

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

**Übungen**

D80170DE10  
Production 1.0  
Oktober 2013  
Bestellnummer: D83983

**ORACLE®**

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

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

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

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

**U.S. GOVERNMENT END USERS:**

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

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

## **Autoren**

Supriya Ananth

## **Technischer Inhalt und Überarbeitung**

Wayne Abbott, Anjulapponni Azhagulekshmi, Christopher Burandt, Yanti Chang, Diganta Choudhury, Salome Clement, Laszlo Czinkoczki, Steve Friedberg, Nancy Greenberg, KimSeong Loh, Miyuki Osato, Manish Pawar, Brian Pottle, Swarnapriya Shridhar

**Dieses Buch wurde erstellt mit: *oracle***tutor****

# Inhaltsverzeichnis

---

<b>Übungen zu Lektion 1 – Einführung.....</b>	<b>1-1</b>
Übungen zu Lektion 1 – Überblick .....	1-2
Übung 1 zu Lektion 1 – Verfügbare SQL Developer-Ressourcen bestimmen .....	1-3
Übung 1 zu Lektion 1 – Lösung: Verfügbare SQL Developer-Ressourcen bestimmen.....	1-4
Übung 2 zu Lektion 1 – Neue SQL Developer-Datenbankverbindung erstellen und verwenden .....	1-6
Übung 2 zu Lektion 1 – Lösung: Neue SQL Developer-Datenbankverbindung erstellen und verwenden.....	1-7
Übung 3 zu Lektion 1 – Schematabellen durchsuchen und einfachen anonymen Block erstellen und ausführen.....	1-9
Übung 3 zu Lektion 1 – Lösung: Schematabellen durchsuchen und einfachen anonymen Block erstellen und ausführen .....	1-10
Übung 4 zu Lektion 1 – Verschiedene SQL Developer-Voreinstellungen festlegen.....	1-15
Übung 4 zu Lektion 1 – Lösung: Verschiedene SQL Developer-Voreinstellungen festlegen .....	1-16
Übung 5 zu Lektion 1 – Auf die Onlinedokumentenbibliothek von Oracle Database zugreifen .....	1-20
Übung 5 zu Lektion 1 – Lösung: Auf die Onlinedokumentenbibliothek von Oracle Database zugreifen .....	1-21
<b>Übungen zu Lektion 2 – Prozeduren erstellen.....</b>	<b>2-1</b>
Übungen zu Lektion 2 – Überblick .....	2-2
Übung 1 zu Lektion 2 – Prozeduren erstellen, kompilieren und aufrufen .....	2-3
Übung 1 zu Lektion 2 – Lösung: Prozeduren erstellen, kompilieren und aufrufen .....	2-6
<b>Übungen zu Lektion 3 – Funktionen erstellen und Unterprogramme debuggen .....</b>	<b>3-1</b>
Übungen zu Lektion 3 – Überblick .....	3-2
Übung 1 zu Lektion 3 – Funktionen erstellen .....	3-3
Übung 1 zu Lektion 3 – Lösung: Funktionen erstellen .....	3-5
Übung 2 zu Lektion 3 – SQL Developer-Debugger – Einführung.....	3-11
Übung 2 zu Lektion 3 – Lösung: SQL Developer-Debugger – Einführung .....	3-12
<b>Übungen zu Lektion 4 – Packages erstellen.....</b>	<b>4-1</b>
Übungen zu Lektion 4 – Überblick .....	4-2
Übung 1 zu Lektion 4 – Packages erstellen und verwenden.....	4-3
Übung 1 zu Lektion 4 – Lösung: Packages erstellen und verwenden .....	4-5
<b>Übungen zu Lektion 5 – Mit Packages arbeiten .....</b>	<b>5-1</b>
Übungen zu Lektion 5 – Überblick .....	5-2
Übung 1 zu Lektion 5 – Mit Packages arbeiten .....	5-3
Übung 1 zu Lektion 5 – Lösung: Mit Packages arbeiten .....	5-7
<b>Übungen zu Lektion 6 – Von Oracle bereitgestellte Packages zur Anwendungsentwicklung.....</b>	<b>6-1</b>
Übungen zu Lektion 6 – Überblick .....	6-2
Übung 1 zu Lektion 6 – Package UTL_FILE .....	6-3
Übung 1 zu Lektion 6 – Lösung: Package UTL_FILE .....	6-4
<b>Übungen zu Lektion 7 – Dynamisches SQL.....</b>	<b>7-1</b>
Übungen zu Lektion 7 – Überblick .....	7-2
Übung 1 zu Lektion 7 – Natives dynamisches SQL .....	7-3
Übung 1 zu Lektion 7 – Lösung: Natives dynamisches SQL.....	7-5
<b>Übungen zu Lektion 8 – Überlegungen zum Design von PL/SQL-Code.....</b>	<b>8-1</b>
Übungen zu Lektion 8 – Überblick .....	8-2
Übung 1 zu Lektion 8 – Bulk Binding und autonome Transaktionen .....	8-3
Übung 1 zu Lektion 8 – Lösung: Bulk Binding und autonome Transaktionen .....	8-5

<b>Übungen zu Lektion 9 – Trigger erstellen .....</b>	<b>9-1</b>
Übungen zu Lektion 9 – Überblick .....	9-2
Übung 1 zu Lektion 9 – Statement Trigger und Row Trigger erstellen.....	9-3
Übung 1 zu Lektion 9 – Lösung: Statement Trigger und Row Trigger erstellen .....	9-5
<b>Übungen zu Lektion 10 – Komplexe, DDL- und Datenbankereignis-Trigger erstellen.....</b>	<b>10-1</b>
Übungen zu Lektion 10 – Überblick .....	10-2
Übung 1 zu Lektion 10 – Datenintegritätsregeln und Exceptions für sich verändernde Tabellen verwalten .....	10-3
Übung 1 zu Lektion 10 – Lösung: Datenintegritätsregeln und Exceptions für sich verändernde Tabellen verwalten .....	10-6
<b>Übungen zu Lektion 11 – PL/SQL-Compiler .....</b>	<b>11-1</b>
Übungen zu Lektion 11 – Überblick .....	11-2
Übung 1 zu Lektion 11 – PL/SQL-Compilerparameter und -warnungen verwenden.....	11-3
Übung 1 zu Lektion 11 – Lösung: PL/SQL-Compilerparameter und -warnungen verwenden .....	11-5
<b>Übungen zu Lektion 12 – Abhängigkeiten verwalten.....</b>	<b>12-1</b>
Übungen zu Lektion 12 – Überblick .....	12-2
Übung 1 zu Lektion 12 – Abhängigkeiten im Schema verwalten .....	12-3
Übung 1 zu Lektion 12 – Lösung: Abhängigkeiten im Schema verwalten .....	12-4
<b>Zusätzliche Übungen 1 .....</b>	<b>13-1</b>
Zusätzliche Übungen 1 – Überblick.....	13-2
Übung 1 zu Lektion 1 – Neue Datenbankverbindung für SQL Developer erstellen.....	13-3
Übung 1 zu Lektion 1 – Lösung: Neue Datenbankverbindung für SQL Developer erstellen .....	13-4
Übung 2 zu Lektion 1 – Neue Tätigkeit zur Tabelle JOBS hinzufügen.....	13-6
Übung 2 zu Lektion 1 – Lösung: Neue Tätigkeit zur Tabelle JOBS hinzufügen .....	13-7
Übung 3 zu Lektion 1 – Neue Rolle zur Tabelle JOB_HISTORY hinzufügen.....	13-9
Übung 3 zu Lektion 1 – Lösung: Neue Rolle zur Tabelle JOB_HISTORY hinzufügen .....	13-10
Übung 4 zu Lektion 1 – Mindest- und Höchstgehalt für eine Tätigkeit aktualisieren .....	13-13
Übung 4 zu Lektion 1 – Lösung: Mindest- und Höchstgehalt für eine Tätigkeit aktualisieren.....	13-14
Übung 5 zu Lektion 1 – Mitarbeitergehälter überwachen .....	13-17
Übung 5 zu Lektion 1 – Lösung: Mitarbeitergehälter überwachen .....	13-18
Übung 6 zu Lektion 1 – Gesamtanzahl der Dienstjahre eines Mitarbeiters abrufen.....	13-22
Übung 6 zu Lektion 1 – Lösung: Gesamtanzahl der Dienstjahre eines Mitarbeiters abrufen .....	13-23
Übung 7 zu Lektion 1 – Gesamtanzahl verschiedener Tätigkeiten eines Mitarbeiters abrufen .....	13-26
Übung 7 zu Lektion 1 – Lösung: Gesamtanzahl verschiedener Tätigkeiten eines Mitarbeiters abrufen .....	13-27
Übung 8 zu Lektion 1 – Neues Package erstellen, das die neu erstellten Prozeduren und Funktionen enthält .....	13-29
Übung 8 zu Lektion 1 – Lösung: Neues Package erstellen, das die neu erstellten Prozeduren und Funktionen enthält .....	13-30
Übung 9 zu Lektion 1 – Trigger erstellen, um sicherzustellen, dass die Mitarbeitergehälter innerhalb des akzeptablen Bereichs liegen.....	13-36
Übung 9 zu Lektion 1 – Lösung: Trigger erstellen, um sicherzustellen, dass die Mitarbeitergehälter innerhalb des akzeptablen Bereichs liegen .....	13-37
<b>Zusätzliche Übungen 2 .....</b>	<b>14-1</b>
Zusätzliche Übungen 2 – Überblick.....	14-2
Übung 1 zu Lektion 2 – Package VIDEO_PKG erstellen .....	14-4
Übung 1 zu Lektion 2 – Lösung: Package VIDEO_PKG erstellen .....	14-6

# **Übungen zu Lektion 1 – Einführung**

## **Kapitel 1**

# Übungen zu Lektion 1: Überblick

---

## Lektionsüberblick

Dies ist die erste von mehreren Übungen in diesem Kurs. Die Übungen decken die meisten Themen ab, die in den entsprechenden Lektionen behandelt wurden.

**Hinweis:** Wenn Sie einen Schritt in einer Übung ausgelassen haben, führen Sie das entsprechende Lösungsskript zu diesem Übungsschritt aus, bevor Sie mit dem nächsten Schritt oder der nächsten Übung fortfahren.

In dieser Übung prüfen Sie die verfügbaren SQL Developer-Ressourcen. Sie erhalten Informationen zu Ihrem Benutzeraccount für diesen Kurs. Anschließend starten Sie SQL Developer, erstellen eine neue Datenbankverbindung, durchsuchen die Schematabellen, erstellen einen einfachen anonymen Block und führen ihn aus. Außerdem legen Sie verschiedene SQL Developer-Voreinstellungen fest und führen SQL-Anweisungen sowie einen anonymen PL/SQL-Block mit dem SQL Worksheet aus. Zuletzt greifen Sie auf die Oracle Database-Dokumentation und andere nützliche Websites zu, die Sie in diesem Kurs verwenden können, und setzen Lesezeichen.

# Übung 1 zu Lektion 1 – Verfügbare SQL Developer-Ressourcen bestimmen

---

## Überblick

In dieser Übung prüfen Sie die verfügbaren SQL Developer-Ressourcen.

## Aufgabe

1. Machen Sie sich, soweit erforderlich, mit Oracle SQL Developer vertraut. Verwenden Sie dazu Anhang B, "SQL Developer".
2. Rufen Sie die SQL Developer-Homepage online unter der folgenden Adresse auf:  
[http://www.oracle.com/technology/products/database/sql\\_developer/index.html](http://www.oracle.com/technology/products/database/sql_developer/index.html)
3. Um künftig schneller auf diese Seite zugreifen zu können, setzen Sie ein Lesezeichen.
4. Rufen Sie das SQL Developer-Tutorial online unter der folgenden Adresse auf:  
<http://st-curriculum.oracle.com/tutorial/SQLDeveloper/index.htm>
5. Sehen Sie sich die verfügbaren Links und Demos im Tutorial an, und testen Sie sie gegebenenfalls aus, insbesondere die Links **Creating a Database Connection** und **Accessing Data**.

# Übung 1 zu Lektion 1 – Lösung: Verfügbare SQL Developer-Ressourcen bestimmen

## Lösung

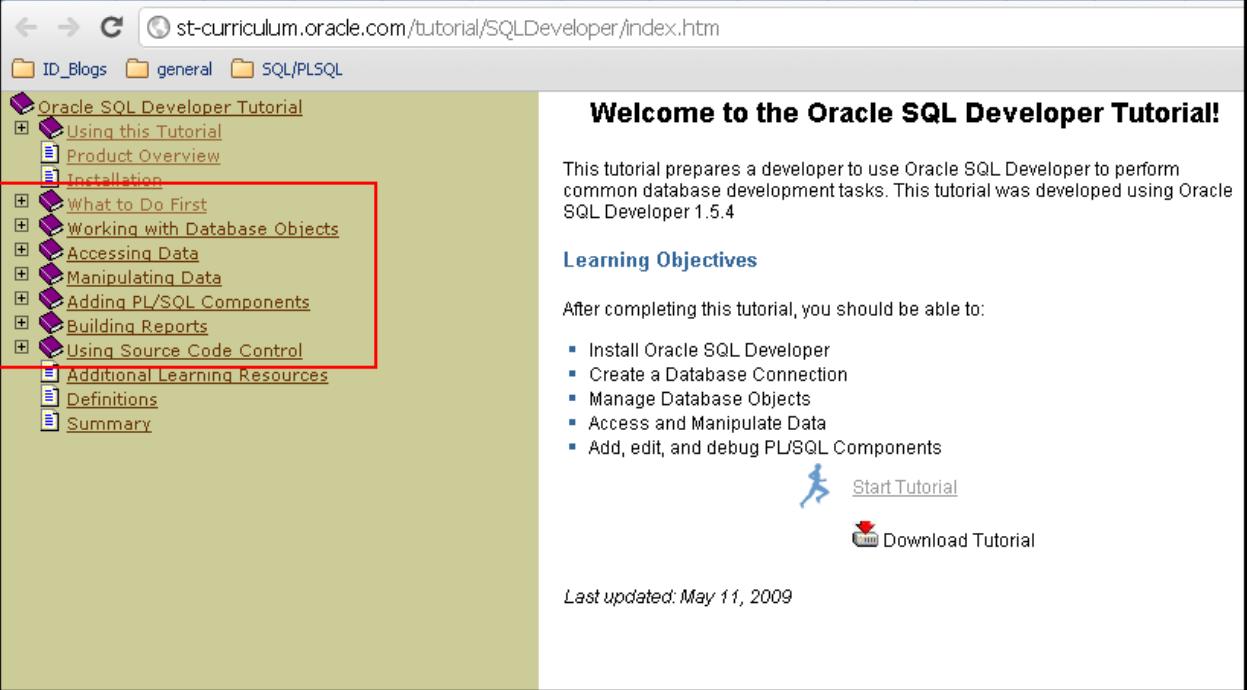
In dieser Übung prüfen Sie die verfügbaren SQL Developer-Ressourcen.

1. Machen Sie sich, soweit erforderlich, mit Oracle SQL Developer vertraut. Verwenden Sie dazu Anhang B, "SQL Developer".
2. Rufen Sie die SQL Developer-Homepage online unter der folgenden Adresse auf:  
[http://www.oracle.com/technology/products/database/sql\\_developer/index.html](http://www.oracle.com/technology/products/database/sql_developer/index.html)

**Hinweis:** Die Screenshots in diesem Kurs stammen aus Version 3.2 von SQL Developer. Die Onlinehomepage von SQL Developer bezieht sich dagegen auf die neueste für den Download verfügbare SQL Developer-Version.

3. Um künftig schneller auf diese Seite zugreifen zu können, setzen Sie ein Lesezeichen.  
**Keine formale Lösung. Der Link wird der Symbolleiste Links wie folgt hinzugefügt:**
4. Rufen Sie das SQL Developer-Tutorial online unter der folgenden Adresse auf:  
<http://st-curriculum.oracle.com/tutorial/SQLDeveloper/index.htm>

Rufen Sie das SQL Developer-Tutorial mit der oben angegebenen URL auf. Die folgende Seite wird angezeigt:



The screenshot shows a web browser displaying the Oracle SQL Developer Tutorial at <http://st-curriculum.oracle.com/tutorial/SQLDeveloper/index.htm>. The page has a sidebar on the left containing a navigation tree. A red box highlights the 'Using Source Code Control' section under the 'Additional Learning Resources' category. The main content area features a title 'Welcome to the Oracle SQL Developer Tutorial!', a brief description of the tutorial's purpose, a 'Learning Objectives' section with a bulleted list of goals, and two buttons at the bottom: 'Start Tutorial' and 'Download Tutorial'. The footer indicates the page was last updated on May 11, 2009.

5. Sehen Sie sich die verfügbaren Links und Demos im Tutorial an, und testen Sie sie gegebenenfalls aus, insbesondere die Links **Creating a Database Connection** und **Accessing Data**.

Um sich den Bereich zum Erstellen einer Datenbankverbindung anzusehen, klicken Sie neben dem Link **What to Do First** auf das Pluszeichen (+). Daraufhin wird der Link **Creating a Database Connection** eingeblendet. Um das Thema "Creating a Database Connection" zu lesen, klicken Sie auf den Link zum Thema. Um den Bereich zum Zugreifen auf Daten zu lesen, klicken Sie neben dem Link **Accessing Data** auf das Pluszeichen (+). Daraufhin wird die Liste der verfügbaren Themen angezeigt. Sie können ein beliebiges Thema aufrufen, indem Sie auf den entsprechenden Link klicken.

# Übung 2 zu Lektion 1 – Neue SQL Developer-Datenbankverbindung erstellen und verwenden

---

## Überblick

In dieser Übung starten Sie SQL Developer mit Ihren Verbindungsinformationen und erstellen eine neue Datenbankverbindung.

## Aufgaben

1. Starten Sie SQL Developer mit der Benutzer-ID und dem Kennwort, die Sie vom Dozenten erhalten haben, beispielsweise `ora61`.
2. Melden Sie sich mit folgenden Daten bei der Datenbank an:
  - a. **Connection Name:** MyDBConnection
  - b. **Username:** `ora61`
  - c. **Password:** `ora61`
  - d. **Hostname:** Geben Sie den Hostnamen für Ihren PC oder alternativ **localhost** ein.
  - e. **Port:** 1521
  - f. **SID:** ORCL
3. Testen Sie die neue Verbindung. Wenn der Status **Success** angezeigt wird, melden Sie sich über diese neue Verbindung bei der Datenbank an:
  - a. Klicken Sie im Fenster **New>Select Database Connection** auf die Schaltfläche **Test**. Wenn der Status **Success** angezeigt wird, klicken Sie auf die Schaltfläche **Connect**.

## Übung 2 zu Lektion 1 – Lösung: Neue SQL Developer-Datenbankverbindung erstellen und verwenden

In dieser Übung starten Sie SQL Developer mit Ihren Verbindungsinformationen und erstellen eine neue Datenbankverbindung.

1. Starten Sie SQL Developer mit der Benutzer-ID und dem Kennwort, die Sie vom Dozenten erhalten haben, beispielsweise ora61.

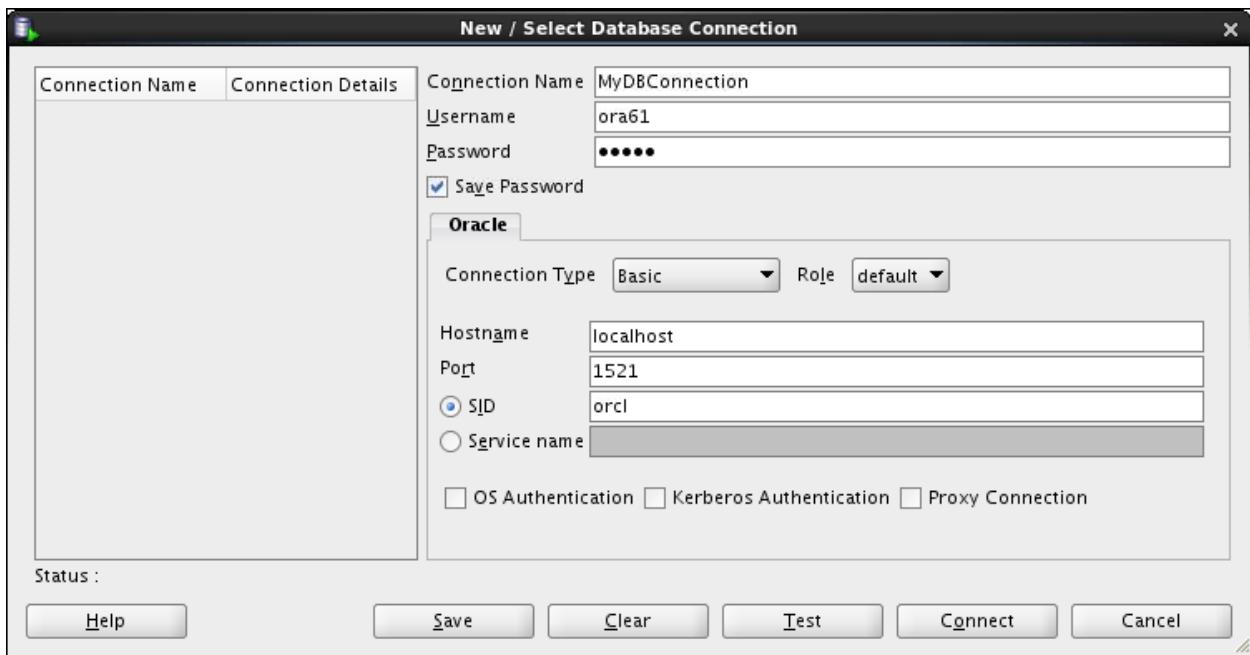
**Klicken Sie auf dem Desktop auf das Symbol SQL Developer.**



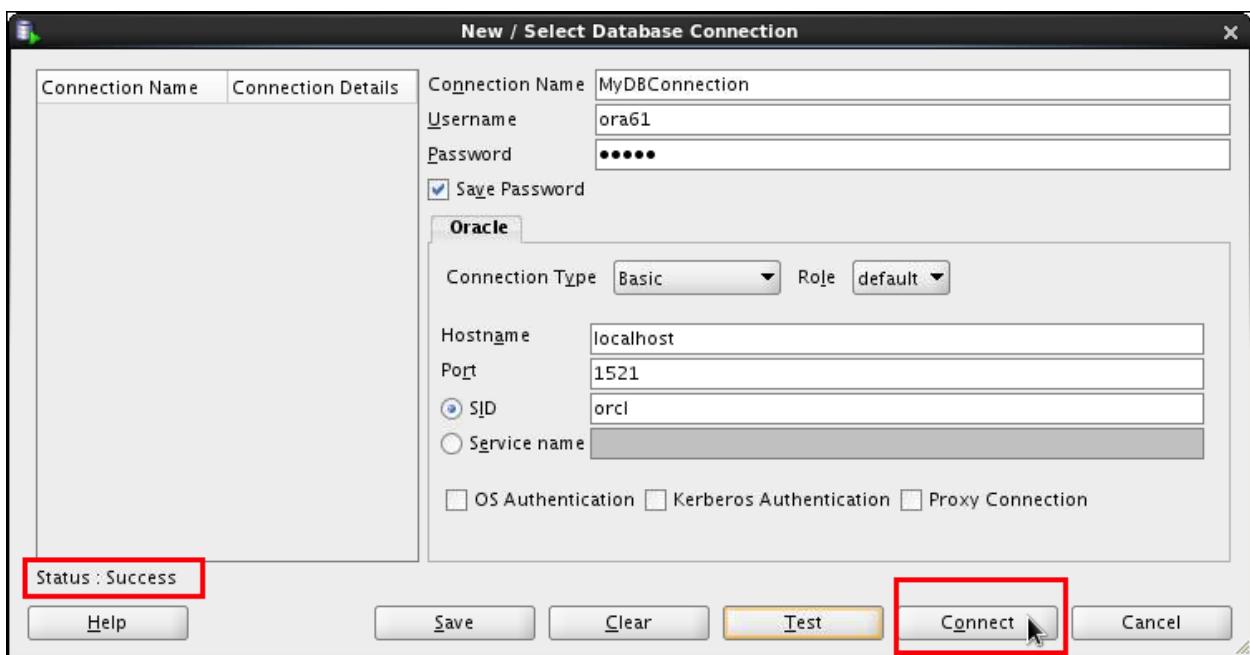
2. Melden Sie sich mit folgenden Daten bei der Datenbank an:
  - a. **Connection Name:** MyDBConnection
  - b. **Username:** ora61
  - c. **Password:** ora61
  - d. Um die Verbindung zu speichern, aktivieren Sie das Kontrollkästchen **Save Password**.
  - e. **Hostname:** Geben Sie den Hostnamen für Ihren PC oder alternativ **localhost** ein.
  - f. **Port:** 1521
  - g. **SID:** ORCL

Klicken Sie in der Registerkarte **Connections** mit der rechten Maustaste auf das Symbol **Connections**, und wählen Sie im Kontextmenü die Option **New Connection**. Das Fenster **New>Select Database Connection** wird angezeigt. Erstellen Sie die neue Datenbankverbindung mithilfe der oben aufgeführten Informationen.

