
  FOR emprec IN Namedept
  LOOP
    i := i +1;
    Tename(i) := emprec.last_name;
    Tdept(i) := emprec.department_id;
  END LOOP;
```

Zusätzliche Übungen 11 und 12 – Lösungen

```
FOR i IN 1..TRACK
LOOP
    DBMS_OUTPUT.PUT_LINE ('Employee Name: ' ||
Tename(i) || ' Department_id: ' || Tdept(i));
END LOOP;
END;
/
```

12. a. Erstellen Sie einen PL/SQL-Block, der einen Cursor namens C_DATE_CUR deklariert. Übergeben Sie an den Cursor einen Parameter vom Datentyp DATE, und geben Sie die Details aller Mitarbeiter aus, die nach diesem Datum eingestellt wurden.

```
DEFINE B_HIREDATE = 08-MAR-00
```

- b. Testen Sie den PL/SQL-Block mit folgenden Einstellungsdaten: 08-MAR-00, 25-JUN-97, 28-SEP-98, 07-FEB-99.

```
DECLARE
    CURSOR C_DATE_CURSOR(JOIN_DATE DATE) IS
        SELECT employee_id, last_name, hire_date FROM employees
        WHERE HIRE_DATE > JOIN_DATE ;
    V_EMPNO    employees.employee_id%TYPE;
    V_ENAME    employees.last_name%TYPE;
    V_HIREDATE employees.hire_date%TYPE;
    V_HDATE    employees.hire_date%TYPE := '&B_HIREDATE';
BEGIN
    OPEN C_DATE_CURSOR(V_HDATE);
    LOOP
        FETCH C_DATE_CURSOR INTO V_EMPNO,VENAME,V_HIREDATE;
        EXIT WHEN C_DATE_CURSOR%NOTFOUND;
        DBMS_OUTPUT.PUT_LINE (V_EMPNO || ' ' || VENAME || ' '
        || V_HIREDATE);
    END LOOP;
END;
/
```

Zusätzliche Übung 13 – Lösungen

13. Führen Sie das Skript lab_ap_09_a.sql aus, um die emp-Tabelle erneut zu erstellen. Erstellen Sie einen PL/SQL-Block, um Sachbearbeiter, die mehr als 3000 verdienen, der Tätigkeitskennung SR_CLERK zuzuweisen und ihr Gehalt um 10 % zu erhöhen. Verwenden Sie für diese Übung die emp-Tabelle. Prüfen Sie die Ergebnisse, indem Sie die emp-Tabelle abfragen.

Tipp: Verwenden Sie einen Cursor mit der Syntax FOR UPDATE und CURRENT OF.

```
DECLARE
    CURSOR C_Senior_Clerk IS
        SELECT employee_id, job_id FROM emp
        WHERE job_id = 'ST_CLERK' AND salary > 3000
        FOR UPDATE OF job_id;
BEGIN
    FOR Emrec IN C_Senior_Clerk
    LOOP
        UPDATE emp
        SET job_id = 'SR_CLERK',
            salary = 1.1 * salary
        WHERE CURRENT OF C_Senior_Clerk;
    END LOOP;
    COMMIT;
END;
/
SELECT * FROM emp;
```

Oracle Internal & Oracle Academy
Use Only

Zusätzliche Übung 14 – Lösungen

14. a. Erstellen Sie für die folgende Übung eine Tabelle, in der die Ergebnisse gespeichert werden. Sie haben folgende Möglichkeiten: Erstellen Sie die analysis-Tabelle manuell oder automatisch mit dem Skript lab_ap_14_a.sql. Erstellen Sie eine analysis-Tabelle mit den folgenden drei Spalten:

Spaltenname	E NAME	YEARS	SAL
Schlüsseltyp			
Null/Unique-Werte			
FK-Tabelle			
FK-Spalte			
Datentyp	VARCHAR2	Number	Number
Länge	20	2	8, 2

```
CREATE TABLE analysis
(ename Varchar2(20),
years Number(2),
sal Number(8,2));
```

- b. Erstellen Sie einen PL/SQL-Block, um die analysis-Tabelle mit den Informationen aus der employees-Tabelle zu füllen. Speichern Sie den Nachnamen eines Mitarbeiters in einer Austauschvariablen.

```
DEFINE B_ENAME = Austin
```

- c. Fragen Sie in der employees-Tabelle ab, ob der betreffende Mitarbeiter länger als fünf Jahre bei der Organisation tätig ist. Wenn das Gehalt gleichzeitig unter 3500 liegt, lösen Sie eine Exception aus. Behandeln Sie die Exception mit einem geeigneten Exception-Handler, der die folgenden Werte in die analysis-Tabelle einfügt: Nachname, Dienstjahre und aktuelles Gehalt des Mitarbeiters. Zeigen Sie andernfalls die Meldung Not due for a raise im Fenster an. Prüfen Sie die Ergebnisse, indem Sie die analysis-Tabelle abfragen. Testen Sie den PL/SQL-Block mit den folgenden Werten:

LAST_NAME	MESSAGE
Austin	Not due for a raise
Nayer	Due for a raise
Fripp	Not due for a raise
Khoo	Due for a raise

Zusätzliche Übung 14 – Lösungen (Fortsetzung)

```
DECLARE
    E_DUE_FOR_RAISE EXCEPTION;
    V_HIREDATE EMPLOYEES.HIRE_DATE%TYPE;
    V_ENAME EMPLOYEES.LAST_NAME%TYPE := INITCAP( '& B_ENAME' );
    V_SAL EMPLOYEES.SALARY%TYPE;
    V_YEARS NUMBER(2);
BEGIN
    SELECT LAST_NAME, SALARY, HIRE_DATE
    INTO V_ENAME, V_SAL, V_HIREDATE
    FROM employees WHERE last_name = V_ENAME;
    V_YEARS := MONTHS_BETWEEN(SYSDATE, V_HIREDATE)/12;
    IF V_SAL < 3500 AND V_YEARS > 5 THEN
        RAISE E_DUE_FOR_RAISE;
    ELSE
        DBMS_OUTPUT.PUT_LINE ('Not due for a raise');
    END IF;

EXCEPTION
    WHEN E_DUE_FOR_RAISE THEN
        INSERT INTO ANALYSIS(ENAME, YEARS, SAL)
        VALUES (V_ENAME, V_YEARS, V_SAL);
END;
/
```