**Hinweis:** Um die Eigenschaften der neu erstellten Verbindung anzuzeigen, klicken Sie mit der rechten Maustaste auf den Namen der Verbindung und wählen im Kontextmenü die Option **Properties**. Ersetzen Sie die Werte für **Username**, **Password**, **Hostname** und **Service name** durch die Werte, die Sie von Ihrem Dozenten erhalten haben. Nachstehend finden Sie ein Beispiel für eine neu erstellte Datenbankverbindung für den Teilnehmer **ora61**:



3. Testen Sie die neue Verbindung. Wenn der Status **Success** angezeigt wird, melden Sie sich über diese neue Verbindung bei der Datenbank an:
- Klicken Sie im Fenster **New>Select Database Connection** auf die Schaltfläche **Test**. Wenn der Status **Success** angezeigt wird, klicken Sie auf die Schaltfläche **Connect**.



# Übung 3 zu Lektion 1 – Schematabellen durchsuchen und einfachen anonymen Block erstellen und ausführen

## Überblick

In dieser Übung durchsuchen Sie die Schematabellen, erstellen einen einfachen anonymen Block und führen ihn aus.

## Aufgaben

1. Durchsuchen Sie die Struktur der Tabelle EMPLOYEES, und zeigen Sie ihre Daten an.
  - a. Blenden Sie die Verbindung **MyDBConnection** ein, indem Sie daneben auf das Pluszeichen klicken.
  - b. Blenden Sie das Symbol **Tables** ein, indem Sie daneben auf das Pluszeichen klicken.
  - c. Zeigen Sie die Struktur der Tabelle EMPLOYEES an.
2. Durchsuchen Sie die Tabelle EMPLOYEES, und zeigen Sie ihre Daten an.
3. Wählen Sie mit dem SQL Worksheet die Nachnamen und Gehälter aller Mitarbeiter, deren Jahresgehalt über \$ 10.000 liegt. Führen Sie die Anweisung `SELECT` sowohl über das Symbol **Execute Statement** (F9) als auch über das Symbol **Run Script** (F5) aus. Prüfen Sie die Ergebnisse beider Ausführungsmethoden für die `SELECT`-Anweisungen in den entsprechenden Registerkarten.
- Hinweis:** Machen Sie sich einige Minuten mit den Daten vertraut, oder konsultieren Sie Anhang A mit den Beschreibungen und Daten aller Tabellen im Schema `HR`, die in diesem Kurs verwendet werden.
4. Erstellen Sie einen einfachen anonymen Block, der "Hello World" ausgibt, und führen Sie ihn aus.
  - a. Um die Ausgabe der Anweisungen aus dem Package `DBMS_OUTPUT` anzuzeigen, aktivieren Sie `SET SERVEROUTPUT ON`.
  - b. Geben Sie den Code für den anonymen Block im SQL Worksheet-Bereich ein.
  - c. Um den anonymen Block auszuführen, klicken Sie auf das Symbol **Run Script** (F5).

## Übung 3 zu Lektion 1 – Lösung: Schematabellen durchsuchen und einfachen anonymen Block erstellen und ausführen

In dieser Übung durchsuchen Sie die Schematabellen, erstellen einen einfachen anonymen Block und führen ihn aus.

1. Durchsuchen Sie die Struktur der Tabelle EMPLOYEES, und zeigen Sie ihre Daten an.
  - a. Blenden Sie die Verbindung MyDBConnection ein, indem Sie daneben auf das Pluszeichen klicken.
  - b. Blenden Sie das Symbol **Tables** ein, indem Sie daneben auf das Pluszeichen klicken.
  - c. Zeigen Sie die Struktur der Tabelle EMPLOYEES an.

Doppelklicken Sie auf die Tabelle EMPLOYEES. In der Registerkarte Columns werden die Spalten der Tabelle EMPLOYEES wie folgt angezeigt:

The screenshot shows the Oracle SQL Developer interface. On the left, the Connections pane shows a connection named 'MyDBConnection' with several tables listed under 'Tables (Filtered)'. The 'EMPLOYEES' table is selected and highlighted in blue. On the right, the main workspace displays the 'EMPLOYEES' table structure. The 'Columns' tab is selected, showing the following columns:

COLUMN_NAME	DATA_TYPE	NULLABLE	DATA_DEFAULT	COLUMN_ID	COMMENTS
1 EMPLOYEE_ID	NUMBER(6,0)	No	(null)	1	Primary key of employees table.
2 FIRST_NAME	VARCHAR2(20 BYTE)	Yes	(null)	2	First name of the employee. A not null column.
3 LAST_NAME	VARCHAR2(25 BYTE)	No	(null)	3	Last name of the employee. A not null column.
4 EMAIL	VARCHAR2(25 BYTE)	No	(null)	4	Email id of the employee.
5 PHONE_NUMBER	VARCHAR2(20 BYTE)	Yes	(null)	5	Phone number of the employee; includes area code.
6 HIRE_DATE	DATE	No	(null)	6	Date when the employee started on the job.
7 JOB_ID	VARCHAR2(10 BYTE)	No	(null)	7	Current job of the employee; foreign key to JOBS table.
8 SALARY	NUMBER(8,2)	Yes	(null)	8	Monthly salary of the employee. Null if not applicable.
9 COMMISSION_PCT	NUMBER(2,2)	Yes	(null)	9	Commission percentage of the employee.
10 MANAGER_ID	NUMBER(6,0)	Yes	(null)	10	Manager id of the employee; has salary.
11 DEPARTMENT_ID	NUMBER(4,0)	Yes	(null)	11	Department id where employee works.

2. Durchsuchen Sie die Tabelle EMPLOYEES, und zeigen Sie ihre Daten an.

Um die Daten der Mitarbeiter anzuzeigen, klicken Sie auf die Registerkarte **Data**. Die Daten der Tabelle EMPLOYEES werden wie folgt angezeigt:

The screenshot shows the Oracle SQL Developer interface with the title bar "Oracle SQL Developer : Table ORA61.EMPLOYEES@MyDBConnection". The menu bar includes File, Edit, View, Navigate, Run, Versioning, Tools, and Help. The toolbar has icons for New Connection, Reports, and various database operations. The Connections panel on the left shows a connection named "MyDBConnection" with tables like COUNTRIES, DEPARTMENTS, EMPLOYEES, JOBS, LOCATIONS, and REGIONS. The main area displays the "EMPLOYEES" table with 19 rows of data. The columns are: EMPLOYEE\_ID, FIRST\_NAME, LAST\_NAME, EMAIL, PHONE\_NUMBER, HIRE\_DATE, JOB\_ID, and SALARY. The data includes entries such as Steven King, Neena Kochhar, Lex De Haan, Alexander Hunold, Bruce Ernst, David Austin, Valili Pataballa, Diana Lorentz, Nancy Greenberg, Daniel Faviet, John Chen, Ismael Sciarra, Jose Manuel Urman, Luis Popp, Den Raphaely, Alexander Khoo, Shellie Baida, Sigal Tobias, and Guy Himuro.

	EMPLOYEE_ID	FIRST_NAME	LAST_NAME	EMAIL	PHONE_NUMBER	HIRE_DATE	JOB_ID	SALARY
1	100	Steven	King	SKING	515.123.4567	17-JUN-03	AD_PRES	2400
2	101	Neena	Kochhar	NKOCHHAR	515.123.4568	21-SEP-05	AD_VP	1700
3	102	Lex	De Haan	LDEHAAN	515.123.4569	13-JAN-01	AD_VP	1700
4	103	Alexander	Hunold	AHUNOLD	590.423.4567	03-JAN-06	IT_PROG	900
5	104	Bruce	Ernst	BERNST	590.423.4568	21-MAY-07	IT_PROG	600
6	105	David	Austin	DAUSTIN	590.423.4569	25-JUN-05	IT_PROG	480
7	106	Valli	Pataballa	VPATABAL	590.423.4560	05-FEB-06	IT_PROG	480
8	107	Diana	Lorentz	DLORENTZ	590.423.5567	07-FEB-07	IT_PROG	420
9	108	Nancy	Greenberg	NGREENBE	515.124.4569	17-AUG-02	FI_MGR	1200
10	109	Daniel	Faviet	DFAVIET	515.124.4169	16-AUG-02	FI_ACCOUNT	900
11	110	John	Chen	JCHEN	515.124.4269	28-SEP-05	FI_ACCOUNT	820
12	111	Ismael	Sciarra	ISCIARRA	515.124.4369	30-SEP-05	FI_ACCOUNT	770
13	112	Jose Manuel	Urman	JMURMAN	515.124.4469	07-MAR-06	FI_ACCOUNT	780
14	113	Luis	Popp	LPOPP	515.124.4567	07-DEC-07	FI_ACCOUNT	690
15	114	Den	Raphaely	DRAPHEAL	515.127.4561	07-DEC-02	PU_MAN	1100
16	115	Alexander	Khoo	AKHOO	515.127.4562	18-MAY-03	PU_CLERK	310
17	116	Shellie	Baida	SBAIDA	515.127.4563	24-DEC-05	PU_CLERK	290
18	117	Sigal	Tobias	STOBIAS	515.127.4564	24-JUL-05	PU_CLERK	280
19	118	Guy	Himuro	GHIMURO	515.127.4565	15-NOV-06	PU_CLERK	260

3. Wählen Sie mit dem SQL Worksheet die Nachnamen und Gehälter aller Mitarbeiter, deren Jahresgehalt über \$ 10.000 liegt. Führen Sie die **SELECT**-Anweisung sowohl über das Symbol **Execute Statement** (F9) als auch über das Symbol **Run Script** (F5) aus. Prüfen Sie die Ergebnisse beider Ausführungsmethoden für die **SELECT**-Anweisungen in den entsprechenden Registerkarten.

**Hinweis:** Machen Sie sich einige Minuten mit den Daten vertraut, oder konsultieren Sie Anhang A mit den Beschreibungen und Daten aller Tabellen im Schema HR, die in diesem Kurs verwendet werden.

Zeigen Sie das SQL Worksheet mithilfe einer der folgenden beiden Methoden an:

- Wählen Sie **Tools > SQL Worksheet**, oder klicken Sie auf das Symbol **Open SQL Worksheet**. Das Fenster **Select Connection** wird angezeigt.
- Wählen Sie in der Dropdown-Liste **Connection** die neue Verbindung **MyDBConnection** (falls noch nicht geschehen), und klicken Sie dann auf **OK**.

Öffnen Sie im Verzeichnis /home/oracle/labs/plpu/solns die Datei sol\_01.sql. Wählen Sie dazu eine der beiden folgenden Methoden:

- Wählen Sie in der Registerkarte **Files** die zu öffnende Skriptdatei (oder navigieren Sie zu dieser Datei).
- Doppelklicken Sie auf den Dateinamen. Der Code der Skriptdatei wird im SQL Worksheet-Bereich angezeigt. Entfernen Sie die Kommentarzeichen der Lösung für die 3. Aufgabe, und wählen Sie die Lösung.

- c. Um den Code auszuführen, klicken Sie in der SQL Worksheet-Symbolleiste auf das Symbol **Run Script** (F5).

Alternative:

- Wählen Sie im Menü **File** die Option **Open**. Das Dialogfeld **Open** wird angezeigt.
- Wählen Sie im Dialogfeld **Open** die zu öffnende Skriptdatei (oder navigieren Sie zu dieser Datei).
- Klicken Sie auf **Open**. Der Code der Skriptdatei wird im SQL Worksheet-Bereich angezeigt. Entfernen Sie die Kommentarzeichen der Lösung für die 3. Aufgabe, und wählen Sie die Lösung.
- Um den Code auszuführen, klicken Sie in der SQL Worksheet-Symbolleiste auf das Symbol "**Run Script**" (F5).

Um eine einzige SELECT-Anweisung auszuführen, klicken Sie in der SQL Worksheet-Symbolleiste auf das Symbol **Execute Statement** (F9). (Achten Sie darauf, dass der Cursor in einer Zeile der SELECT-Anweisung steht.) Der Code und das Ergebnis werden wie folgt angezeigt:

```
SELECT LAST_NAME, SALARY
FROM EMPLOYEES
WHERE SALARY > 10000;
```

The screenshot shows the 'Query Result' window from the Oracle SQL Worksheet. The window title is 'Query Result'. At the top, there are icons for Run, Save, Print, and Close, followed by the text 'SQL | All Rows Fetched: 15 in 0.004 seconds'. Below this is a table with two columns: 'LAST\_NAME' and 'SALARY'. The data consists of 15 rows, each containing a number from 1 to 15 followed by a name and its salary. The data is as follows:

	LAST_NAME	SALARY
1	King	24000
2	Kochhar	17000
3	De Haan	17000
4	Greenberg	12008
5	Raphaely	11000
6	Russell	14000
7	Partners	13500
8	Errazuriz	12000
9	Cambrault	11000
10	Zlotkey	10500
11	Vishney	10500
12	Ozer	11500
13	Abel	11000
14	Hartstein	13000
15	Higgins	12008

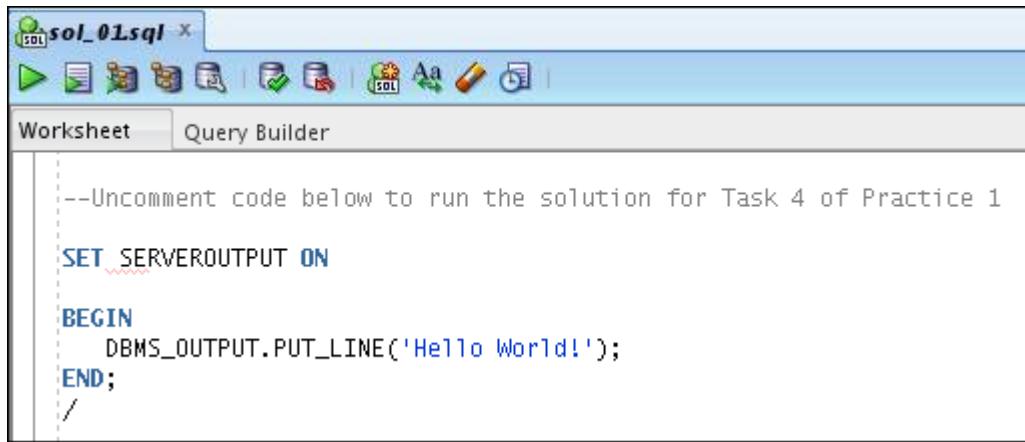
4. Erstellen Sie einen einfachen anonymen Block, der "Hello World" ausgibt, und führen Sie ihn aus.

- a. Um die Ausgabe der Anweisungen aus dem Package DBMS\_OUTPUT anzuzeigen, aktivieren Sie SET SERVEROUTPUT ON.

Geben Sie folgenden Befehl im SQL Worksheet-Bereich ein, und klicken Sie auf das Symbol **Run Script** (F5).

```
SET SERVEROUTPUT ON
```

- b. Geben Sie im SQL Worksheet-Bereich den Code für den anonymen Block ein. Öffnen Sie im Verzeichnis /home/oracle/labs/plpu/solns die Datei sol\_01.sql. Entfernen Sie die Kommentarzeichen des Codes in der 4. Aufgabe, und wählen Sie den Code. Der Code wird wie folgt angezeigt:



The screenshot shows the Oracle SQL Worksheet interface. The title bar says 'sol\_01sql'. The main area has tabs for 'Worksheet' and 'Query Builder', with 'Worksheet' selected. The code in the worksheet pane is:

```
--Uncomment code below to run the solution for Task 4 of Practice 1
SET SERVEROUTPUT ON
BEGIN
  DBMS_OUTPUT.PUT_LINE('Hello World!');
END;
/
```

- c. Um den anonymen Block auszuführen, klicken Sie auf das Symbol **Run Script** (F5). Die Registerkarte **Script Output** zeigt die Ausgabe des anonymen Blockes wie folgt an:

The screenshot shows the Oracle SQL Developer interface. At the top, there is a toolbar with various icons and a connection dropdown set to "MyDBConnection". Below the toolbar are two tabs: "Worksheet" and "Query Builder", with "Worksheet" selected. The main workspace contains the following PL/SQL code:

```
--Uncomment code below to run the solution for Task 4 of Practice 1
SET SERVEROUTPUT ON
BEGIN
    DBMS_OUTPUT.PUT_LINE('Hello World!');
END;
```

Below the workspace is a "Script Output" window. It has a toolbar with icons for copy, paste, and refresh, followed by the message "Task completed in 0.003 seconds". The output area displays the results of the executed anonymous block:

```
anonymous block completed
Hello World!
```

At the bottom of the window, there are status indicators: "Line 12 Column 3", "Insert", "Modified", "Unix/Mac: LF", and "Editing".

# Übung 4 zu Lektion 1 – Verschiedene SQL Developer-Voreinstellungen festlegen

## Überblick

In dieser Übung legen Sie einige SQL Developer-Voreinstellungen fest.

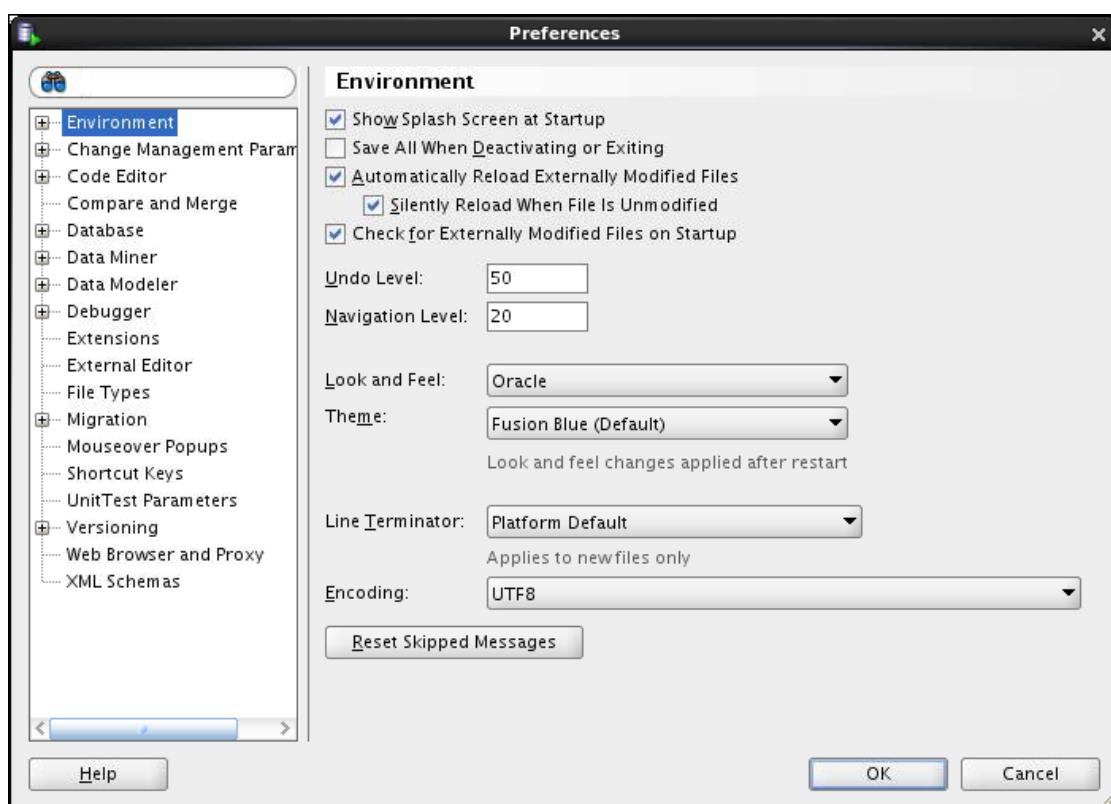
## Aufgaben

1. Navigieren Sie im SQL Developer-Menü zu **Tools > Preferences**. Das Fenster **Preferences** wird angezeigt.
2. Blenden Sie die Option **Code Editor** ein, und klicken Sie auf die Option **Display**, um den Bereich **Code Editor: Display** anzuzeigen. Der Bereich **Code Editor: Display** enthält allgemeine Optionen für die Darstellung und das Verhalten des Code Editors.
  - a. Geben Sie im Bereich **Show Visible Right Margin** im Textfeld **Right Margin Column** den Wert 100 ein. Damit wird ein rechter Rand angezeigt, den Sie zur Steuerung der Länge von Codezeilen einstellen können.
  - b. Klicken Sie auf die Option **Line Gutter**. Die Option **Line Gutter** legt die Optionen für den linken Rand des Code Editors fest. Um die Codezeilenummern anzuzeigen, aktivieren Sie das Kontrollkästchen **Show Line Numbers**.
3. Klicken Sie unter der Option **Database** auf die Option **Worksheet Parameters**. Geben Sie im Textfeld **Select default path to look for scripts** das Verzeichnis /home/oracle/labs/plpu an. Dieses Verzeichnis enthält die Lösungsskripte, Codebeispielskripte und alle in diesem Kurs verwendeten Übungen oder Demos.
4. Um die Änderungen zu übernehmen und das Fenster **Preferences** zu schließen, klicken Sie auf **OK**.
5. Machen Sie sich mit dem Verzeichnis /home/oracle/labs/plpu vertraut.
  - a. Klicken Sie (neben der Registerkarte **Connections**) auf die Registerkarte **Files**.
  - b. Navigieren Sie zum Verzeichnis /home/oracle/labs/plpu.
  - c. Wie viele Unterverzeichnisse enthält das Verzeichnis labs?
  - d. Navigieren Sie durch die Verzeichnisse, und öffnen Sie eine Skriptdatei, ohne den Code auszuführen.

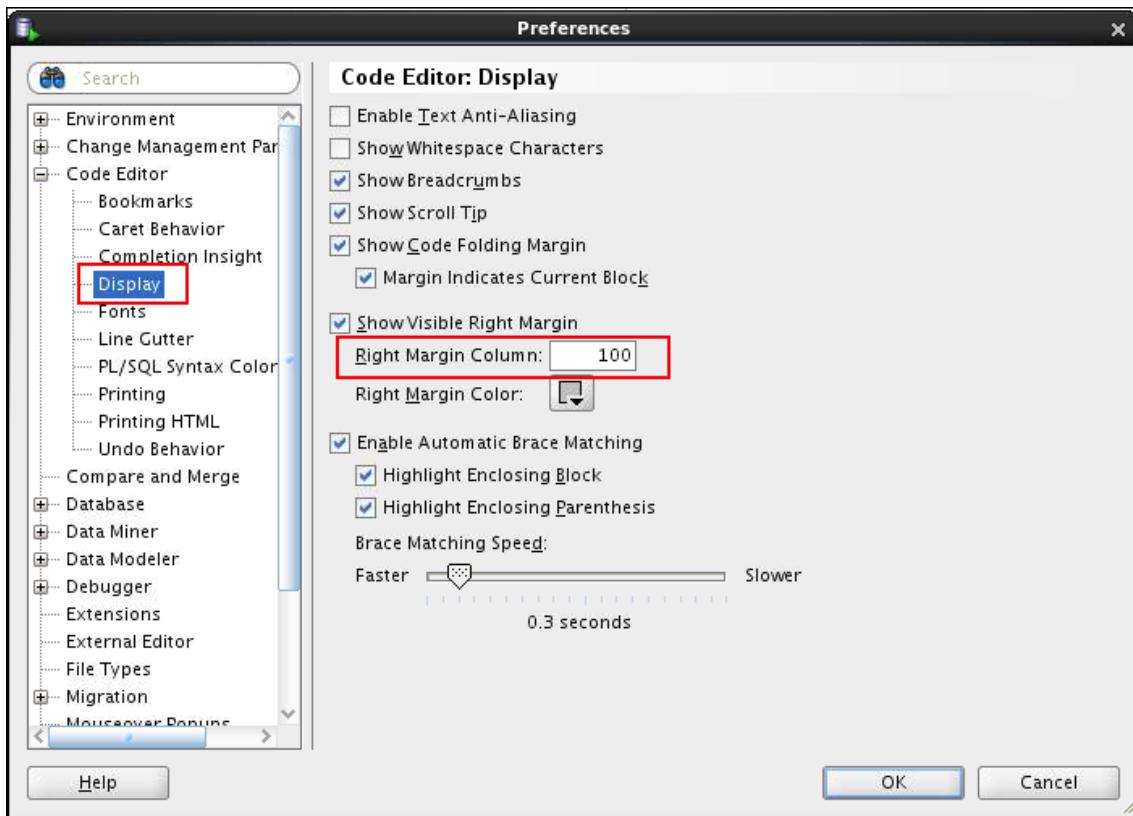
## Übung 4 zu Lektion 1 – Lösung: Verschiedene SQL Developer-Voreinstellungen festlegen

In dieser Übung legen Sie einige SQL Developer-Voreinstellungen fest.

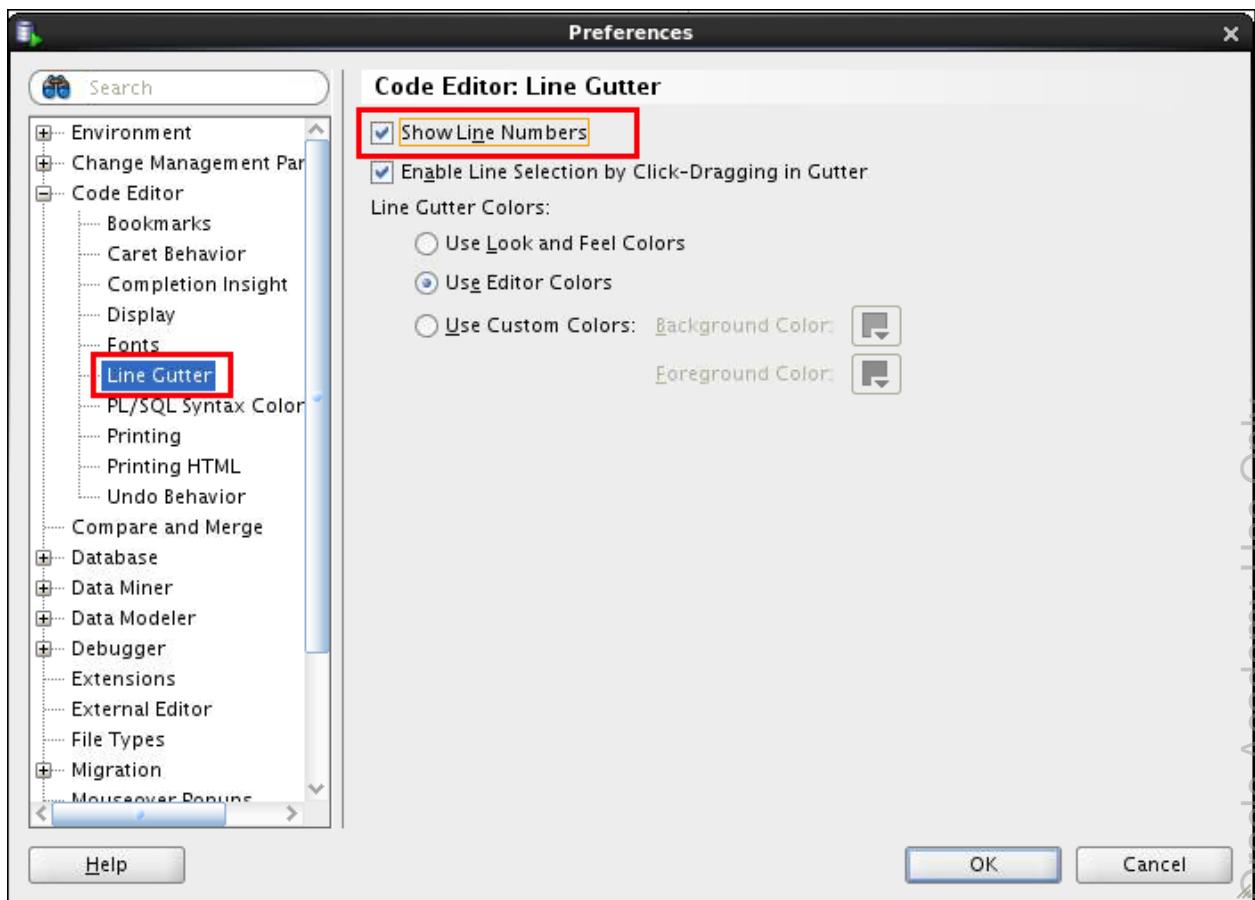
1. Navigieren Sie im SQL Developer-Menü zu **Tools > Preferences**. Das Fenster **Preferences** wird angezeigt.



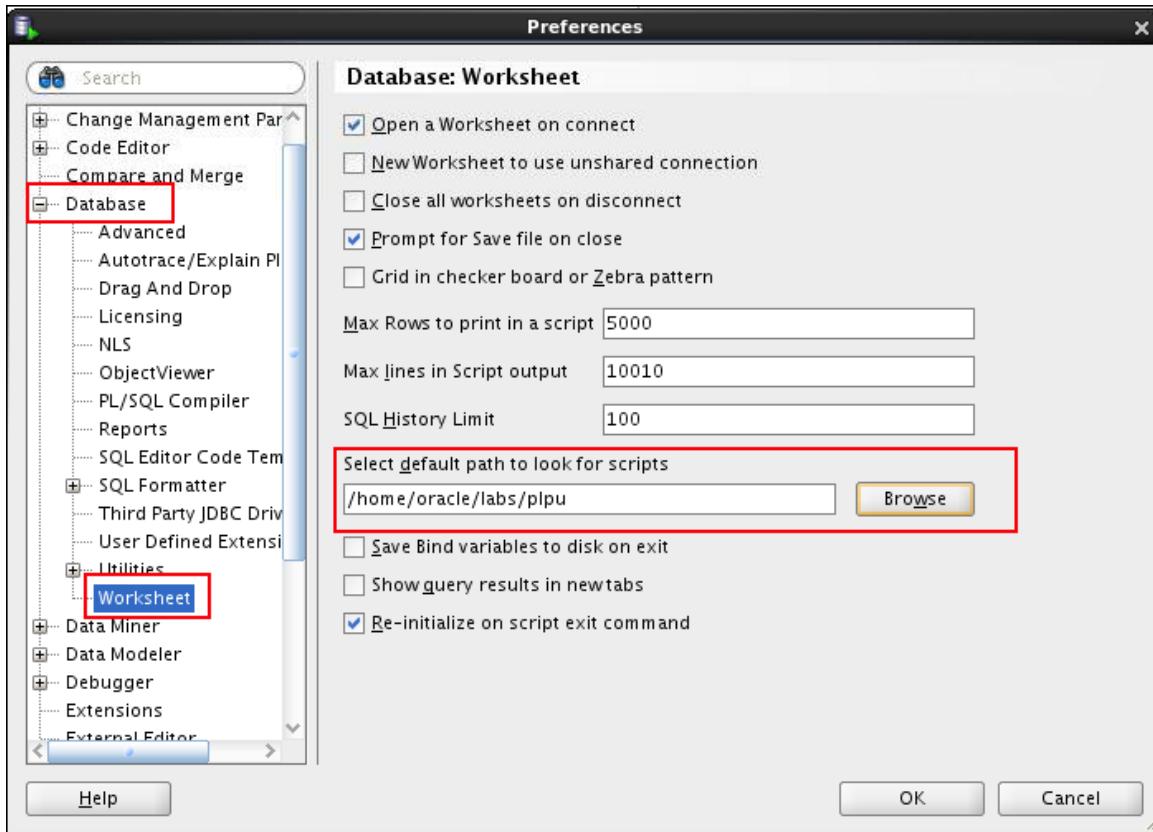
2. Blenden Sie die Option **Code Editor** ein, und klicken Sie auf die Option **Display**, um den Bereich **Code Editor: Display** anzuzeigen. Der Bereich **Code Editor: Display** enthält allgemeine Optionen für die Darstellung und das Verhalten des Code Editors.
  - a. Geben Sie im Bereich **Show Visible Right Margin** im Textfeld **Right Margin Column** den Wert 100 ein. Damit wird ein rechter Rand angezeigt, den Sie zur Steuerung der Länge von Codezeilen einstellen können.



- b. Klicken Sie auf die Option **Line Gutter**. Die Option **Line Gutter** legt die Optionen für den linken Rand des Code Editors fest. Um die Codezeilennummern anzuzeigen, aktivieren Sie das Kontrollkästchen **Show Line Numbers**.



3. Klicken Sie unter der Option **Database** auf die Option **Worksheet Parameters**. Geben Sie im Textfeld **Select default path to look for scripts** das Verzeichnis `/home/oracle/labs/plpu` an. Dieses Verzeichnis enthält die Lösungsskripte, Codebeispielskripte und alle in diesem Kurs verwendeten Übungen oder Demos.



4. Um die Änderungen zu übernehmen und das Fenster **Preferences** zu schließen, klicken Sie auf **OK**.
5. Machen Sie sich im Verzeichnis `/home/oracle/labs/plpu` mit dem Übungsordner **labs** vertraut.
- Klicken Sie (neben der Registerkarte **Connections**) auf die Registerkarte **Files**.
  - Navigieren Sie zum Verzeichnis `/home/oracle/labs/plpu`.
  - Wie viele Unterverzeichnisse enthält das Verzeichnis **labs**?
  - Navigieren Sie durch die Verzeichnisse, und öffnen Sie eine Skriptdatei, ohne den Code auszuführen.

# Übung 5 zu Lektion 1 – Auf die Onlinedokumentenbibliothek von Oracle Database zugreifen

---

## Überblick

In dieser Übung greifen Sie auf einige Verweise für die Oracle Database-Dokumentation zu, die Sie in diesem Kurs verwenden werden, und setzen Lesezeichen.

## Aufgaben

1. Rufen Sie die Webseite der Oracle Database-Dokumentation unter der folgenden Adresse auf: <http://www.oracle.com/pls/db121/homepage>
2. Um künftig schneller auf diese Seite zugreifen zu können, setzen Sie ein Lesezeichen.
3. Zeigen Sie die komplette Liste der Bücher an, die für Oracle Database zur Verfügung stehen.
4. Notieren Sie sich folgende Dokumentationsunterlagen, die Sie im Verlauf dieses Kurses bei Bedarf verwenden werden:
  - a. Advanced Application Developer's Guide
  - b. New Features Guide
  - c. PL/SQL Language Reference
  - d. Oracle Database Reference
  - e. Oracle Database Concepts
  - f. SQL Developer User's Guide
  - g. SQL Language Reference Guide
  - h. SQL\*Plus User's Guide and Reference

## Übung 5 zu Lektion 1 – Lösung: Auf die Onlinedokumentenbibliothek von Oracle Database zugreifen

In dieser Übung greifen Sie auf einige Verweise für die Oracle Database-Dokumentation zu, die Sie in diesem Kurs verwenden werden, und setzen Lesezeichen.

1. Rufen Sie die Webseite der Oracle Database-Dokumentation unter der folgenden Adresse auf:  
<http://www.oracle.com/pls/db121/homepage>
2. Um künftig schneller auf diese Seite zugreifen zu können, setzen Sie ein Lesezeichen.
3. Zeigen Sie die komplette Liste der Bücher an, die für Oracle Database zur Verfügung stehen.
4. Notieren Sie sich folgende Dokumentationsunterlagen, die Sie im Verlauf dieses Kurses bei Bedarf verwenden werden:
  - a. Advanced Application Developer's Guide
  - b. New Features Guide
  - c. PL/SQL Language Reference
  - d. Oracle Database Reference
  - e. Oracle Database Concepts
  - f. SQL Developer User's Guide
  - g. SQL Language Reference Guide
  - h. SQL\*Plus User's Guide and Reference



# **Übungen zu Lektion 2 – Prozeduren erstellen**

## **Kapitel 2**

## Übungen zu Lektion 2: Überblick

---

### Lektionsüberblick

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

#### Hinweis:

1. Führen Sie das Skript  
`/home/oracle/plpu/code_ex/code_ex_scripts/clean_up_scripts/  
cleanup_02.sql` aus, bevor Sie mit dieser Übung beginnen.
2. Wenn Sie einen Schritt in einer Übung ausgelassen haben, führen Sie das entsprechende Lösungsskript zu diesem Übungsschritt aus, bevor Sie mit dem nächsten Schritt oder der nächsten Übung fortfahren.

# Übung 1 zu Lektion 2 – Prozeduren erstellen, kompilieren und aufrufen

---

## Überblick

In dieser Übung erstellen Sie die Prozedur ADD\_JOB und rufen sie auf. Anschließend prüfen Sie die Ergebnisse. Außerdem erstellen Sie eine Prozedur namens UPD\_JOB und rufen sie auf, um eine Tätigkeit in der Tabelle JOBS zu ändern, sowie eine Prozedur namens DEL\_JOB, um eine Tätigkeit aus der Tabelle JOBS zu löschen. Schließlich erstellen Sie eine Prozedur namens GET\_EMPLOYEE, um die Tabelle EMPLOYEES abzufragen und damit das Gehalt sowie die Tätigkeits-ID für einen Mitarbeiter abzurufen, für den die Personalnummer bereitgestellt wurde.

**Hinweis:** Führen Sie die Datei cleanup\_02.sql im Verzeichnis

/home/oracle/plpu/code\_ex/code\_ex\_scripts/clean\_up\_scripts/ aus, bevor Sie die folgende Aufgabe absolvieren.

## Aufgabe

- Erstellen Sie die Prozedur ADD\_JOB, kompilieren Sie sie, und rufen Sie sie auf. Prüfen Sie anschließend die Ergebnisse.

- Erstellen Sie eine Prozedur namens ADD\_JOB, um eine neue Tätigkeit in die Tabelle JOBS einzufügen. Geben Sie die ID und die Tätigkeitsbezeichnung mithilfe von zwei Parametern an.

**Hinweis:** Sie können die Prozedur (und andere Objekte) erstellen, indem Sie den Code im SQL Worksheet-Bereich eingeben und dann auf das Symbol **Run Script** (F5) klicken. Damit wird die Prozedur erstellt und kompiliert. Wenn Sie herausfinden möchten, ob die Prozedur Fehler enthält, klicken Sie im Knoten **Procedures** mit der rechten Maustaste auf den Prozedurnamen und wählen dann im Popup-Menü die Option **Compile**.

- Rufen Sie die Prozedur mit der Tätigkeits-ID IT\_DBA und der Tätigkeitsbezeichnung Database Administrator auf. Fragen Sie die Tabelle JOBS ab, um die Ergebnisse anzuzeigen.

anonymous block completed			
JOB_ID	JOB_TITLE	MIN_SALARY	MAX_SALARY
IT_DBA	Database Administrator		

- Rufen Sie die Prozedur erneut auf, und übergeben Sie dabei die Tätigkeits-ID ST\_MAN und die Tätigkeitsbezeichnung Stock Manager. Was geschieht und warum?
- Erstellen Sie eine Prozedur namens UPD\_JOB, um eine Tätigkeit in der Tabelle JOBS zu ändern.
- Erstellen Sie eine Prozedur namens UPD\_JOB, um die Tätigkeitsbezeichnung zu aktualisieren. Geben Sie die Tätigkeits-ID und eine neue Bezeichnung mithilfe von zwei Parametern an. Nehmen Sie die notwendige Exception-Behandlung für den Fall auf, dass keine Aktualisierung erfolgt.

- b. Rufen Sie die Prozedur auf, um die Tätigkeitsbezeichnung für die Tätigkeits-ID IT\_DBA in Data Administrator zu ändern. Fragen Sie die Tabelle JOBS ab, um die Ergebnisse anzuzeigen.

anonymous block completed		MIN_SALARY	MAX_SALARY
JOB_ID	JOB_TITLE		
IT_DBA	Data Administrator		

- c. Testen Sie den Bereich zur Exception-Behandlung der Prozedur, indem Sie eine nicht vorhandene Tätigkeit aktualisieren. Sie können beispielsweise die Tätigkeits-ID IT\_WEB und die Tätigkeitsbezeichnung Web Master verwenden.

```
Error starting at line 1 in command:
EXECUTE upd_job ('IT_WEB', 'Web Master')
Error report:
ORA-20202: No job updated.
ORA-06512: at "ORA80.UPD_JOB", line 9
ORA-06512: at line 1
```

3. Erstellen Sie eine Prozedur namens DEL\_JOB, um eine Tätigkeit aus der Tabelle JOBS zu löschen.

- a. Erstellen Sie eine Prozedur namens DEL\_JOB, um eine Tätigkeit zu löschen. Nehmen Sie den notwendigen Exception-Behandlungscode für den Fall auf, dass keine Tätigkeit gelöscht wird.
- b. Rufen Sie die Prozedur mit der Tätigkeits-ID IT\_DBA auf. Fragen Sie die Tabelle JOBS ab, um die Ergebnisse anzuzeigen.

```
anonymous block completed
no rows selected
```

- c. Testen Sie den Bereich zur Exception-Behandlung der Prozedur, indem Sie eine nicht vorhandene Tätigkeit löschen. Verwenden Sie die Tätigkeits-ID IT\_WEB. Sie müssten die Meldung erhalten, die Sie im Bereich zur Exception-Behandlung der Prozedur als Ausgabe angegeben haben.

```
Error starting at line 1 in command:
EXECUTE del_job ('IT_WEB')
Error report:
ORA-20203: No jobs deleted.
ORA-06512: at "ORA80.DEL_JOB", line 6
ORA-06512: at line 1
```

4. Erstellen Sie eine Prozedur namens GET\_EMPLOYEE, um die Tabelle EMPLOYEES abzufragen und damit das Gehalt und die Tätigkeits-ID für einen Mitarbeiter abzurufen, für den die Personalnummer bereitgestellt wurde.

- a. Erstellen Sie eine Prozedur, die einen Wert aus den Spalten SALARY und JOB\_ID für eine angegebene Personalnummer zurückgibt. Beheben Sie etwaige Syntaxfehler, und kompilieren Sie den Code dann neu.

- b. Führen Sie die Prozedur mithilfe von Hostvariablen für die beiden OUT-Parameter aus: eine für das Gehalt und eine für die Tätigkeits-ID. Zeigen Sie das Gehalt und die Tätigkeits-ID für die Personalnummer 120 an.

```
v_salary  
----  
8000  
  
v_job  
-----  
ST_MAN
```

- c. Rufen Sie die Prozedur erneut auf. Übergeben Sie dabei die Personalnummer (EMPLOYEE\_ID) 300. Was geschieht und warum?

## Übung 1 zu Lektion 2 – Lösung: Prozeduren erstellen, kompilieren und aufrufen

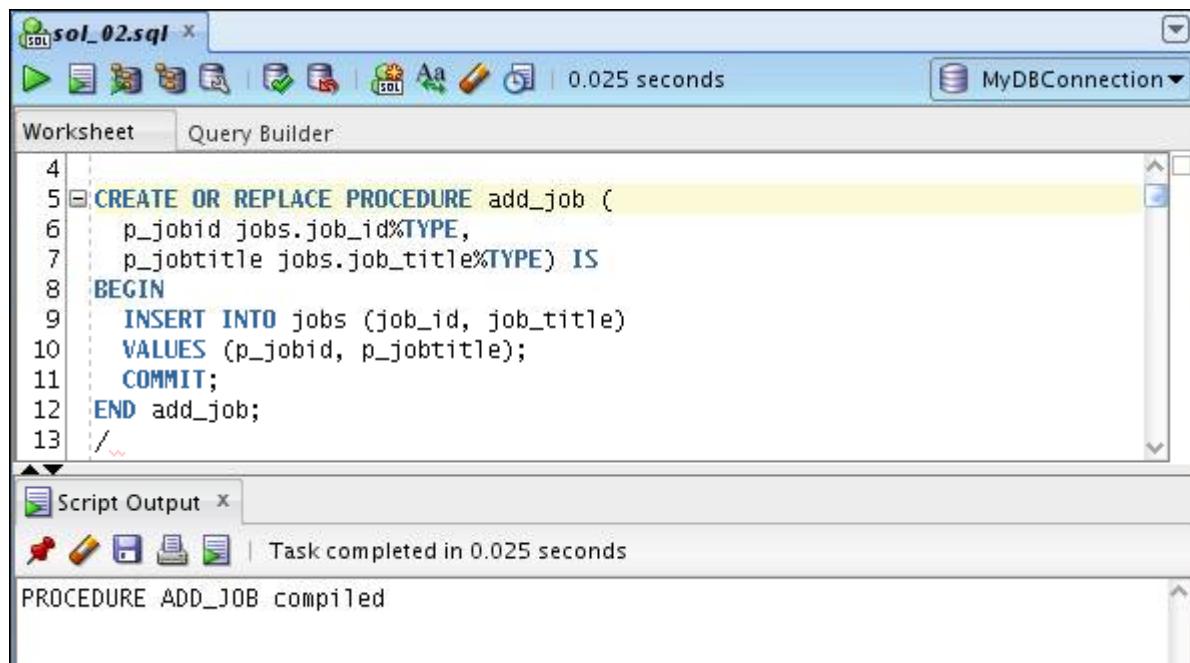
In dieser Übung erstellen Sie die Prozedur ADD\_JOB und rufen sie auf. Anschließend prüfen Sie die Ergebnisse. Außerdem erstellen Sie eine Prozedur namens UPD\_JOB und rufen sie auf, um eine Tätigkeit in der Tabelle JOBS zu ändern, sowie eine Prozedur namens DEL\_JOB, um eine Tätigkeit aus der Tabelle JOBS zu löschen. Schließlich erstellen Sie eine Prozedur namens GET\_EMPLOYEE, um die Tabelle EMPLOYEES abzufragen und damit das Gehalt sowie die Tätigkeits-ID für einen Mitarbeiter abzurufen, für den die Personalnummer bereitgestellt wurde.

1. Erstellen Sie die Prozedur ADD\_JOB, kompilieren Sie sie, und rufen Sie sie auf. Prüfen Sie anschließend die Ergebnisse.

- a. Erstellen Sie eine Prozedur namens ADD\_JOB, um eine neue Tätigkeitsbezeichnung in die Tabelle JOBS einzufügen. Geben Sie die ID und die Tätigkeitsbezeichnung mithilfe von zwei Parametern an.

**Hinweis:** Sie können die Prozedur (und andere Objekte) erstellen, indem Sie den Code im SQL Worksheet-Bereich eingeben und dann auf das Symbol **Run Script** (F5) klicken. Damit wird die Prozedur erstellt und kompiliert. Wenn die Prozedur beim Erstellen eine Fehlermeldung generiert, klicken Sie im Knoten **Procedures** auf den Prozedurnamen, bearbeiten die Prozedur und wählen dann im Popup-Menü die Option **Compile**.

Öffnen Sie im Verzeichnis `/home/oracle/labs/plpu/solns` die Datei `sol_02.sql`. Entfernen Sie die Kommentarzeichen des Codes für Aufgabe 1\_a, und wählen Sie den Code. Um die Prozedur zu erstellen und zu kompilieren, klicken Sie in der SQL Worksheet-Symbolleiste auf das Symbol "Run Script" (F5). Der Code und das Ergebnis werden wie folgt angezeigt:



The screenshot shows the Oracle SQL Worksheet interface. The top window is titled 'sol\_02.sql' and contains the following PL/SQL code:

```

4
5  CREATE OR REPLACE PROCEDURE add_job (
6      p_jobid jobs.job_id%TYPE,
7      p_jobtitle jobs.job_title%TYPE) IS
8      BEGIN
9          INSERT INTO jobs (job_id, job_title)
10         VALUES (p_jobid, p_jobtitle);
11         COMMIT;
12     END add_job;
13   /

```

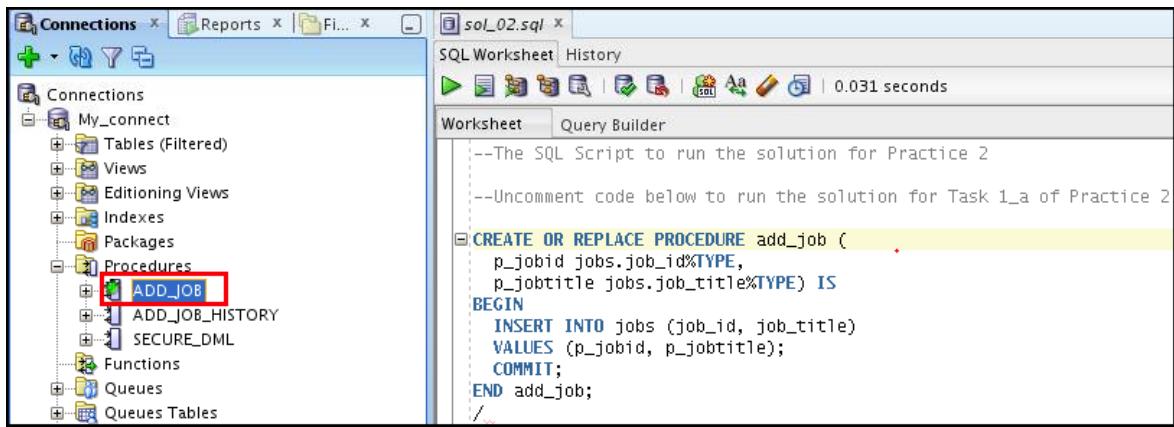
The bottom window is titled 'Script Output' and displays the result of the compilation:

```

Task completed in 0.025 seconds
PROCEDURE ADD_JOB compiled

```

Um die neu erstellte Prozedur anzuzeigen, klicken Sie im Object Navigator auf den Knoten **Procedures**. Wenn die neu erstellte Prozedur nicht angezeigt wird, klicken Sie mit der rechten Maustaste auf den Knoten **Procedures** und wählen dann im Kontextmenü die Option **Refresh**. Die neue Prozedur wird wie folgt angezeigt:



- Rufen Sie die Prozedur mit der Tätigkeits-ID `IT_DBA` und der Tätigkeitsbezeichnung `Database Administrator` auf. Fragen Sie die Tabelle `JOBS` ab, um die Ergebnisse anzuzeigen.

**Führen Sie den Code für Aufgabe 1\_b aus dem Skript sol\_02.sql aus. Der Code und das Ergebnis werden wie folgt angezeigt:**

**Hinweis:** Sie müssen die Kommentarzeichen für den vorhergehenden Code wieder einfügen, bevor Sie die Kommentarzeichen des nächsten Codeblocks entfernen.

```

sol_02.sql x
Worksheet Query Builder
16
17 --Uncomment code below to run the solution for Task 1_b for Practice 2
18
19
20 EXECUTE add_job ('IT_DBA', 'Database Administrator')
21 SELECT * FROM jobs WHERE job_id = 'IT_DBA';
22
23
24

Script Output x
anonymous block completed
JOB_ID      JOB_TITLE          MIN_SALARY MAX_SALARY
-----      -----
IT_DBA      Database Administrator

```

- c. Rufen Sie die Prozedur erneut auf, und übergeben Sie dabei die Tätigkeits-ID ST\_MAN und die Tätigkeitsbezeichnung Stock Manager. Was geschieht und warum?

**Führen Sie den Code für Aufgabe 1\_c aus dem Skript sol\_02.sql aus. Der Code und das Ergebnis werden wie folgt angezeigt:**

**Eine Exception tritt auf, da für die Spalte JOB\_ID ein UNIQUE KEY-Integritäts-Constraint definiert wurde.**

The screenshot shows the Oracle SQL Developer interface. The top window is titled 'sol\_02.sql' and contains the following SQL code:

```

26
27 --Uncomment code below to run the solution for Task 1_c  for Practice 2
28
29 EXECUTE add_job ('ST_MAN', 'Stock Manager')
30
31

```

The bottom window is titled 'Script Output' and displays the execution results:

```

Task completed in 2.324 seconds

Error starting at line 30 in command:
EXECUTE add_job ('ST_MAN', 'Stock Manager')
Error report:
ORA-00001: unique constraint (ORA61.JOB_ID_PK) violated
ORA-06512: at "ORA61.ADD_JOB", line 5
ORA-06512: at line 1
00001. 00000 -  "unique constraint (%s.%s) violated"
*Cause: An UPDATE or INSERT statement attempted to insert a duplicate key.
        For Trusted Oracle configured in DBMS MAC mode, you may see
        this message if a duplicate entry exists at a different level.
*Action: Either remove the unique restriction or do not insert the key.

```

2. Erstellen Sie eine Prozedur namens UPD\_JOB, um eine Tätigkeit in der Tabelle JOBS zu ändern.
- Erstellen Sie eine Prozedur namens UPD\_JOB, um die Tätigkeitsbezeichnung zu aktualisieren. Geben Sie die Tätigkeits-ID und eine neue Bezeichnung mithilfe von zwei Parametern an. Nehmen Sie die notwendige Exception-Behandlung für den Fall auf, dass keine Aktualisierung erfolgt.

Führen Sie den Code für Aufgabe 2\_c aus dem Skript sol\_02.sql aus. Der Code und das Ergebnis werden wie folgt angezeigt:

The screenshot shows the SQL Worksheet interface in SQL Developer. The title bar says "MyDBConnection" and "sol\_02.sql". The worksheet contains the following code:

```

33: --Uncomment code below to run the solution for Task 2_a for Practice 2
34:
35:
36:
37: CREATE OR REPLACE PROCEDURE upd_job (
38:   p_jobid IN jobs.job_id%TYPE,
39:   p_jobtitle IN jobs.job_title%TYPE) IS
40: BEGIN
41:   UPDATE jobs
42:     SET job_title = p_jobtitle
43:   WHERE job_id = p_jobid;
44:   IF SQL%NOTFOUND THEN
45:     RAISE_APPLICATION_ERROR(-20202, 'No job updated.');
46:   END IF;
47: END upd_job;
48:
49:
50: */

```

The "Script Output" tab at the bottom shows the message: "PROCEDURE UPD\_JOB compiled".

- Rufen Sie die Prozedur auf, um die Tätigkeitsbezeichnung für die Tätigkeits-ID IT\_DBA in Data Administrator zu ändern. Fragen Sie die Tabelle JOBS ab, um die Ergebnisse anzuzeigen.

Führen Sie den Code für Aufgabe 2\_b aus dem Skript sol\_02.sql aus. Der Code und das Ergebnis werden wie folgt angezeigt:

The screenshot shows the SQL Worksheet interface in SQL Developer. The title bar says "MyDBConnection". The worksheet contains the following code:

```

1: EXECUTE upd_job ('IT_DBA', 'Data Administrator')
2: SELECT * FROM jobs WHERE job_id = 'IT_DBA';

```

The "Script Output" tab at the bottom shows the message: "anonymous block completed". The "Query Result" tab shows the following data:

JOB_ID	JOB_TITLE	MIN_SALARY	MAX_SALARY
IT_DBA	Data Administrator		

- c. Testen Sie den Bereich zur Exception-Behandlung der Prozedur, indem Sie eine nicht vorhandene Tätigkeit aktualisieren. Sie können beispielsweise die Tätigkeits-ID IT\_WEB und die Tätigkeitsbezeichnung Web Master verwenden.

**Führen Sie den Code für Aufgabe 2\_c aus dem Skript sol\_02.sql aus. Der Code und das Ergebnis werden wie folgt angezeigt:**

The screenshot shows the Oracle SQL Developer interface. The top window is titled 'sol\_02.sql' and contains the following SQL code:

```

59
60
61
62 --Uncomment code below to run the solution for Task 2_c for Practice 2
63
64
65
66 EXECUTE upd_job ('IT_WEB', 'Web Master')
67 SELECT * FROM jobs WHERE job_id = 'IT_WEB';
68
69
70

```

The bottom window is titled 'Query Result' and displays the results of the executed query:

JOB_ID	JOB_TITLE	MIN_SALARY	MAX_SALARY
IT_WEB	Web Master	4500	5500

3. Erstellen Sie eine Prozedur namens DEL\_JOB, um eine Tätigkeit aus der Tabelle JOBS zu löschen.
- Erstellen Sie eine Prozedur namens DEL\_JOB, um eine Tätigkeit zu löschen. Nehmen Sie den notwendigen Exception-Behandlungscode für den Fall auf, dass keine Tätigkeit gelöscht wird.

Führen Sie den Code für Aufgabe 3\_c aus dem Skript sol\_02.sql aus. Der Code und das Ergebnis werden wie folgt angezeigt:

The screenshot shows the Oracle SQL Worksheet interface. The code in the worksheet pane is as follows:

```

70
71 --Uncomment code below to run the solution for Task 3_a for Practice 2
72
73 CREATE OR REPLACE PROCEDURE del_job (p_jobid jobs.job_id%TYPE) IS
74 BEGIN
75   DELETE FROM jobs
76   WHERE job_id = p_jobid;
77   IF SQL%NOTFOUND THEN
78     RAISE_APPLICATION_ERROR(-20203, 'No jobs deleted.');
79   END IF;
80 END DEL_JOB;
81 /

```

The script output pane shows the message "PROCEDURE DEL\_JOB compiled".

- b. Um die Prozedur aufzurufen und anschließend die Tabelle JOBS abzufragen, entfernen Sie die Kommentarzeichen des Codes unter Aufgabe 3\_b im Skript /home/oracle/labs/plpu/solns/sol\_02.sql und wählen den Code. Um die Prozedur aufzurufen, klicken Sie in der SQL Worksheet-Symbolleiste auf das Symbol "Run Script" (F9). Klicken Sie auf die Registerkarte "Query Result", um den Code und das Ergebnis wie folgt anzuseigen:

The screenshot shows the Oracle SQL Worksheet interface with two tabs open: "sol\_02.sql" and "MyDBConnection". The code in the worksheet pane is:

```

86
87 --Uncomment code below to run the solution for Task 3_b for Practice 2
88
89
90 EXECUTE del_job ('IT_DBA')
91 SELECT * FROM jobs WHERE job_id = 'IT_DBA';
92
93

```

The script output pane shows the message "All Rows Fetched: 0 in 0.001 seconds". The query result pane shows the following table:

JOB_ID	JOB_TITLE	MIN_SA...	MAX_SA...

- c. Testen Sie den Bereich zur Exception-Behandlung der Prozedur, indem Sie eine nicht vorhandene Tätigkeit löschen. Verwenden Sie die Tätigkeits-ID IT\_WEB. Sie müssten die Meldung erhalten, die Sie im Bereich zur Exception-Behandlung der Prozedur als Ausgabe angegeben haben.

**Um die Prozedur aufzurufen und anschließend die Tabelle JOBS abzufragen, entfernen Sie die Kommentarzeichen des Codes unter Aufgabe 3\_c im Skript /home/oracle/labs/plpu/solns/sol\_02.sql und wählen den Code. Um die Prozedur aufzurufen, klicken Sie in der SQL Worksheet-Symbolleiste auf das Symbol "Run Script" (F5). Der Code und das Ergebnis werden wie folgt angezeigt:**

The screenshot shows the Oracle SQL Worksheet interface. The title bar says 'MyDBConnection x sol\_02.sql x'. The main area is a 'Worksheet' tab showing the following code:

```

95
96
97 --Uncomment code below to run the solution for Task 3_c for Practice 2
98
99 EXECUTE del_job ('IT_WEB')
100
101
102

```

Below the worksheet is a 'Script Output' tab showing the results of the execution:

```

Task completed in 1.946 seconds
Error starting at line 99 in command:
EXECUTE del_job ('IT_WEB')
Error report:
ORA-20203: No jobs deleted.
ORA-06512: at "ORA61.DEL_JOB", line 6
ORA-06512: at line 1

```

4. Erstellen Sie eine Prozedur namens GET\_EMPLOYEE, um die Tabelle EMPLOYEES abzufragen und damit das Gehalt und die Tätigkeits-ID für einen Mitarbeiter abzurufen, für den die Personalnummer bereitgestellt wurde.
- Erstellen Sie eine Prozedur, die einen Wert aus den Spalten SALARY und JOB\_ID für eine angegebene Personalnummer zurückgibt. Beheben Sie etwaige Syntaxfehler, und kompilieren Sie den Code dann neu.

Entfernen Sie die Kommentarzeichen des Codes für Aufgabe 4\_a aus dem Skript sol\_02.sql, und wählen Sie den Code. Um die Prozedur zu erstellen und zu kompilieren, klicken Sie in der SQL Worksheet-Symbolleiste auf das Symbol "Run Script" (F5). Der Code und das Ergebnis werden wie folgt angezeigt:

The screenshot shows the Oracle SQL Worksheet interface. The main area displays the following PL/SQL code:

```

100 --Uncomment code below to run the solution for Task 4_a  for Practice 2
101
102 CREATE OR REPLACE PROCEDURE get_employee
103   (p_empid IN employees.employee_id%TYPE,
104    p_sal   OUT employees.salary%TYPE,
105    p_job   OUT employees.job_id%TYPE) IS
106 BEGIN
107   SELECT salary, job_id
108     INTO p_sal, p_job
109    FROM employees
110   WHERE employee_id = p_empid;
111 END get_employee;
112 /

```

The code is numbered from 100 to 113. Lines 100-101 are comments. Lines 102-112 define the procedure. Line 113 ends the code block with a slash (/).

Below the code, the "Script Output" window shows the result of the compilation:

```

PROCEDURE GET_EMPLOYEE compiled

```

**Hinweis:** Wenn die neu erstellte Prozedur im Object Navigator nicht angezeigt wird, klicken Sie im Object Navigator mit der rechten Maustaste auf den Knoten **Procedures** und wählen im Kontextmenü die Option **Refresh**. Klicken Sie im Object Navigator mit der rechten Maustaste auf den Namen der Prozedur, und wählen Sie im Kontextmenü die Option **Compile**. Die Prozedur wird kompiliert.



- Führen Sie die Prozedur mithilfe von Hostvariablen für die beiden OUT-Parameter aus: eine für das Gehalt und eine für die Tätigkeits-ID. Zeigen Sie das Gehalt und die Tätigkeits-ID für die Personalnummer 120 an.

Entfernen Sie die Kommentarzeichen des Codes für Aufgabe 4\_b aus dem Skript sol\_02.sql, und wählen Sie den Code. Um die Prozedur aufzurufen, klicken Sie in der SQL Worksheet-Symbolleiste auf das Symbol "Run Script" (F5). Der Code und das Ergebnis werden wie folgt angezeigt:

The screenshot shows the Oracle SQL Worksheet interface. The title bar says 'MyDBConnection x sol\_02.sql x'. The main area is a 'Worksheet' tab showing the following PL/SQL code:

```
--Uncomment code below to run the solution for Task 4_b for Practice 2

VARIABLE v_salary NUMBER
VARIABLE v_job    VARCHAR2(15)
EXECUTE get_employee(120, :v_salary, :v_job)
PRINT v_salary v_job
```

Below the code, the 'Script Output' tab shows the results:

```
anonymous block completed
V_SALARY
-----
8000

V_JOB
-----
ST_MAN
```

- c. Rufen Sie die Prozedur erneut auf. Übergeben Sie dabei die Personalnummer (`EMPLOYEE_ID`) 300. Was geschieht und warum?

**Es gibt in der Tabelle `EMPLOYEES` keinen Mitarbeiter mit der Personalnummer (`EMPLOYEE_ID`) 300. Die Anweisung `SELECT` hat keine Daten aus der Datenbank abgerufen. Dies hat zum folgenden schwerwiegenden PL/SQL-Fehler vom Typ `NO_DATA_FOUND` geführt:**

The screenshot shows the Oracle SQL Developer interface. The top window is titled "sol\_02.sql" and contains a PL/SQL block:

```
--Uncomment code below to run the solution for Task 4_c for Practice 2
VARIABLE v_salary NUMBER
VARIABLE v_job    VARCHAR2(15)
EXECUTE get_employee(300, :v_salary, :v_job)
```

The bottom window is titled "Script Output" and displays the error message:

```
Error starting at line 131 in command:
EXECUTE get_employee(300, :v_salary, :v_job)
Error report:
ORA-01403: no data found
ORA-06512: at "ORA61.GET_EMPLOYEE", line 6
ORA-06512: at line 1
01403. 00000 -  "no data found"
*Cause:
*Action:
```



# **Übungen zu Lektion 3 – Funktionen erstellen und Unterprogramme debuggen**

**Kapitel 3**

## Übungen zu Lektion 3 – Überblick

---

### Lektionsüberblick

In Übung 1 zu Lektion 3 erstellen, kompilieren und verwenden Sie folgende Elemente:

- Eine Funktion namens `GET_JOB`, um eine Tätigkeitsbezeichnung zurückzugeben.
- Eine Funktion namens `GET_ANNUAL_COMP`, um das Jahresgehalt eines Mitarbeiters zurückzugeben, das aus dem Monatsgehalt und der Provision als übergebene Parameter berechnet wird.
- Eine Prozedur namens `ADD_EMPLOYEE`, um einen neuen Mitarbeiter in die Tabelle `EMPLOYEES` einzufügen.

In Übung 2 zu Lektion 3 erhalten Sie eine Einführung in die grundlegende Funktionalität des SQL Developer-Debuggers:

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

### Hinweis:

1. Führen Sie das Skript  
`/home/oracle/plpu/code_ex/code_ex_scripts/clean_up_scripts/cleanup_03.sql` aus, bevor Sie mit dieser Übung beginnen.
2. Wenn Sie einen Schritt in einer Übung ausgelassen haben, führen Sie das entsprechende Lösungsskript zu diesem Übungsschritt aus, bevor Sie mit dem nächsten Schritt oder der nächsten Übung fortfahren.

# Übung 1 zu Lektion 3 – Funktionen erstellen

---

## Überblick

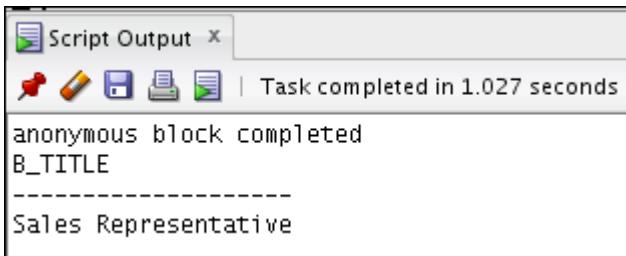
In dieser Übung erstellen, kompilieren und verwenden Sie Stored Functions und eine Prozedur.

**Hinweis:** Führen Sie die Datei `cleanup_03.sql` im Verzeichnis

`/home/oracle/plpu/code_ex/code_ex_scripts/clean_up_scripts/` aus, bevor Sie die folgenden Aufgaben absolvieren.

## Aufgabe

1. Erstellen Sie die Funktion `GET_JOB`, und rufen Sie sie auf, um eine Tätigkeitsbezeichnung zurückzugeben.
  - a. Erstellen Sie eine Funktion namens `GET_JOB`, und kompilieren Sie sie, um eine Tätigkeitsbezeichnung zurückzugeben.
  - b. Erstellen Sie eine `VARCHAR2`-Hostvariable namens `b_title` mit einer Maximallänge von 35 Zeichen. Rufen Sie die Funktion mit der Tätigkeits-ID `SA REP` auf, um einen Wert in der Hostvariablen zurückzugeben. Geben Sie dann die Hostvariable aus, um das Ergebnis anzuzeigen.



```
Script Output x
| Task completed in 1.027 seconds
anonymous block completed
B_TITLE
-----
Sales Representative
```

2. Erstellen Sie eine Funktion namens `GET_ANNUAL_COMP`, um das Jahresgehalt eines Mitarbeiters zurückzugeben, das aus dem Monatsgehalt und der Provision als übergebene Parameter berechnet wird.
  - a. Erstellen Sie die Funktion `GET_ANNUAL_COMP`, die Parameterwerte für das Monatsgehalt und die Provision annimmt. Einer oder beide der übergebenen Werte können `NULL` sein. Die Funktion soll dennoch ein Jahresgehalt zurückgeben, das nicht `NULL` ist. Berechnen Sie das Jahresgehalt mit der folgenden Basisformel:

$$(\text{salary} * 12) + (\text{commission_pct} * \text{salary} * 12)$$

- b. Verwenden Sie die Funktion in einer SELECT-Anweisung für die Tabelle EMPLOYEES für Mitarbeiter der Abteilung 30.

EMPLOYEE_ID	LAST_NAME	Annual Compensation
114	Raphaely	132000
115	Khoo	37200
116	Baida	34800
117	Tobias	33600
118	Himuro	31200
119	Colmenares	30000

6 rows selected

3. Erstellen Sie eine Prozedur namens ADD\_EMPLOYEE, um einen neuen Mitarbeiter in die Tabelle EMPLOYEES einzufügen. Die Prozedur soll die Funktion VALID\_DEPTID aufrufen, um zu prüfen, ob die für den neuen Mitarbeiter angegebene Abteilungsnummer in der Tabelle DEPARTMENTS vorhanden ist.
- Erstellen Sie zur Prüfung einer angegebenen Abteilungsnummer eine Funktion namens VALID\_DEPTID, die den BOOLEAN-Wert TRUE zurückgibt, wenn die Abteilung vorhanden ist.
  - Erstellen Sie die Prozedur ADD\_EMPLOYEE, um einen Mitarbeiter in die Tabelle EMPLOYEES einzufügen. Die Zeile soll der Tabelle EMPLOYEES hinzugefügt werden, wenn die Funktion VALID\_DEPTID den Wert TRUE zurückgibt. Andernfalls soll der Benutzer mit einer entsprechenden Meldung gewarnt werden. Geben Sie die folgenden Parameter an:
    - first\_name
    - last\_name
    - email
    - job: Verwenden Sie 'SA\_REP' als Standard.
    - mgr: Verwenden Sie 145 als Standard.
    - sal: Verwenden Sie 1000 als Standard.
    - comm: Verwenden Sie 0 als Standard.
    - deptid: Verwenden Sie 30 als Standard.
    - Verwenden Sie die Sequence EMPLOYEES\_SEQ, um die Spalte employee\_id festzulegen.
    - Stellen Sie die Spalte hire\_date auf TRUNC(SYSDATE) ein.
  - Rufen Sie ADD\_EMPLOYEE für den Namen 'Jane Harris' in Abteilung 15 auf. Behalten Sie für die übrigen Parameter die Standardwerte bei. Wie lautet das Ergebnis?
  - Fügen Sie einen weiteren Mitarbeiter namens Joe Harris in Abteilung 80 hinzu. Behalten Sie für die übrigen Parameter die Standardwerte bei. Wie lautet das Ergebnis?

## Übung 1 zu Lektion 3 – Lösung: Funktionen erstellen

In dieser Übung erstellen, kompilieren und verwenden Sie Stored Functions und eine Prozedur.

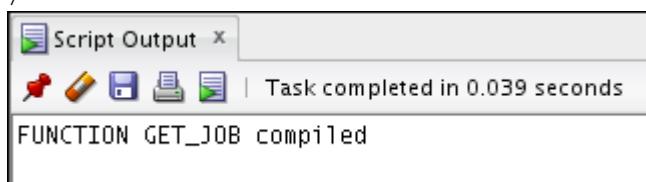
1. Erstellen Sie die Funktion GET\_JOB, und rufen Sie sie auf, um eine Tätigkeitsbezeichnung zurückzugeben.

- a. Erstellen Sie eine Funktion namens GET\_JOB, und kompilieren Sie sie, um eine Tätigkeitsbezeichnung zurückzugeben.

**Öffnen Sie das Skript /home/oracle/labs/plpu/solns/sol\_03.sql.**

**Entfernen Sie die Kommentarzeichen, und wählen Sie den Code unter der Aufgabe 1\_a. Um die Funktion zu erstellen und zu kompilieren, klicken Sie in der SQL Worksheet-Symbolleiste auf das Symbol "Run Script" (F5). Der Code und das Ergebnis werden wie folgt angezeigt:**

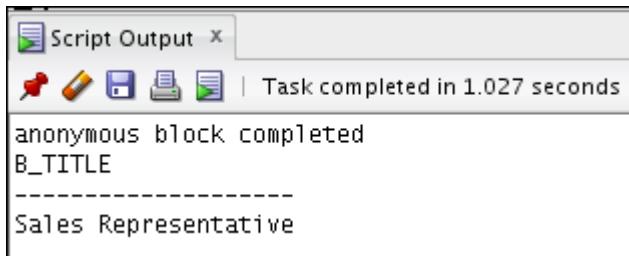
```
CREATE OR REPLACE FUNCTION get_job (p_jobid IN jobs.job_id%type)
  RETURN jobs.job_title%type IS
    v_title jobs.job_title%type;
BEGIN
  SELECT job_title
  INTO v_title
  FROM jobs
  WHERE job_id = p_jobid;
  RETURN v_title;
END get_job;
/
```



- b. Erstellen Sie eine VARCHAR2-Hostvariable namens b\_title mit einer Maximallänge von 35 Zeichen. Rufen Sie die Funktion mit der Tätigkeits-ID SA REP auf, um einen Wert in der Hostvariablen zurückzugeben. Geben Sie dann die Hostvariable aus, um das Ergebnis anzuzeigen.

**Entfernen Sie die Kommentarzeichen des Codes unter Aufgabe 1\_b, und wählen Sie den Code. Um die Funktion zu erstellen und zu kompilieren, klicken Sie in der SQL Worksheet-Symbolleiste auf das Symbol "Run Script" (F5). Der Code und das Ergebnis werden wie folgt angezeigt:**

```
VARIABLE b_title VARCHAR2(35)
EXECUTE :b_title := get_job ('SA REP');
PRINT b_title
```



**Hinweis:** Sie müssen die Kommentarzeichen für den vorhergehenden Code wieder einfügen, bevor Sie die Kommentarzeichen des nächsten Codeblocks entfernen. Alternativ können Sie den gesamten Code wählen, bevor Sie ihn mit dem Symbol **Run Script** (F5) ausführen.

2. Erstellen Sie eine Funktion namens GET\_ANNUAL\_COMP, um das Jahresgehalt eines Mitarbeiters zurückzugeben, das aus dem Monatsgehalt und der Provision als übergebene Parameter berechnet wird.
  - a. Erstellen Sie die Funktion GET\_ANNUAL\_COMP, die Parameterwerte für das Monatsgehalt und die Provision annimmt. Einer oder beide der übergebenen Werte können NULL sein. Die Funktion soll dennoch ein Jahresgehalt zurückgeben, das nicht NULL ist. Berechnen Sie das Jahresgehalt mit der folgenden Basisformel:

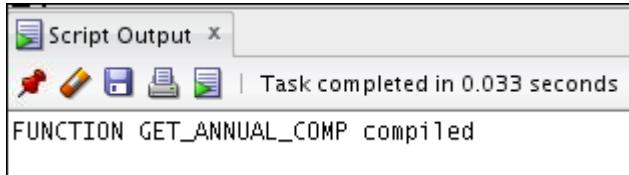
$$(salary * 12) + (commission_pct * salary * 12)$$

**Entfernen Sie die Kommentarzeichen, und wählen Sie den Code unter der Aufgabe 2\_a. Um die Funktion zu erstellen und zu kompilieren, klicken Sie in der SQL Worksheet-Symbolleiste auf das Symbol "Run Script" (F5). Der Code und das Ergebnis werden wie folgt angezeigt:**

```

CREATE OR REPLACE FUNCTION get_annual_comp(
    p_sal IN employees.salary%TYPE,
    p_comm IN employees.commission_pct%TYPE)
RETURN NUMBER IS
BEGIN
    RETURN (NVL(p_sal, 0) * 12 + (NVL(p_comm, 0) * nvl(p_sal, 0) *
12));
END get_annual_comp;
/

```



- b. Verwenden Sie die Funktion in einer SELECT-Anweisung für die Tabelle EMPLOYEES für Mitarbeiter der Abteilung 30.

**Entfernen Sie die Kommentarzeichen, und wählen Sie den Code unter der Aufgabe 2\_b. Um die Funktion zu erstellen und zu komplizieren, klicken Sie in der SQL Worksheet-Symbolleiste auf das Symbol "Run Script" (F5). Der Code und das Ergebnis werden wie folgt angezeigt:**

```
SELECT employee_id, last_name,
       get_annual_comp(salary,commission_pct) "Annual
Compensation"
  FROM employees
 WHERE department_id=30
 /
```

EMPLOYEE_ID	LAST_NAME	Annual Compensation
114	Raphaely	132000
115	Khoo	37200
116	Baida	34800
117	Tobias	33600
118	Himuro	31200
119	Colmenares	30000

6 rows selected

3. Erstellen Sie eine Prozedur namens ADD\_EMPLOYEE, um einen neuen Mitarbeiter in die Tabelle EMPLOYEES einzufügen. Die Prozedur soll die Funktion VALID\_DEPTID aufrufen, um zu prüfen, ob die für den neuen Mitarbeiter angegebene Abteilungsnummer in der Tabelle DEPARTMENTS vorhanden ist.

- a. Erstellen Sie zur Prüfung einer angegebenen Abteilungsnummer eine Funktion namens VALID\_DEPTID, die den BOOLEAN-Wert TRUE zurückgibt, wenn die Abteilung vorhanden ist.

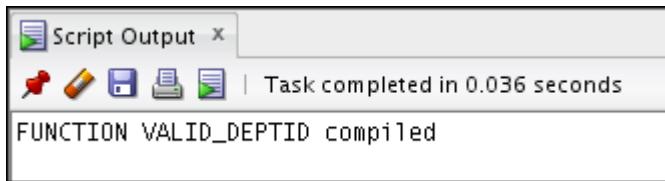
**Entfernen Sie die Kommentarzeichen, und wählen Sie den Code unter der Aufgabe 3\_a. Um die Funktion zu erstellen, klicken Sie in der SQL Worksheet-Symbolleiste auf das Symbol "Run Script" (F5). Der Code und das Ergebnis werden wie folgt angezeigt:**

```
CREATE OR REPLACE FUNCTION valid_deptid(
  p_deptid IN departments.department_id%TYPE)
  RETURN BOOLEAN IS
  v_dummy PLS_INTEGER;
```

```

BEGIN
    SELECT 1
    INTO v_dummy
    FROM departments
    WHERE department_id = p_deptid;
    RETURN TRUE;
EXCEPTION
    WHEN NO_DATA_FOUND THEN
        RETURN FALSE;
END valid_deptid;
/

```



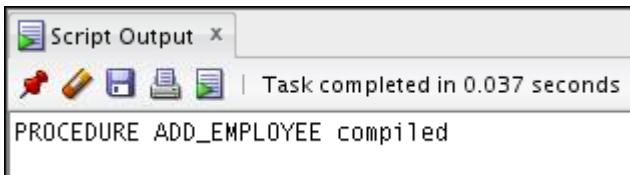
- b. Erstellen Sie die Prozedur ADD\_EMPLOYEE, um einen Mitarbeiter in die Tabelle EMPLOYEES einzufügen. Die Zeile soll der Tabelle EMPLOYEES hinzugefügt werden, wenn die Funktion VALID\_DEPTID den Wert TRUE zurückgibt. Andernfalls soll der Benutzer mit einer entsprechenden Meldung gewarnt werden. Geben Sie die folgenden Parameter an:
- first\_name
  - last\_name
  - email
  - job: Verwenden Sie 'SA\_REP' als Standard.
  - mgr: Verwenden Sie 145 als Standard.
  - sal: Verwenden Sie 1000 als Standard.
  - comm: Verwenden Sie 0 als Standard.
  - deptid: Verwenden Sie 30 als Standard.
  - Verwenden Sie die Sequence EMPLOYEES\_SEQ, um die Spalte employee\_id festzulegen.
  - Stellen Sie die Spalte hire\_date auf TRUNC(SYSDATE) ein.

**Entfernen Sie die Kommentarzeichen, und wählen Sie den Code unter der Aufgabe 3\_b. Um die Prozedur zu erstellen und zu kompilieren, klicken Sie in der SQL Worksheet-Symbolleiste auf das Symbol "Run Script" (F5). Der Code und das Ergebnis werden wie folgt angezeigt:**

```

CREATE OR REPLACE PROCEDURE add_employee(
    p_first_name employees.first_name%TYPE,
    p_last_name  employees.last_name%TYPE,
    p_email      employees.email%TYPE,
    p_job        employees.job_id%TYPE          DEFAULT 'SA_REP',
    p_mgr        employees.manager_id%TYPE       DEFAULT 145,
    p_sal        employees.salary%TYPE          DEFAULT 1000,
    p_comm       employees.commission_pct%TYPE  DEFAULT 0,
    p_deptid     employees.department_id%TYPE   DEFAULT 30) IS
BEGIN
    IF valid_deptid(p_deptid) THEN
        INSERT INTO employees(employee_id, first_name, last_name,
email,
                job_id, manager_id, hire_date, salary, commission_pct,
department_id)
        VALUES (employees_seq.NEXTVAL, p_first_name, p_last_name,
p_email,
                p_job, p_mgr, TRUNC(SYSDATE), p_sal, p_comm, p_deptid);
    ELSE
        RAISE_APPLICATION_ERROR (-20204, 'Invalid department ID. Try
again.');
    END IF;
END add_employee;
/

```



- c. Rufen Sie ADD\_EMPLOYEE für den Namen 'Jane Harris' in Abteilung 15 auf. Behalten Sie für die übrigen Parameter die Standardwerte bei. Wie lautet das Ergebnis?

**Entfernen Sie die Kommentarzeichen des Codes unter Aufgabe 3\_c, und wählen Sie den Code. Um die Prozedur aufzurufen, klicken Sie in der SQL Worksheet-Symbolleiste auf das Symbol "Run Script" (F5). Der Code und das Ergebnis werden wie folgt angezeigt:**

```
EXECUTE add_employee('Jane','Harris','JAHARRIS',p_deptid=> 15)
```

The screenshot shows the 'Script Output' window from the Oracle SQL Worksheet. It displays the following text:

```
Script Output X
Task completed in 1.06 seconds
Error starting at line 109 in command:
EXECUTE add_employee('Jane', 'Harris', 'JAHARRIS', p_deptid=> 15)
Error report:
ORA-20204: Invalid department ID. Try again.
ORA-06512: at "ORA61.ADD_EMPLOYEE", line 17
ORA-06512: at line 1
```

- d. Fügen Sie einen weiteren Mitarbeiter namens Joe Harris in Abteilung 80 hinzu. Behalten Sie für die übrigen Parameter die Standardwerte bei. Wie lautet das Ergebnis?

**Entfernen Sie die Kommentarzeichen des Codes unter Aufgabe 3\_d, und wählen Sie den Code. Um die Prozedur aufzurufen, klicken Sie in der SQL Worksheet-Symbolleiste auf das Symbol "Run Script" (F5). Der Code und das Ergebnis werden wie folgt angezeigt:**

```
EXECUTE add_employee('Joe', 'Harris', 'JAHARRIS', p_deptid=> 80)
```

The screenshot shows the 'Script Output' window from the Oracle SQL Worksheet. It displays the following text:

```
Script Output X
Task completed in 1.041 seconds
anonymous block completed
```

## Übung 2 zu Lektion 3 – SQL Developer-Debugger – Einführung

---

### Überblick

In dieser Übung experimentieren Sie mit der grundlegenden Funktionalität des SQL Developer-Debuggers.

### Aufgaben

1. Aktivieren Sie SERVEROUTPUT.
2. Um die Prozedur `emp_list` zu erstellen, führen Sie die Lösung unter der 2. Aufgabe von Übung 2 zu Lektion 3 aus. Untersuchen Sie den Code der Prozedur, und kompilieren Sie die Prozedur. Warum wird ein Compilerfehler ausgegeben?
3. Um die Prozedur `get_location` zu erstellen, führen Sie die Lösung unter der 3. Aufgabe von Übung 2 zu Lektion 3 aus. Untersuchen Sie den Code der Funktion, kompilieren Sie die Funktion, und beheben Sie dann etwaige Fehler.
4. Rekompilieren Sie die Prozedur `emp_list`. Die Prozedur sollte erfolgreich kompiliert werden.
5. Bearbeiten Sie die Prozedur `emp_list` und die Funktion `get_location`.
6. Fügen Sie der Prozedur `emp_list` in folgenden Codezeilen vier Breakpoints hinzu:
  - a. `OPEN cur_emp;`
  - b. `WHILE (cur_emp%FOUND) AND (i <= p_maxrows) LOOP`
  - c. `v_city := get_location (rec_emp.department_name);`
  - d. `CLOSE cur_emp;`
7. Kompilieren Sie die Prozedur `emp_list` zum Debuggen.
8. Debuggen Sie die Prozedur.
9. Geben Sie 100 als Wert des Parameters `PMAXROWS` ein.
10. Untersuchen Sie den Wert der Variablen in der Registerkarte **Data**. Welche Werte sind `REC_EMP` und `EMP_TAB` zugewiesen? Nennen Sie die Gründe.
11. Verwenden Sie die Debugoption **Step Into**, um in `emp_list` in alle Codezeilen hineinzugehen und die gesamte Schleife nur einmal zu durchlaufen.
12. Untersuchen Sie den Wert der Variablen in der Registerkarte **Data**. Welche Werte sind `REC_EMP` zugewiesen?
13. Drücken Sie weiterhin F7, bis die Zeile `emp_tab(i) := rec_emp;` ausgeführt wird. Untersuchen Sie den Wert der Variablen in der Registerkarte **Data**. Welche Werte sind `EMP_TAB` zugewiesen?
14. Ändern Sie den Wert des Zählers "i" mit der Registerkarte **Data** in 98.
15. Drücken Sie weiterhin F7, bis Sie feststellen, dass die Liste der Mitarbeiter in der Registerkarte **Debugging – Log** angezeigt wird. Wie viele Mitarbeiter werden angezeigt?
16. Wenn Sie den Code mit der Debugger-Option **Step Over** ablaufen lassen, läuft dann auch die Funktion `get_location` ab? Warum oder warum nicht?

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

---

In dieser Übung experimentieren Sie mit der grundlegenden Funktionalität des SQL Developer-Debuggers.

1. Aktivieren Sie SERVEROUTPUT.

Geben Sie folgenden Befehl im SQL Worksheet-Bereich ein, und klicken Sie in der SQL Worksheet-Symbolleiste auf das Symbol **Run Script** (F5).

```
SET SERVEROUTPUT ON
```

2. Um die Prozedur `emp_list` zu erstellen, führen Sie die Lösung unter der 2. Aufgabe von Übung 2 zu Lektion 3 aus. Untersuchen Sie den Code der Prozedur, und kompilieren Sie die Prozedur. Warum wird ein Compilerfehler ausgegeben?

**Entfernen Sie die Kommentarzeichen des Codes unter der 2. Aufgabe von Übung 2 zu Lektion 3, und wählen Sie den Code. Um die Prozedur zu erstellen und zu kompilieren, klicken Sie in der SQL Worksheet-Symbolleiste auf das Symbol "Run Script" (F5). Der Code und das Ergebnis werden wie folgt angezeigt:**

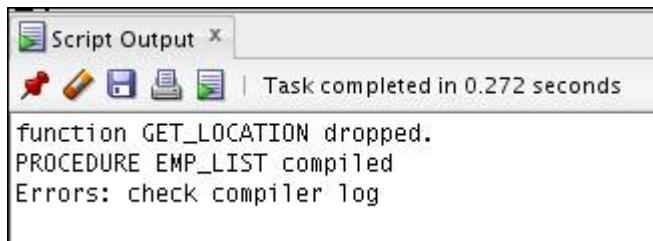
```
DROP FUNCTION get_location;

CREATE OR REPLACE PROCEDURE emp_list
(p_maxrows IN NUMBER)
IS
CURSOR cur_emp IS
SELECT d.department_name, e.employee_id, e.last_name,
       e.salary, e.commission_pct
  FROM departments d, employees e
 WHERE d.department_id = e.department_id;
rec_emp cur_emp%ROWTYPE;
TYPE emp_tab_type IS TABLE OF cur_emp%ROWTYPE INDEX BY
BINARY_INTEGER;
emp_tab emp_tab_type;
i NUMBER := 1;
v_city VARCHAR2(30);
BEGIN
OPEN cur_emp;
FETCH cur_emp INTO rec_emp;
emp_tab(i) := rec_emp;
WHILE (cur_emp%FOUND) AND (i <= p_maxrows) LOOP
  i := i + 1;
  FETCH cur_emp INTO rec_emp;
  emp_tab(i) := rec_emp;
  v_city := get_location (rec_emp.department_name);
```

```

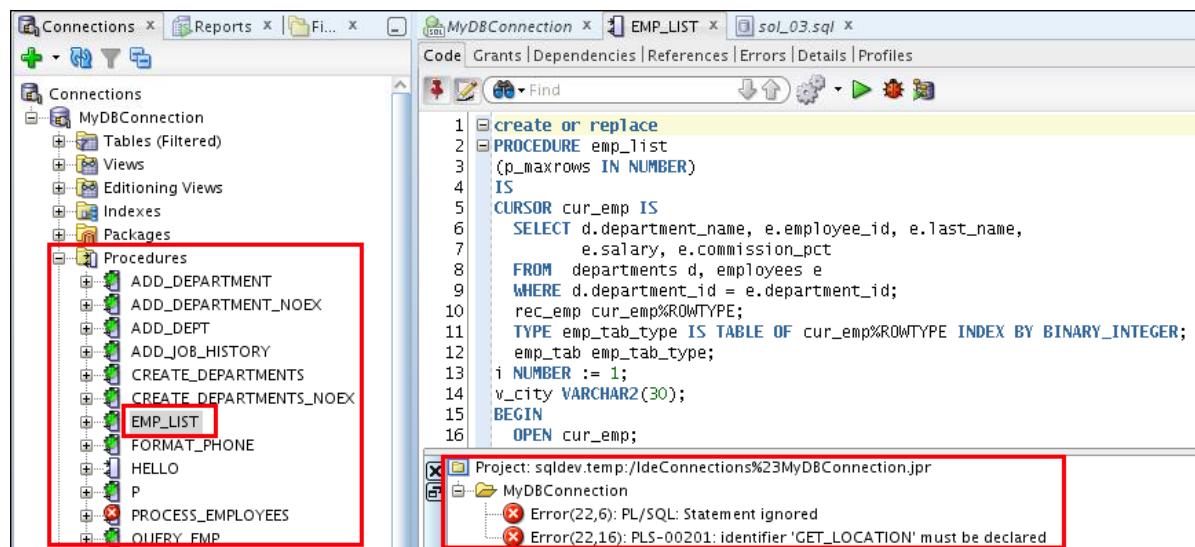
        dbms_output.put_line('Employee ' || rec_emp.last_name ||
        ' works in ' || v_city );
    END LOOP;
CLOSE cur_emp;
FOR j IN REVERSE 1..i LOOP
    DBMS_OUTPUT.PUT_LINE(emp_tab(j).last_name);
END LOOP;
END emp_list;
/

```



**Hinweis:** Sie müssen unter Umständen mit einer Fehlermeldung in der Ausgabe rechnen, wenn die Funktion `get_location` nicht vorhanden ist. Wenn die Funktion vorhanden ist, erhalten Sie die oben aufgeführte Ausgabe.

Die Kompilierungswarnung wird ausgegeben, weil die Funktion `get_location` noch nicht deklariert ist. Um den Kompilierungsfehler detaillierter anzuzeigen, klicken Sie mit der rechten Maustaste im Knoten **Procedures** auf die Prozedur `EMP_LIST`. (Eventuell müssen Sie die Prozedurliste aktualisieren, um die neu erstellte Prozedur `EMP_LIST` anzuzeigen.) Wählen Sie dann im Popup-Menü die Option **Compile**. Die detaillierten Warnmeldungen werden in der Registerkarte **Compiler – Log** wie folgt angezeigt:

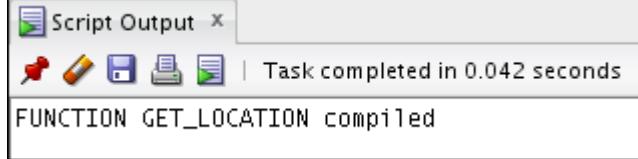


3. Um die Prozedur `get_location` zu erstellen, führen Sie die Lösung unter der 3. Aufgabe von Übung 2 zu Lektion 3 aus. Untersuchen Sie den Code der Funktion, kompilieren Sie die Funktion, und beheben Sie dann etwaige Fehler.

**Entfernen Sie die Kommentarzeichen des Codes unter der 3. Aufgabe von Übung 2 zu Lektion 3, und wählen Sie den Code. Um die Prozedur zu erstellen und zu kompilieren, klicken Sie in der SQL Worksheet-Symbolleiste auf das Symbol "Run Script" (F5). Der Code und das Ergebnis werden wie folgt angezeigt:**

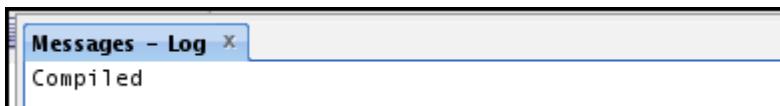
```
CREATE OR REPLACE FUNCTION get_location
( p_deptname IN VARCHAR2 ) RETURN VARCHAR2
AS
    v_loc_id NUMBER;
    v_city    VARCHAR2 (30);
BEGIN
    SELECT d.location_id, l.city INTO v_loc_id, v_city
    FROM departments d, locations l
    WHERE upper(department_name) = upper(p_deptname)
    and d.location_id = l.location_id;
    RETURN v_city;
END GET_LOCATION;
/

```



4. Rekompilieren Sie die Prozedur `emp_list`. Die Prozedur sollte erfolgreich kompiliert werden.

Um die Prozedur zu rekompilieren, klicken Sie mit der rechten Maustaste auf den Namen der Prozedur und wählen dann im Kontextmenü die Option **Compile**.



5. Bearbeiten Sie die Prozedur `emp_list` und die Funktion `get_location`.

Klicken Sie im Object Navigator mit der rechten Maustaste auf den Prozedurnamen `emp_list`, und wählen Sie **Edit**. Die Prozedur `emp_list` wird im Bearbeitungsmodus geöffnet. Wenn die Prozedur im SQL Worksheet-Bereich bereits angezeigt wird, sich aber im schreibgeschützten Modus befindet, klicken Sie in der Registerkarte **Code** auf das Symbol **Edit** (Bleistiftsymbol).

**Klicken Sie im Object Navigator mit der rechten Maustaste auf den Funktionsnamen `get_location`, und wählen Sie "Edit". Die Funktion `get_location` wird im Bearbeitungsmodus geöffnet. Wenn die Funktion im SQL Worksheet-Bereich bereits angezeigt wird, sich aber im schreibgeschützten Modus befindet, klicken Sie in der Registerkarte **Code** auf das Symbol "Edit" (Bleistiftsymbol).**

6. Fügen Sie der Prozedur emp\_list in folgenden Codezeilen vier Breakpoints hinzu:

- a. OPEN cur\_emp;
- b. WHILE (cur\_emp%FOUND) AND (i <= p\_maxrows) LOOP
- c. v\_city := get\_location (rec\_emp.department\_name);
- d. CLOSE cur\_emp;

**Um einen Breakpoint hinzuzufügen, klicken Sie neben den oben angegebenen Zeilen auf den linken Rand, wie in der Abbildung unten gezeigt:**

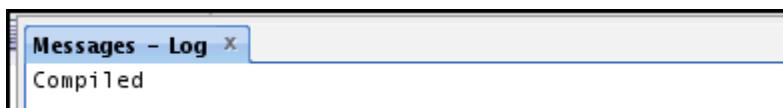
```

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

```

7. Kompilieren Sie die Prozedur emp\_list zum Debuggen.

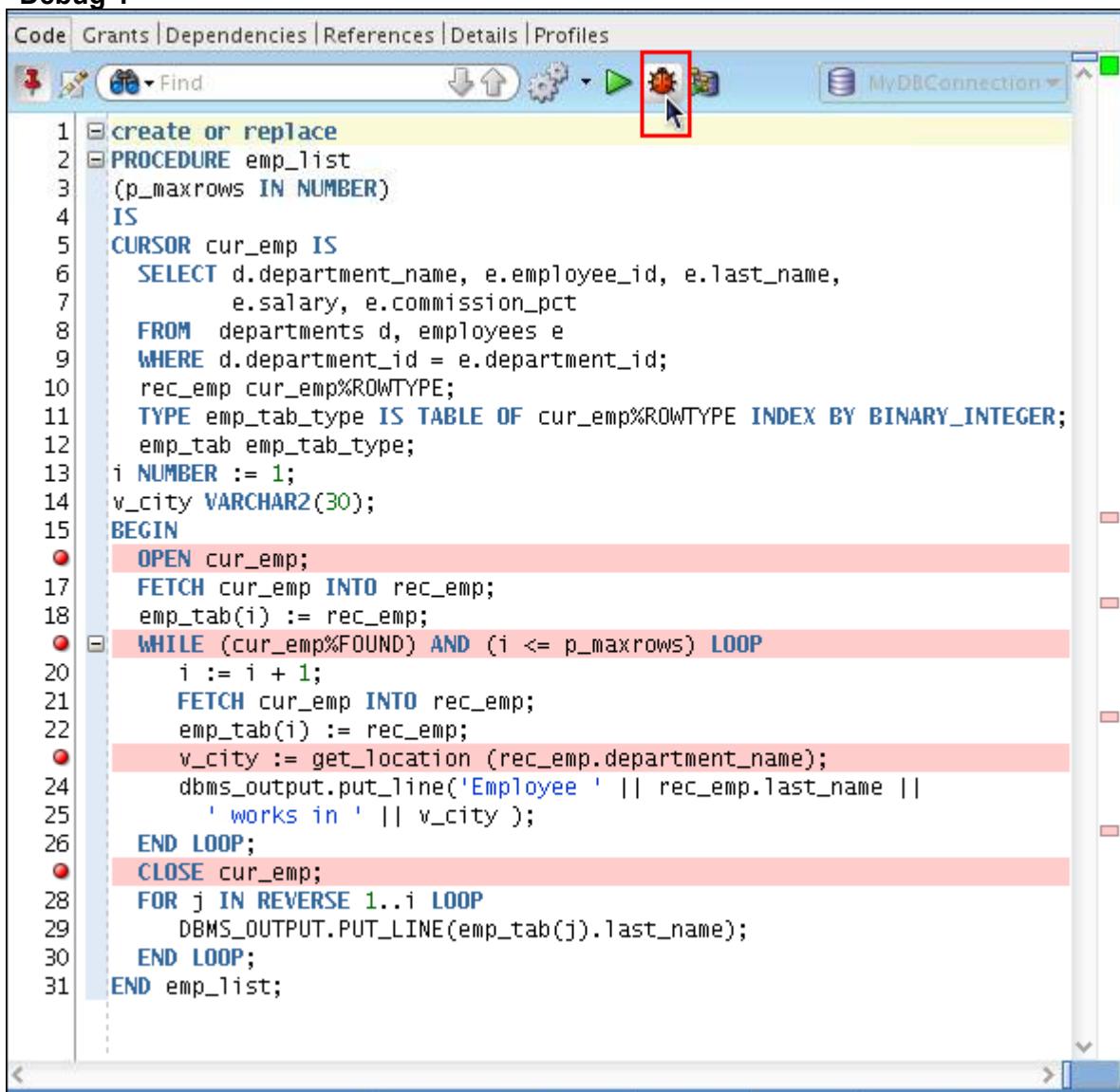
**Klicken Sie in der Symbolleiste der Prozedur auf das Symbol "Compile for Debug".  
Sie erhalten dann folgende Ausgabe:**

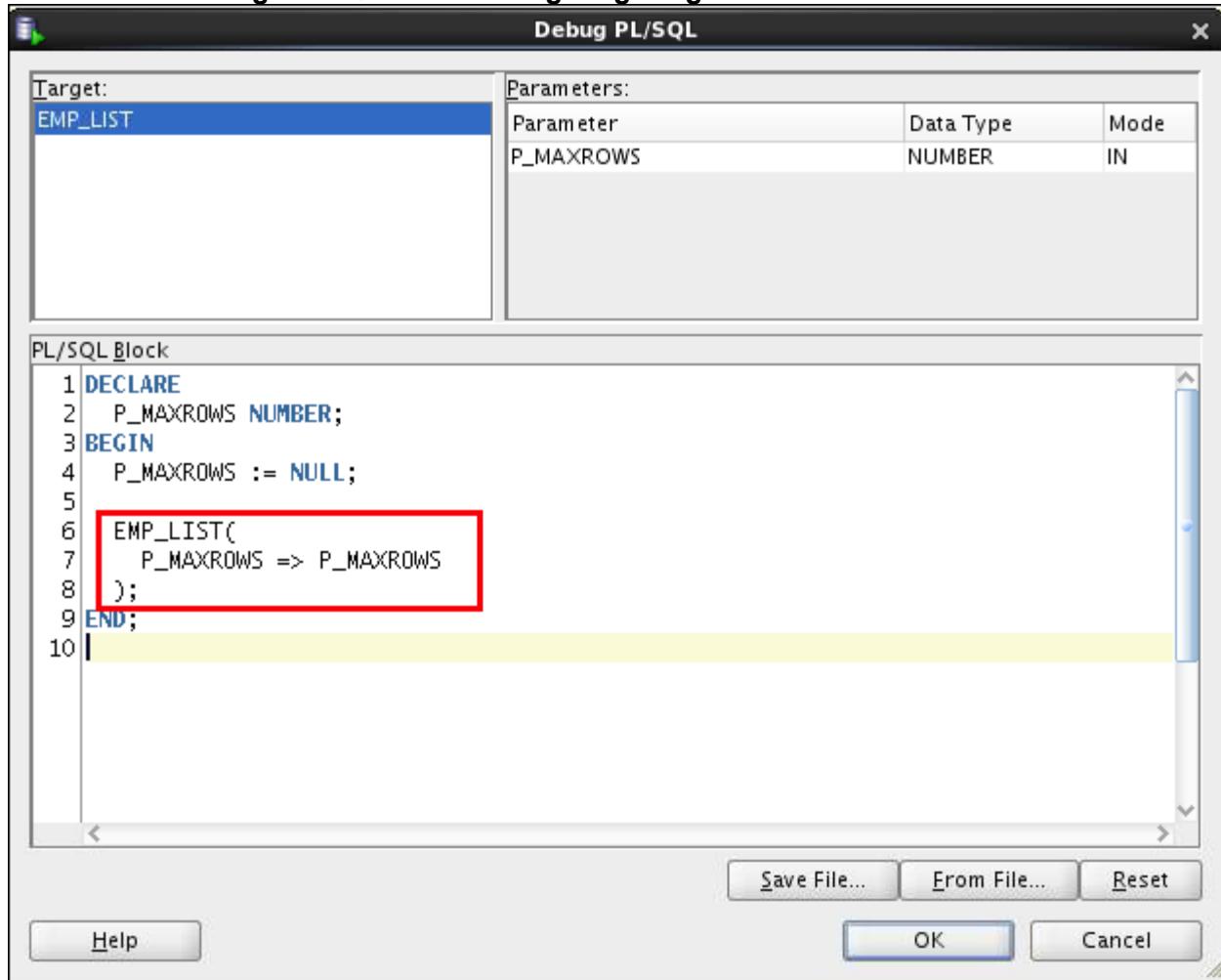


**Hinweis:** Warnungen werden erwartungsgemäß ausgegeben. Die beiden Warnungen erhalten Sie, weil der Parameter PLSQL\_DEBUG in Oracle Database 11g verworfen wurde, während SQL Developer diesen Parameter noch verwendet.

8. Debuggen Sie die Prozedur.

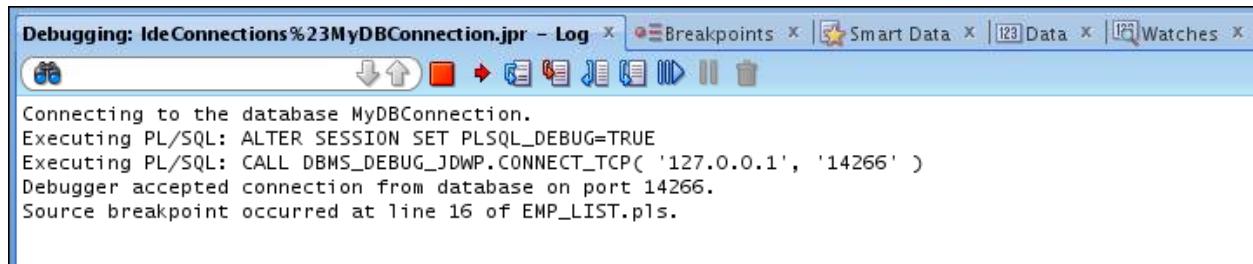
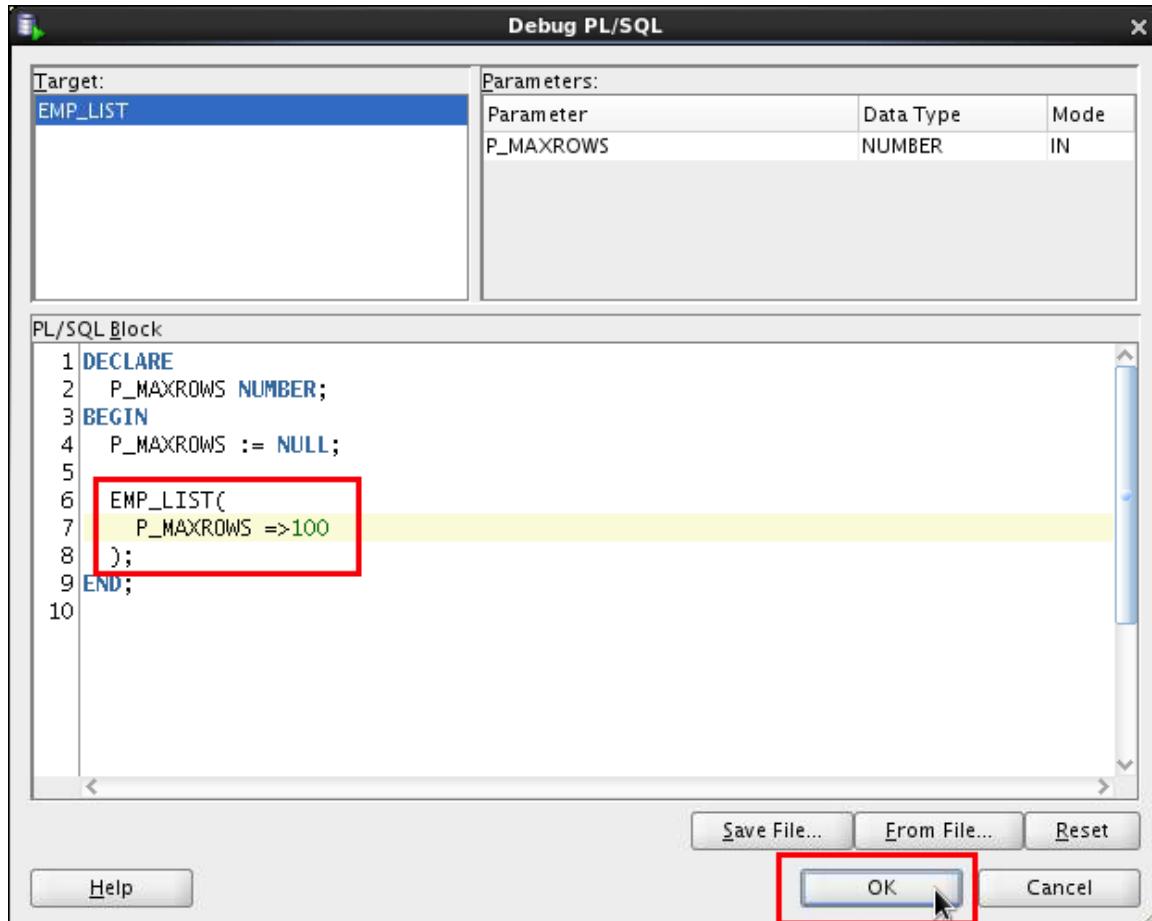
**Klicken Sie in der Symbolleiste der Prozedur wie unten abgebildet auf das Symbol "Debug":**



**Das Fenster Debug PL/SQL wird wie folgt angezeigt:**

9. Geben Sie 100 als Wert des Parameters P\_MAXROWS ein.

**Ersetzen Sie den zweiten Parameter P\_MAXROWS durch 100, und klicken Sie dann auf "OK". Die Programmsteuerung stoppt am ersten Breakpoint in der Prozedur (blau hervorgehoben), und der rote Pfeil weist auf diese Codezeile. Die zusätzlichen Debuggingregisterkarten werden am unteren Rand der Seite angezeigt.**



**Hinweis:** Wenn Sie bei der Ausführung dieses Schrittes den Fehler "access control list (ACL) error" erhalten, gehen Sie wie folgt vor:

- Öffnen Sie SQL\*Plus.
- Melden Sie sich als SYSDBA an.

- c. Führen Sie den folgenden Code aus:

```
BEGIN
  DBMS_NETWORK_ACL_ADMIN.APPEND_HOST_ACE(
    host => '127.0.0.1',
    ace => xs$ace_type(privilege_list => xs$name_list('jdwp'),
    principal_name => 'ora61',
    principal_type => xs_acl.ptype_db));
END;
/
```

**Der ACL-Fehler ist aufgelöst. Setzen Sie die Übung ab dem 8. Schritt der Lösung für Übung 2 zu Lektion 3 fort.**

10. Untersuchen Sie den Wert der Variablen in der Registerkarte **Data**. Welche Werte sind `REC_EMP` und `EMP_TAB` zugewiesen? Nennen Sie die Gründe.

**Beide sind auf NULL gesetzt, da die Daten noch nicht in den Cursor abgerufen wurden.**

The screenshot shows the Oracle Database Debugger interface with the 'Data' tab selected. The 'Data' tab has three tabs: Breakpoints, Smart Data, Data, and Watches. The Data tab displays a table of variables with their names, values, and types. The table has columns for Name, Value, and Type. The variable `REC_EMP` is expanded to show its components: `COMMISSION_PCT`, `DEPARTMENT_NAME`, `EMPLOYEE_ID`, `LAST_NAME`, and `SALARY`, all of which have a value of NULL. The variable `EMP_TAB` is also expanded to show its components: `I` (with value 1) and `V_CITY`, both of which have a value of NULL. The type for `REC_EMP` is listed as Rowtype.

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

11. Verwenden Sie die Debugoption **Step Into**, um in alle Codezeilen in `emp_list` hineinzugehen und die gesamte Schleife nur einmal zu durchlaufen.

**Drücken Sie F7, um nur einmal in den Code hineinzugehen.**

12. Untersuchen Sie den Wert der Variablen in der Registerkarte **Data**. Welche Werte sind REC\_EMP und EMP\_TAB zugewiesen?

**rec\_emp** wird beim Ausführen der Zeile `FETCH cur_emp INTO rec_emp;` wie folgt initialisiert:

Name	Value	Type
P_MAXROWS	100	NUMBER
REC_EMP		Rowtype
COMMISSION_PCT	NULL	NUMBER(2,2)
DEPARTMENT_NAME	'Administration'	VARCHAR2(30)
EMPLOYEE_ID	200	NUMBER(6,0)
LAST_NAME	'Whalen'	VARCHAR2(25)
SALARY	4400	NUMBER(8,2)
EMP_TAB	indexed table	EMP_TAB_TYPE
_values		EMP_TAB_TYPE element[0]
I	1	NUMBER
V_CITY	NULL	VARCHAR2(30)

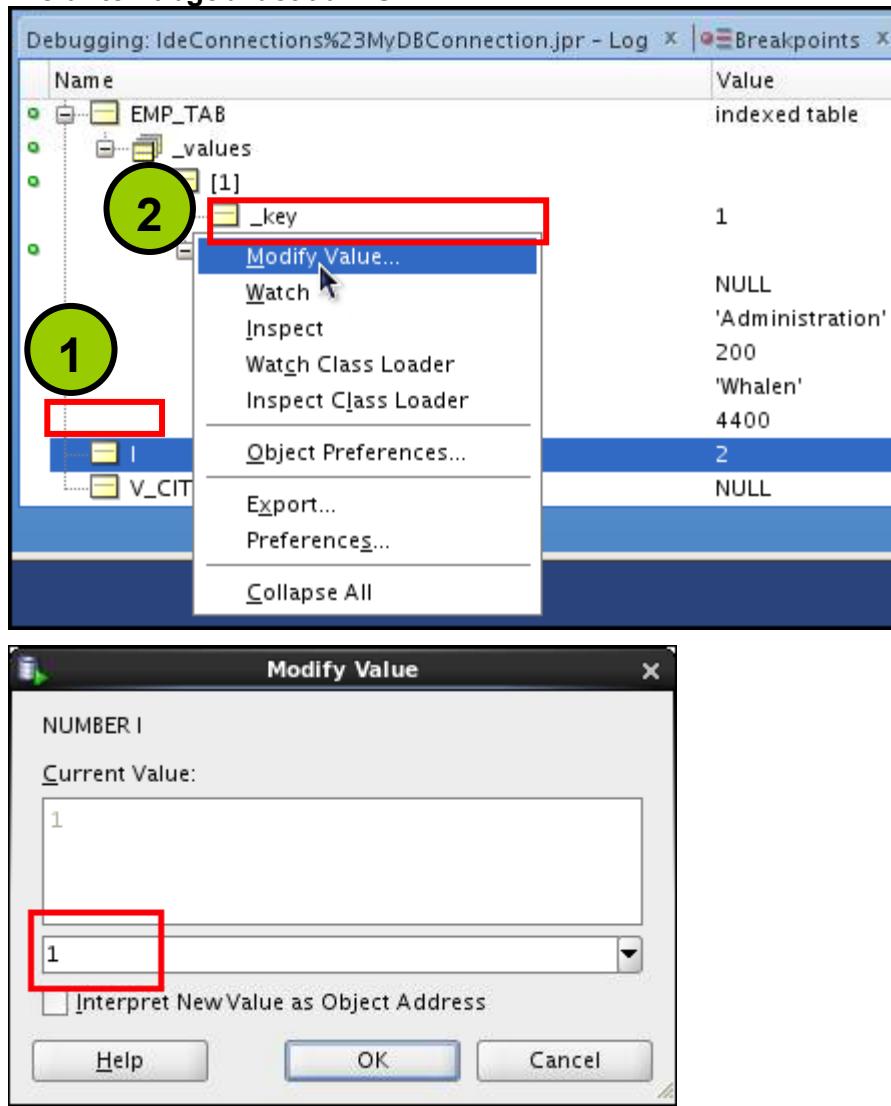
13. Drücken Sie weiterhin F7, bis die Zeile `emp_tab(i) := rec_emp;` ausgeführt wird. Untersuchen Sie den Wert der Variablen in der Registerkarte **Data**. Welche Werte sind EMP\_TAB zugewiesen?

Bei Ausführung der Zeile `emp_tab(i) := rec_emp;` wird emp\_tab wie unten abgebildet mit `rec_emp` initialisiert:

Name	Value	Type
P_MAXROWS	100	NUMBER
REC_EMP		Rowtype
COMMISSION_PCT	NULL	NUMBER(2,2)
DEPARTMENT_NAME	'Administration'	VARCHAR2(30)
EMPLOYEE_ID	200	NUMBER(6,0)
LAST_NAME	'Whalen'	VARCHAR2(25)
SALARY	4400	NUMBER(8,2)
EMP_TAB	indexed table	EMP_TAB_TYPE
I	1	NUMBER
V_CITY	NULL	VARCHAR2(30)

14. Ändern Sie den Wert des Zählers "i" mit der Registerkarte Data in 98.

Klicken Sie in der Registerkarte Data mit der rechten Maustaste auf "i", und wählen Sie dann im Kontextmenü die Option "Modify Value". Das Fenster "Modify Value" wird angezeigt. Ersetzen Sie den Wert 1 im Textfeld durch 98, und klicken Sie dann wie unten abgebildet auf "OK":





15. Drücken Sie weiterhin F7, bis Sie feststellen, dass die Liste der Mitarbeiter in der Registerkarte **Debugging – Log** angezeigt wird. Wie viele Mitarbeiter werden angezeigt?

**Die Ausgabe am Ende der Debuggingsession ist unten abgebildet, wobei drei Mitarbeiter angezeigt werden:**

```

Exception breakpoint occurred at line 29 of EMP_LIST.pls.
$0oracle.EXCEPTION_ORA_1403:
ORA-01403: no data found
ORA-06512: at "ORA61.EMP_LIST", line 28
ORA-06512: at line 6
Executing PL/SQL: CALL DBMS_DEBUG_JDWP.DISCONNECT()
Employee Hartstein works in Toronto
Employee Fay works in Toronto
Employee Raphaely works in Seattle
Employee Khoo works in Seattle
Khoo
Raphaely
Fay
Hartstein
Process exited.
Disconnecting from the database MyDBConnection.
Debugger disconnected from database.

Messages Debugging: IdeConnections%23MyDBConnection.jpr x

```

16. Wenn Sie den Code mit der Debugger-Option **Step Over** ablaufen lassen, läuft dann auch die Funktion `get_location` ab? Warum oder warum nicht?

**Obwohl die Codezeile an der Stelle, an der der dritte Breakpoint gesetzt wurde, einen Aufruf für die Funktion `get_location` enthält, führt Step Over (F8) die Codezeile aus und ruft den zurückgegebenen Wert der Funktion ab (wie mit F7). Die Steuerung wird jedoch nicht auf die Funktion `get_location` übertragen.**

# **Übungen zu Lektion 4 – Packages erstellen**

## **Kapitel 4**

# Übungen zu Lektion 4 – Überblick

---

## Lektionsüberblick

In diesen Übungen erstellen Sie eine Packagespezifikation und einen Packagebody namens `JOB_PKG`, der eine Kopie der Prozeduren `ADD_JOB`, `UPD_JOB` und `DEL_JOB` sowie der Funktion `GET_JOB` enthält. Außerdem erstellen Sie ein Package, das private und öffentliche Konstrukte enthält, und rufen es mit Beispieldaten auf.

### Hinweis:

1. Führen Sie das Skript  
`/home/oracle/plpu/code_ex/code_ex_scripts/clean_up_scripts/cleanup_04.sql` aus, bevor Sie mit dieser Übung beginnen.
2. Wenn Sie einen Schritt in einer Übung ausgelassen haben, führen Sie das entsprechende Lösungsskript zu diesem Übungsschritt aus, bevor Sie mit dem nächsten Schritt oder der nächsten Übung fortfahren.

# Übung 1 zu Lektion 4 – Packages erstellen und verwenden

---

## Überblick

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

**Hinweis:** Führen Sie das Skript `cleanup_04.sql` im Verzeichnis

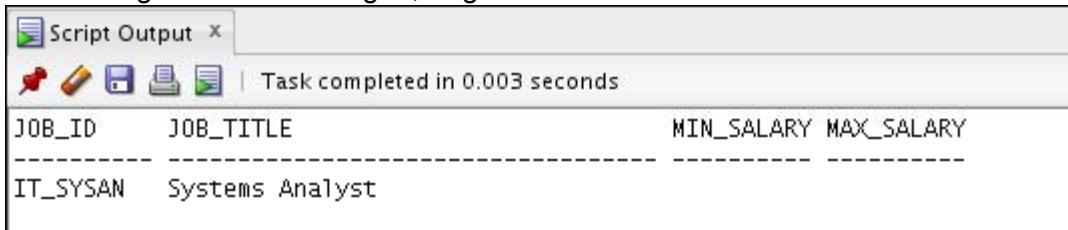
`/home/oracle/plpu/code_ex/code_ex_scripts/clean_up_scripts/` aus, bevor Sie die folgenden Aufgaben absolvieren.

## Aufgabe

1. Erstellen Sie eine Packagespezifikation und einen Packagebody namens `JOB_PKG`, der eine Kopie der Prozeduren `ADD_JOB`, `UPD_JOB` und `DEL_JOB` sowie der Funktion `GET_JOB` enthält.

**Hinweis:** Erstellen Sie das Package mithilfe des Codes aus Ihren zuletzt gespeicherten Prozeduren und Funktionen. Kopieren Sie hierfür den Code in einer Prozedur oder Funktion, und fügen Sie ihn dann im entsprechenden Bereich des Packages ein.

- a. Erstellen Sie die Packagespezifikation einschließlich der Procedur- und Funktionsüberschriften als öffentliche Konstrukte.
- b. Erstellen Sie den Packagebody mit den Implementierungen für die einzelnen Unterprogramme.
- c. Löschen Sie unterhalb der Knoten **Procedures** und **Functions** im Object Navigator-Baum die folgenden Standalone-Prozeduren und die folgende Standalone-Funktion, die Sie gerade in das Package integriert haben:
  - 1) Prozeduren `ADD_JOB`, `UPD_JOB` und `DEL_JOB`
  - 2) Funktion `GET_JOB`
- d. Rufen Sie die Packageprozedur `ADD_JOB` auf, indem Sie die Werte `IT_SYSAN` und `SYSTEMS ANALYST` als Parameter übergeben.
- e. Um die Ergebnisse anzuzeigen, fragen Sie die Tabelle `JOBS` ab.



The screenshot shows the 'Script Output' window from Oracle SQL Developer. It displays the results of a query on the JOBS table. The output window has a toolbar with icons for script, redo, undo, save, and refresh. Below the toolbar, it says 'Task completed in 0.003 seconds'. The table data is as follows:

JOB_ID	JOB_TITLE	MIN_SALARY	MAX_SALARY
<hr/>			
IT_SYSAN	Systems Analyst		

2. Erstellen Sie ein Package, das private und öffentliche Konstrukte enthält, und rufen Sie es auf.
  - a. Erstellen Sie eine Packagespezifikation und einen Packagebody namens `EMP_PKG`, der die folgenden bereits erstellten Prozeduren und Funktionen enthält:
    - 1) Prozedur `ADD_EMPLOYEE` als öffentliches Konstrukt
    - 2) Prozedur `GET_EMPLOYEE` als öffentliches Konstrukt
    - 3) Funktion `VALID_DEPTID` als privates Konstrukt

- b. Rufen Sie die Prozedur `EMP_PKG.ADD_EMPLOYEE` auf. Verwenden Sie die Abteilungsnummer 15 für die Mitarbeiterin Jane Harris mit der E-Mail-ID `JAHARRIS`. Da die Abteilungsnummer 15 nicht vorhanden ist, müssten Sie eine Fehlermeldung erhalten, wie im Exception Handler der Prozedur angegeben.
- c. Rufen Sie die Packageprozedur `ADD_EMPLOYEE` auf. Verwenden Sie die Abteilungsnummer 80 für den Mitarbeiter David Smith mit der E-Mail-ID `DASMITH`.
- d. Um zu verifizieren, dass der neue Mitarbeiter hinzugefügt wurde, fragen Sie die Tabelle `EMPLOYEES` ab.

## Übung 1 zu Lektion 4 – Lösung: Packages erstellen und verwenden

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

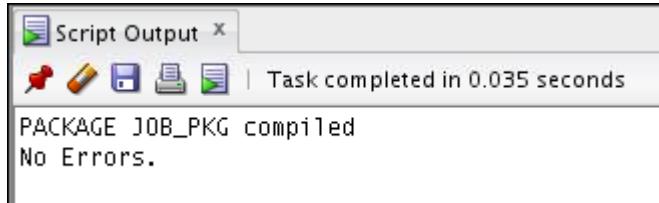
1. Erstellen Sie eine Packagespezifikation und einen Packagebody namens `JOB_PKG`, der eine Kopie der Prozeduren `ADD_JOB`, `UPD_JOB` und `DEL_JOB` sowie der Funktion `GET_JOB` enthält.

**Hinweis:** Erstellen Sie das Package mithilfe des Codes aus Ihren zuletzt gespeicherten Prozeduren und Funktionen. Kopieren Sie hierfür den Code in einer Prozedur oder Funktion, und fügen Sie ihn dann im entsprechenden Bereich des Packages ein.

- a. Erstellen Sie die Packagespezifikation einschließlich der Prozedur- und Funktionsüberschriften als öffentliche Konstrukte.

Öffnen Sie das Skript `/home/oracle/labs/plpu/solns/sol_04.sql`. Entfernen Sie die Kommentarzeichen des Codes unter Aufgabe 1\_a, und wählen Sie den Code. Um die Packagespezifikation zu erstellen und zu kompilieren, klicken Sie in der SQL Worksheet-Symbolleiste auf das Symbol "Run Script" (F5). Der Code und das Ergebnis werden wie folgt angezeigt:

```
CREATE OR REPLACE PACKAGE job_pkg IS
    PROCEDURE add_job (p_jobid jobs.job_id%TYPE, p_jobtitle
jobs.job_title%TYPE);
    PROCEDURE del_job (p_jobid jobs.job_id%TYPE);
    FUNCTION get_job (p_jobid IN jobs.job_id%type) RETURN
jobs.job_title%type;
    PROCEDURE upd_job(p_jobid IN jobs.job_id%TYPE, p_jobtitle IN
jobs.job_title%TYPE);
END job_pkg;
/
SHOW ERRORS
```



- b. Erstellen Sie den Packagebody mit den Implementierungen für die einzelnen Unterprogramme.

Entfernen Sie die Kommentarzeichen des Codes unter Aufgabe 1\_b, und wählen Sie den Code. Um den Packagebody zu erstellen und zu kompilieren, klicken Sie in der SQL Worksheet-Symbolleiste auf das Symbol "Run Script" (F5). Der Code und das Ergebnis werden wie folgt angezeigt:

```
CREATE OR REPLACE PACKAGE BODY job_pkg IS
    PROCEDURE add_job (
        p_jobid jobs.job_id%TYPE,
        p_jobtitle jobs.job_title%TYPE) IS
```

```

BEGIN
    INSERT INTO jobs (job_id, job_title)
    VALUES (p_jobid, p_jobtitle);
    COMMIT;
END add_job;

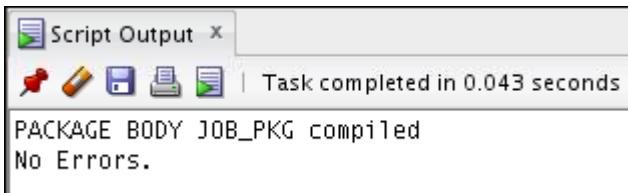
PROCEDURE del_job (p_jobid jobs.job_id%TYPE) IS
BEGIN
    DELETE FROM jobs
    WHERE job_id = p_jobid;
    IF SQL%NOTFOUND THEN
        RAISE_APPLICATION_ERROR(-20203, 'No jobs deleted.');
    END IF;
END DEL_JOB;

FUNCTION get_job (p_jobid IN jobs.job_id%type)
    RETURN jobs.job_title%type IS
    v_title jobs.job_title%type;
BEGIN
    SELECT job_title
    INTO v_title
    FROM jobs
    WHERE job_id = p_jobid;
    RETURN v_title;
END get_job;

PROCEDURE upd_job(
    p_jobid IN jobs.job_id%TYPE,
    p_jobtitle IN jobs.job_title%TYPE) IS
BEGIN
    UPDATE jobs
    SET job_title = p_jobtitle
    WHERE job_id = p_jobid;
    IF SQL%NOTFOUND THEN
        RAISE_APPLICATION_ERROR(-20202, 'No job updated.');
    END IF;
END upd_job;

END job_pkg;
/
SHOW ERRORS

```



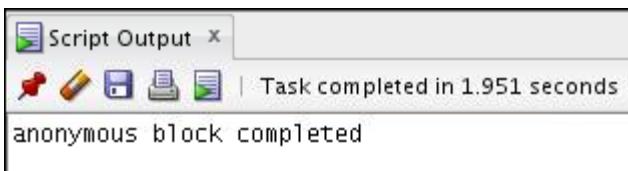
- c. Löschen Sie unterhalb der Knoten **Procedures** und **Functions** im Object Navigator-Baum die folgenden Standalone-Prozeduren und die folgende Standalone-Funktion, die Sie gerade in das Package integriert haben:
- 1) Prozeduren ADD\_JOB, UPD\_JOB und DEL\_JOB
  - 2) Funktion GET\_JOB

**Um eine Prozedur oder Funktion zu löschen, klicken Sie im Object Navigator-Baum mit der rechten Maustaste auf den Namen der Prozedur oder Funktion und wählen dann im Popup-Menü die Option "Drop". Das Fenster "Drop" wird geöffnet. Um die Prozedur oder Funktion zu löschen, klicken Sie auf "Apply". Klicken Sie im Bestätigungsdialogfeld auf "OK".**

- d. Rufen Sie die Packageprozedur ADD\_JOB auf, indem Sie die Werte IT\_SYSAN und SYSTEMS ANALYST als Parameter übergeben.

**Entfernen Sie die Kommentarzeichen des Codes unter Aufgabe 1\_d, und wählen Sie den Code. Um die Packageprozedur aufzurufen, klicken Sie in der SQL Worksheet-Symbolleiste auf das Symbol "Run Script" (F5). Der Code und das Ergebnis werden wie folgt angezeigt:**

```
EXECUTE job_pkg.add_job('IT_SYSAN', 'Systems Analyst')
```



- e. Um die Ergebnisse anzuzeigen, fragen Sie die Tabelle JOBS ab.

**Entfernen Sie die Kommentarzeichen des Codes unter Aufgabe 1\_e, und wählen Sie den Code. Um die Tabelle JOBS abzufragen, klicken Sie in der SQL Worksheet-Symbolleiste auf das Symbol "Run Script" (F5) oder "Execute Statement" (F9). Der Code und das Ergebnis (über das Symbol "Run Script") werden wie folgt angezeigt:**

```
SELECT *
FROM jobs
WHERE job_id = 'IT_SYSAN';
```

Script Output X			
Task completed in 0.003 seconds			
JOB_ID	JOB_TITLE	MIN_SALARY	MAX_SALARY
IT_SYSAN	Systems Analyst		

2. Erstellen Sie ein Package, das private und öffentliche Konstrukte enthält, und rufen Sie es auf.
- Erstellen Sie eine Packagespezifikation und einen Packagebody namens `EMP_PKG`, der die folgenden bereits erstellten Prozeduren und Funktionen enthält:
    - Prozedur `ADD_EMPLOYEE` als öffentliches Konstrukt
    - Prozedur `GET_EMPLOYEE` als öffentliches Konstrukt
    - Funktion `VALID_DEPTID` als privates Konstrukt

**Entfernen Sie die Kommentarzeichen des Codes unter Aufgabe 2\_a, und wählen Sie den Code. Um die Packageprozedur aufzurufen, klicken Sie in der SQL Worksheet-Symbolleiste auf das Symbol "Run Script" (F5). Der Code und das Ergebnis werden wie folgt angezeigt:**

```
CREATE OR REPLACE PACKAGE emp_pkg IS
  PROCEDURE add_employee(
    p_first_name employees.first_name%TYPE,
    p_last_name employees.last_name%TYPE,
    p_email employees.email%TYPE,
    p_job employees.job_id%TYPE DEFAULT 'SA_REP',
    p_mgr employees.manager_id%TYPE DEFAULT 145,
    p_sal employees.salary%TYPE DEFAULT 1000,
    p_commission employees.commission_pct%TYPE DEFAULT 0,
    p_deptid employees.department_id%TYPE DEFAULT 30);
  PROCEDURE get_employee(
    p.empid IN employees.employee_id%TYPE,
    p_sal OUT employees.salary%TYPE,
    p_job OUT employees.job_id%TYPE);
END emp_pkg;
```

```
/SHOW ERRORS

CREATE OR REPLACE PACKAGE BODY emp_pkg IS
  FUNCTION valid_deptid(p_deptid IN
    departments.department_id%TYPE) RETURN BOOLEAN IS
    v_dummy PLS_INTEGER;
  BEGIN
    SELECT 1
    INTO v_dummy
    FROM departments
    WHERE department_id = p_deptid;
    RETURN TRUE;
  EXCEPTION
    WHEN NO_DATA_FOUND THEN
      RETURN FALSE;
```

```

END valid_deptid;

PROCEDURE add_employee(
    p_first_name employees.first_name%TYPE,
    p_last_name employees.last_name%TYPE,
    p_email employees.email%TYPE,
    p_job employees.job_id%TYPE DEFAULT 'SA_REP',
    p_mgr employees.manager_id%TYPE DEFAULT 145,
    p_sal employees.salary%TYPE DEFAULT 1000,
    p_comm employees.commission_pct%TYPE DEFAULT 0,
    p_deptid employees.department_id%TYPE DEFAULT 30) IS
BEGIN
    IF valid_deptid(p_deptid) THEN
        INSERT INTO employees(employee_id, first_name, last_name,
email,
                           job_id, manager_id, hire_date, salary, commission_pct,
department_id)
        VALUES (employees_seq.NEXTVAL, p_first_name, p_last_name,
p_email,
               p_job, p_mgr, TRUNC(SYSDATE), p_sal, p_comm, p_deptid);
    ELSE
        RAISE_APPLICATION_ERROR (-20204, 'Invalid department ID.
Try again.');
    END IF;
END add_employee;

PROCEDURE get_employee(
    p.empid IN employees.employee_id%TYPE,
    p_sal OUT employees.salary%TYPE,
    p_job OUT employees.job_id%TYPE) IS
BEGIN
    SELECT salary, job_id
    INTO p_sal, p_job
    FROM employees
    WHERE employee_id = p.empid;
END get_employee;
END emp_pkg;
/
SHOW ERRORS

```

```
Script Output | Task completed in 0.07 seconds
PACKAGE EMP_PKG compiled
No Errors.
PACKAGE BODY EMP_PKG compiled
No Errors.
```

- b. Rufen Sie die Prozedur `EMP_PKG.ADD_EMPLOYEE` auf. Verwenden Sie die Abteilungsnummer 15 für die Mitarbeiterin Jane Harris mit der E-Mail-ID JAHARRIS. Da die Abteilungsnummer 15 nicht vorhanden ist, müssten Sie eine Fehlermeldung erhalten, wie im Exception Handler der Prozedur angegeben.

**Entfernen Sie die Kommentarzeichen des Codes unter Aufgabe 2\_b, und wählen Sie den Code. Um die Packageprozedur aufzurufen, klicken Sie in der SQL Worksheet-Symbolleiste auf das Symbol "Run Script" (F5). Der Code und das Ergebnis werden wie folgt angezeigt:**

**Hinweis:** Sie müssen erst Schritt 3-2-a abschließen, bevor Sie diesen Schritt ausführen. Wenn Sie Schritt 3-2-a nicht abgeschlossen haben, führen Sie zunächst den Code unter Aufgabe 2\_a aus.

```
EXECUTE emp_pkg.add_employee('Jane', 'Harris', 'JAHARRIS',
p_deptid => 15)
```

```
Script Output | Task completed in 1.03 seconds
Error starting at line 165 in command:
EXECUTE emp_pkg.add_employee('Jane', 'Harris', 'JAHARRIS', p_deptid => 15)
Error report:
ORA-20204: Invalid department ID. Try again.
ORA-06512: at "ORA61.EMP_PKG", line 32
ORA-06512: at line 1
```

- c. Rufen Sie die Packageprozedur `ADD_EMPLOYEE` auf. Verwenden Sie die Abteilungsnummer 80 für den Mitarbeiter David Smith mit der E-Mail-ID DASMITH.

**Entfernen Sie die Kommentarzeichen des Codes unter Aufgabe 2\_c, und wählen Sie den Code. Um die Packageprozedur aufzurufen, klicken Sie in der SQL Worksheet-Symbolleiste auf das Symbol "Run Script" (F5). Der Code und das Ergebnis werden wie folgt angezeigt:**

```
EXECUTE emp_pkg.add_employee('David', 'Smith', 'DASMITH',
p_deptid => 80)
```

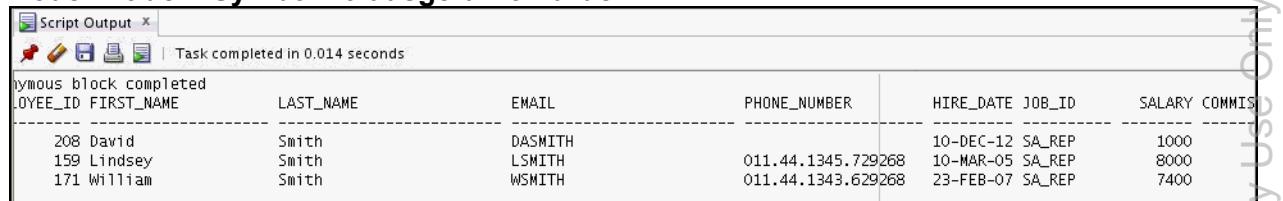
```
Script Output | Task completed in 0.125 seconds
anonymous block completed
```

- d. Um zu verifizieren, dass der neue Mitarbeiter hinzugefügt wurde, fragen Sie die Tabelle EMPLOYEES ab.

**Entfernen Sie die Kommentarzeichen des Codes unter Aufgabe 2\_d, und wählen Sie den Code. Um die Tabelle EMPLOYEES abzurufen, klicken Sie in der SQL Worksheet-Symbolleiste auf das Symbol "Run Script" (F5) oder "Execute Statement" (F9). (Achten Sie dabei darauf, dass der Cursor in einer Zeile der SELECT-Anweisung steht.) Der Code und das Ergebnis (Symbol "Execute Statement") werden wie folgt angezeigt:**

```
SELECT *
FROM employees
WHERE last_name = 'Smith';
```

**Die folgende Ausgabe wird in der Registerkarte "Results" angezeigt, weil der Code mit dem Symbol F9 ausgeführt wurde.**



The screenshot shows the 'Script Output' window of the Oracle SQL Worksheet. It displays the results of a SQL query that selects all columns for employees whose last name is 'Smith'. The output includes the employee ID, first name, last name, email, phone number, hire date, job ID, salary, and commission percentage. The results show three rows: David Smith, Lindsey Smith, and William Smith.

EMPLOYEE_ID	FIRST_NAME	LAST_NAME	EMAIL	PHONE_NUMBER	HIRE_DATE	JOB_ID	SALARY	COMMISSION_PCT
208	David	Smith	DASMITH		10-DEC-12	SA_REP	1000	
159	Lindsey	Smith	LSMITH	011.44.1345.729268	10-MAR-05	SA_REP	8000	
171	William	Smith	WSMITH	011.44.1343.629268	23-FEB-07	SA_REP	7400	



# **Übungen zu Lektion 5 – Mit Packages arbeiten**

## **Kapitel 5**

# Übungen zu Lektion 5 – Überblick

---

## Lektionsüberblick

In diesen Übungen fügen Sie überladene Unterprogramme in ein vorhandenes Package ein und verwenden Vorwärtsdeklarationen. Außerdem erstellen Sie in einem Packagebody einen Packageinitialisierungsblock zum Füllen einer PL/SQL-Tabelle.

### Hinweis:

1. Führen Sie das Skript  
`/home/oracle/plpu/code_ex/code_ex_scripts/clean_up_scripts/  
cleanup_05.sql` aus, bevor Sie mit dieser Übung beginnen.
2. Wenn Sie einen Schritt in einer Übung ausgelassen haben, führen Sie das entsprechende Lösungsskript zu diesem Übungsschritt aus, bevor Sie mit dem nächsten Schritt oder der nächsten Übung fortfahren.

# Übung 1 zu Lektion 5 – Mit Packages arbeiten

---

## Überblick

In dieser Übung ändern Sie den Code für das Package `EMP_PKG`, das Sie zuvor erstellt haben, und überladen dann die Prozedur `ADD_EMPLOYEE`. Als Nächstes erstellen Sie zwei überladene Funktionen namens `GET_EMPLOYEE` im Package `EMP_PKG`. Außerdem fügen Sie `EMP_PKG` eine öffentliche Prozedur hinzu, um eine private PL/SQL-Tabelle gültiger Abteilungsnummern zu füllen, ändern die Funktion `VALID_DEPTID`, um den Inhalt der privaten PL/SQL-Tabelle zum Validieren der Werte der Abteilungsnummer zu verwenden. Außerdem ändern Sie die Validierungsverarbeitungsfunktion `VALID_DEPTID` so, dass die private PL/SQL-Tabelle der Abteilungsnummern verwendet wird. Schließlich reorganisieren Sie die Unterprogramme in der Packagespezifikation und im Body so, dass sie in alphabetischer Reihenfolge angezeigt werden.

**Hinweis:** Führen Sie das Skript `cleanup_05.sql` im Verzeichnis `/home/oracle/plpu/code_ex/code_ex_scripts/clean_up_scripts/` aus, bevor Sie die folgenden Aufgaben absolvieren.

## Aufgabe

1. Ändern Sie den Code des Packages `EMP_PKG`, das Sie im 2. Schritt von Übung 4 erstellt haben, und überladen Sie die Prozedur `ADD_EMPLOYEE`.
  - a. Fügen Sie in der Packagespezifikation eine neue Prozedur namens `ADD_EMPLOYEE` hinzu, die die folgenden drei Parameter annimmt:
    - 1) Vorname
    - 2) Nachname
    - 3) Abteilungsnummer
  - b. Um das Package zu erstellen und zu kompilieren, klicken Sie auf das Symbol **Run Script** (F5).
  - c. Implementieren Sie die neue Prozedur `ADD_EMPLOYEE` im Packagebody wie folgt:
    - 1) Formatieren Sie die E-Mail-Adresse in Großbuchstaben. Verketten Sie dafür den ersten Buchstaben des Vornamens mit den ersten sieben Buchstaben des Nachnamens.
    - 2) Um den eigentlichen `INSERT`-Vorgang durchzuführen, soll die Prozedur die vorhandene Prozedur `ADD_EMPLOYEE` aufrufen. Sie verwendet dabei ihre Parameter und die formatierte E-Mail-Adresse, um die Werte bereitzustellen.
    - 3) Um das Package zu erstellen, klicken Sie auf **Run Script**. Kompilieren Sie das Package.
  - d. Um den Namen `Samuel Joplin` zur Abteilung 30 hinzuzufügen, rufen Sie die neue Prozedur `ADD_EMPLOYEE` auf.
  - e. Vergewissern Sie sich, dass der neue Mitarbeiter in die Tabelle `EMPLOYEES` eingefügt wurde.

2. Erstellen Sie im Package `EMP_PKG` zwei überladene Funktionen namens `GET_EMPLOYEE`:
  - a. Fügen Sie in der Packagespezifikation die folgenden Funktionen hinzu:
    - 1) Die Funktion `GET_EMPLOYEE`, die den Parameter `p_emp_id` auf Basis des Typs `employees.employee_id%TYPE` annimmt. Diese Funktion muss `EMPLOYEES%ROWTYPE` zurückgeben.
    - 2) Die Funktion `GET_EMPLOYEE`, die den Parameter `p_family_name` mit dem Typ `employees.last_name%TYPE` annimmt. Diese Funktion muss `EMPLOYEES%ROWTYPE` zurückgeben.
  - b. Um das Package neu zu erstellen und zu kompilieren, klicken Sie auf **Run Script**.
  - c. Gehen Sie im Packagebody wie folgt vor:
    - 1) Um einen Mitarbeiter mithilfe seiner Personalnummer abzufragen, implementieren Sie die erste Funktion `GET_EMPLOYEE`.
    - 2) Um den Gleich-Operator für den Wert zu verwenden, der im Parameter `p_family_name` bereitgestellt wird, implementieren Sie die zweite Funktion `GET_EMPLOYEE`.
  - d. Um das Package neu zu erstellen und zu kompilieren, klicken Sie auf **Run Script**.
  - e. Fügen Sie dem Package `EMP_PKG` wie folgt die Utilityprozedur `PRINT_EMPLOYEE` hinzu:
    - 1) Die Prozedur nimmt `EMPLOYEES%ROWTYPE` als Parameter an.
    - 2) Die Prozedur zeigt mithilfe des Packages `DBMS_OUTPUT` die folgenden Elemente für einen Mitarbeiter in einer Zeile an:
      - `department_id`
      - `employee_id`
      - `first_name`
      - `last_name`
      - `job_id`
      - `salary`
  - f. Um das Package zu erstellen und zu kompilieren, klicken Sie auf **Run Script (F5)**.
  - g. Rufen Sie mit einem anonymen Block die Funktion `EMP_PKG.GET_EMPLOYEE` mit der Personalnummer 100 und dem Nachnamen 'Joplin' auf. Zeigen Sie die Ergebnisse der zurückgegebenen Zeilen mithilfe der Prozedur `PRINT_EMPLOYEE` an.
  3. Da das Unternehmen die Abteilungsdaten nicht oft ändert, können Sie die Performance von `EMP_PKG` steigern, indem Sie eine öffentliche Prozedur namens `INIT_DEPARTMENTS` hinzufügen, um eine private PL/SQL-Tabelle mit gültigen Abteilungsnummern zu füllen. Ändern Sie die Funktion `VALID_DEPTID` so, dass die Werte der Abteilungsnummer mit dem Inhalt der privaten PL/SQL-Tabelle validiert werden.

**Hinweis:** Der Code unter der 3. Aufgabe enthält die Lösung für die Schritte a, b und c.

  - a. Erstellen Sie in der Packagespezifikation eine Prozedur namens `INIT_DEPARTMENTS` ohne Parameter. Fügen Sie im Packagespezifikationsbereich dazu folgenden Code vor der Spezifikation `PRINT_EMPLOYEES` hinzu:

```
PROCEDURE init_departments;
```

  - b. Um alle Abteilungsnummern in einer privaten PL/SQL-INDEX BY-Tabelle namens `valid_departments` zu speichern, die `BOOLEAN`-Werte enthält, implementieren Sie im Packagebody die Prozedur `INIT_DEPARTMENTS`.

- 1) Deklarieren Sie die Variable `valid_departments` und ihre Typdefinition `boolean_tab_type` vor allen Prozeduren des Bodys. Geben Sie am Anfang des Packagebodys folgenden Code ein:

```
TYPE boolean_tab_type IS TABLE OF BOOLEAN
INDEX BY BINARY_INTEGER;
valid_departments boolean_tab_type;
```

- 2) Verwenden Sie den Wert der Spalte `department_id` als Index für die Erstellung des Eintrags in der INDEX BY-Tabelle, um dessen Vorhandensein anzuzeigen, und weisen Sie dem Eintrag den Wert `TRUE` zu. Geben Sie die Prozedurdeklaration `INIT_DEPARTMENTS` am Ende des Packagebodys (unmittelbar nach der Prozedur `print_employees`) wie folgt ein:

```
PROCEDURE init_departments IS
BEGIN
  FOR rec IN (SELECT department_id FROM departments)
  LOOP
    valid_departments(rec.department_id) := TRUE;
  END LOOP;
END;
```

- c. Um die Prozedur `INIT_DEPARTMENTS` zur Initialisierung der Tabelle aufzurufen, erstellen Sie im Body den folgenden Initialisierungsblock:

```
BEGIN
  init_departments;
END;
```

- d. Um das Package zu erstellen und zu kompilieren, klicken Sie auf **Run Script** (F5).

4. Ändern Sie die Validierungsverarbeitungsfunktion `VALID_DEPTID` so, dass die private INDEX BY-Tabelle der Abteilungsnummern verwendet wird.

- Ändern Sie die Funktion `VALID_DEPTID` so, dass die Validierung mithilfe der Werte in der PL/SQL-Tabelle der Abteilungsnummern erfolgt. Um das Package zu erstellen, klicken Sie auf **Run Script** (F5). Kompilieren Sie das Package.
- Testen Sie den Code, indem Sie `ADD_EMPLOYEE` mit dem Namen `James Bond` in Abteilung 15 aufrufen. Was geschieht?
- Fügen Sie eine neue Abteilung ein. Geben Sie die Abteilungsnummer 15 und den Abteilungsnamen 'Security' ein. Schreiben Sie die Änderungen fest, und verifizieren Sie sie.
- Testen Sie den Code noch erneut, indem Sie `ADD_EMPLOYEE` mit dem Namen `James Bond` in Abteilung 15 aufrufen. Was geschieht?
- Um die interne PL/SQL-Tabelle mit den neuesten Abteilungsdaten zu aktualisieren, führen Sie die Prozedur `EMP_PKG.INIT_DEPARTMENTS` aus.
- Testen Sie den Code, indem Sie `ADD_EMPLOYEE` mit dem Namen `James Bond` aufrufen, der in Abteilung 15 arbeitet. Was geschieht?
- Löschen Sie den Mitarbeiter `James Bond` und die Abteilung 15 aus den entsprechenden Tabellen, schreiben Sie die Änderungen fest, und aktualisieren Sie die Abteilungsdaten, indem Sie die Prozedur `EMP_PKG.INIT_DEPARTMENTS` aufrufen. Zuvor müssen Sie jedoch `SET SERVEROUTPUT ON` eingeben.

5. Reorganisieren Sie die Unterprogramme in der Packagespezifikation und im Body so, dass sie in alphabetischer Reihenfolge angezeigt werden.
  - Bearbeiten Sie die Packagespezifikation so, dass die Unterprogramme in alphabetischer Reihenfolge neu angeordnet werden. Um die Packagespezifikation neu zu erstellen, klicken Sie auf **Run Script**. Kompilieren Sie die Packagespezifikation. Was geschieht?
  - Bearbeiten Sie den Packagebody so, dass alle Unterprogramme in alphabetischer Reihenfolge neu angeordnet werden. Um die Packagespezifikation neu zu erstellen, klicken Sie auf **Run Script**. Rekompilieren Sie die Packagespezifikation. Was geschieht?
  - Beheben Sie den Kompilierungsfehler mithilfe einer Vorwärtsdeklaration im Body für die entsprechende Unterprogrammreferenz. Klicken Sie auf **Run Script**, um das Package neu zu erstellen, und rekompilieren Sie dann das Package. Was geschieht?

## Übung 1 zu Lektion 5 – Lösung: Mit Packages arbeiten

In dieser Übung ändern Sie den Code für das Package `EMP_PKG`, das Sie zuvor erstellt haben, und überladen dann die Prozedur `ADD_EMPLOYEE`. Als Nächstes erstellen Sie zwei überladene Funktionen namens `GET_EMPLOYEE` im Package `EMP_PKG`. Außerdem fügen Sie `EMP_PKG` eine öffentliche Prozedur hinzu, um eine private PL/SQL-Tabelle gültiger Abteilungsnummern zu füllen, ändern die Funktion `VALID_DEPTID`, um den Inhalt der privaten PL/SQL-Tabelle zum Validieren der Werte der Abteilungsnummer zu verwenden. Außerdem ändern Sie die Validierungsverarbeitungsfunktion `VALID_DEPTID` so, dass die private PL/SQL-Tabelle der Abteilungsnummern verwendet wird. Schließlich reorganisieren Sie die Unterprogramme in der Packagespezifikation und im Body so, dass sie in alphabetischer Reihenfolge angezeigt werden.

- Ändern Sie den Code des Packages `EMP_PKG`, das Sie im 2. Schritt von Übung 4 erstellt haben, und überladen Sie die Prozedur `ADD_EMPLOYEE`.

- Fügen Sie in der Packagespezifikation eine neue Prozedur namens `ADD_EMPLOYEE` hinzu, die die folgenden drei Parameter annimmt:
  - 1) Vorname
  - 2) Nachname
  - 3) Abteilungsnummer

Öffnen Sie die Datei `/home/oracle/labs/plpu/solns/sol_05.sql`. Entfernen Sie die Kommentarzeichen des Codes unter Aufgabe 1\_a, und wählen Sie den Code. Der Code wird folgendermaßen angezeigt:

```
CREATE OR REPLACE PACKAGE emp_pkg IS
  PROCEDURE add_employee(
    p_first_name employees.first_name%TYPE,
    p_last_name employees.last_name%TYPE,
    p_email employees.email%TYPE,
    p_job employees.job_id%TYPE DEFAULT 'SA_REP',
    p_mgr employees.manager_id%TYPE DEFAULT 145,
    p_sal employees.salary%TYPE DEFAULT 1000,
    p_comm employees.commission_pct%TYPE DEFAULT 0,
    p_deptid employees.department_id%TYPE DEFAULT 30);

```

-- New overloaded `add_employee`

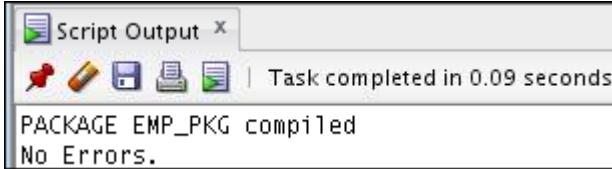
```
PROCEDURE add_employee(
  p_first_name employees.first_name%TYPE,
  p_last_name employees.last_name%TYPE,
  p_deptid employees.department_id%TYPE);
```

```

PROCEDURE get_employee(
    p.empid IN employees.employee_id%TYPE,
    p_sal OUT employees.salary%TYPE,
    p_job OUT employees.job_id%TYPE);
END emp_pkg;
/
SHOW ERRORS

```

- b. Um das Package zu erstellen und zu komplizieren, klicken Sie in der SQL Worksheet-Symbolleiste auf das Symbol "Run Script (F5)".



- c. Implementieren Sie die neue Prozedur ADD\_EMPLOYEE im Packagebody wie folgt:
- 1) Formatieren Sie die E-Mail-Adresse in Großbuchstaben. Verketten Sie dafür den ersten Buchstaben des Vornamens mit den ersten sieben Buchstaben des Nachnamens.
  - 2) Um den eigentlichen INSERT-Vorgang durchzuführen, soll die Prozedur die vorhandene Prozedur ADD\_EMPLOYEE aufrufen. Sie verwendet dabei ihre Parameter und die formatierte E-Mail-Adresse, um die Werte bereitzustellen.
  - 3) Um das Package zu erstellen, klicken Sie auf Run Script. Kompilieren Sie das Package.

**Entfernen Sie die Kommentarzeichen des Codes unter Aufgabe 1\_c, und wählen Sie den Code. Um die Packageprozedur aufzurufen, klicken Sie in der SQL Worksheet-Symbolleiste auf das Symbol "Run Script" (F5). Der Code und das Ergebnis werden wie folgt angezeigt (der neu hinzugefügte Code ist im untenstehenden Codefeld fett hervorgehoben):**

```

CREATE OR REPLACE PACKAGE emp_pkg IS
PROCEDURE add_employee(
    p_first_name employees.first_name%TYPE,
    p_last_name employees.last_name%TYPE,
    p_email employees.email%TYPE,
    p_job employees.job_id%TYPE DEFAULT 'SA_REP',
    p_mgr employees.manager_id%TYPE DEFAULT 145,
    p_sal employees.salary%TYPE DEFAULT 1000,
    p_commission employees.commission_pct%TYPE DEFAULT 0,
    p_deptid employees.department_id%TYPE DEFAULT 30);

-- New overloaded add_employee

```

```

PROCEDURE add_employee(
    p_first_name employees.first_name%TYPE,
    p_last_name employees.last_name%TYPE,
    p_deptid employees.department_id%TYPE);

-- End of the spec of the new overloaded add_employee

PROCEDURE get_employee(
    p.empid IN employees.employee_id%TYPE,
    p_sal OUT employees.salary%TYPE,
    p_job OUT employees.job_id%TYPE);
END emp_pkg;
/
SHOW ERRORS
CREATE OR REPLACE PACKAGE BODY emp_pkg IS
    FUNCTION valid_deptid(p_deptid IN
departments.department_id%TYPE) RETURN BOOLEAN IS
        v_dummy PLS_INTEGER;
    BEGIN
        SELECT 1
        INTO v_dummy
        FROM departments
        WHERE department_id = p_deptid;
        RETURN TRUE;
    EXCEPTION
        WHEN NO_DATA_FOUND THEN
            RETURN FALSE;
    END valid_deptid;

PROCEDURE add_employee(
    p_first_name employees.first_name%TYPE,
    p_last_name employees.last_name%TYPE,
    p_email employees.email%TYPE,
    p_job employees.job_id%TYPE DEFAULT 'SA_REP',
    p_mgr employees.manager_id%TYPE DEFAULT 145,
    p_sal employees.salary%TYPE DEFAULT 1000,
    p_comm employees.commission_pct%TYPE DEFAULT 0,
    p_deptid employees.department_id%TYPE DEFAULT 30) IS

```

```

BEGIN
  IF valid_deptid(p_deptid) THEN
    INSERT INTO employees(employee_id, first_name, last_name,
                           email, job_id, manager_id, hire_date, salary,
                           commission_pct, department_id)
    VALUES (employees_seq.NEXTVAL, p_first_name, p_last_name,
            p_email, p_job, p_mgr, TRUNC(SYSDATE), p_sal, p_comm,
            p_deptid);
  ELSE
    RAISE_APPLICATION_ERROR (-20204, 'Invalid department ID. Try
                                again.');
  END IF;
END add_employee;

-- New overloaded add_employee procedure

PROCEDURE add_employee(
  p_first_name employees.first_name%TYPE,
  p_last_name employees.last_name%TYPE,
  p_deptid employees.department_id%TYPE) IS
  p_email employees.email%type;
BEGIN
  p_email := UPPER(SUBSTR(p_first_name, 1,
                           1) || SUBSTR(p_last_name, 1, 7));
  add_employee(p_first_name, p_last_name, p_email, p_deptid =>
                p_deptid);
END;

-- End declaration of the overloaded add_employee procedure

PROCEDURE get_employee(
  p.empid IN employees.employee_id%TYPE,
  p_sal OUT employees.salary%TYPE,
  p_job OUT employees.job_id%TYPE) IS
BEGIN
  SELECT salary, job_id
  INTO p_sal, p_job
  FROM employees
  WHERE employee_id = p.empid;
END get_employee;
END emp_pkg;
/
SHOW ERRORS

```

```

Script Output X
PACKAGE EMP_PKG compiled
No Errors.
PACKAGE BODY EMP_PKG compiled
No Errors.

```

- d. Um den Namen Samuel Joplin zur Abteilung 30 hinzuzufügen, rufen Sie die neue Prozedur ADD\_EMPLOYEE auf.

**Entfernen Sie die Kommentarzeichen des Codes unter Aufgabe 1\_d, und wählen Sie den Code. Um die Packageprozedur aufzurufen, klicken Sie in der SQL Worksheet-Symbolleiste auf das Symbol "Run Script" (F5). Der Code und das Ergebnis werden wie folgt angezeigt:**

```
EXECUTE emp_pkg.add_employee('Samuel', 'Joplin', 30)
```

```

Script Output X
anonymous block completed

```

- e. Vergewissern Sie sich, dass der neue Mitarbeiter in die Tabelle EMPLOYEES eingefügt wurde.

**Entfernen Sie die Kommentarzeichen, und wählen Sie den Code unter Aufgabe 1\_e. Klicken Sie auf eine beliebige Stelle der SELECT-Anweisung und dann in der SQL Worksheet-Symbolleiste auf das Symbol "Execute Statement" (F5), um die Abfrage auszuführen. Der Code und das Ergebnis werden wie folgt angezeigt:**

```
SELECT *
FROM employees
WHERE last_name = 'Joplin';
```

Script Output X										
Task completed in 0.005 seconds										
EMPLOYEE_ID	FIRST_NAME	LAST_NAME	EMAIL	PHONE_NUMBER	HIRE_DATE	JOB_ID	SALARY	COMMISSION_PCT	MANAGER_ID	DEPARTMENT_ID
209	Samuel	Joplin	SJOPLIN		18-NOV-12	SA_REP	1000	0	145	30

2. Erstellen Sie im Package EMP\_PKG zwei überladene Funktionen namens GET\_EMPLOYEE:

- a. Fügen Sie in der Packagespezifikation die folgenden Funktionen hinzu:

- 1) Die Funktion GET\_EMPLOYEE, die den Parameter p\_emp\_id auf Basis des Typs employees.employee\_id%TYPE annimmt. Diese Funktion muss EMPLOYEES%ROWTYPE zurückgeben.
- 2) Die Funktion GET\_EMPLOYEE, die den Parameter p\_family\_name mit dem Typ employees.last\_name%TYPE annimmt. Diese Funktion muss EMPLOYEES%ROWTYPE zurückgeben.

**Entfernen Sie die Kommentarzeichen des Codes unter Aufgabe 2\_a, und wählen Sie den Code.**

```

CREATE OR REPLACE PACKAGE emp_pkg IS
    PROCEDURE add_employee(
        p_first_name employees.first_name%TYPE,
        p_last_name employees.last_name%TYPE,
        p_email employees.email%TYPE,
        p_job employees.job_id%TYPE DEFAULT 'SA_REP',
        p_mgr employees.manager_id%TYPE DEFAULT 145,
        p_sal employees.salary%TYPE DEFAULT 1000,
        p_commission employees.commission_pct%TYPE DEFAULT 0,
        p_deptid employees.department_id%TYPE DEFAULT 30);

    PROCEDURE add_employee(
        p_first_name employees.first_name%TYPE,
        p_last_name employees.last_name%TYPE,
        p_deptid employees.department_id%TYPE);

    PROCEDURE get_employee(
        p.empid IN employees.employee_id%TYPE,
        p_sal OUT employees.salary%TYPE,
        p_job OUT employees.job_id%TYPE);

-- New overloaded get_employees functions specs starts here:

FUNCTION get_employee(p.emp_id employees.employee_id%type)
    return employees%rowtype;

FUNCTION get_employee(p.family_name employees.last_name%type)
    return employees%rowtype;

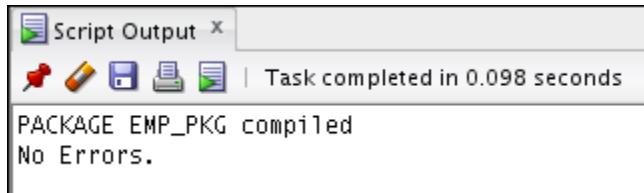
-- New overloaded get_employees functions specs ends here.

END emp_pkg;
/
SHOW ERRORS

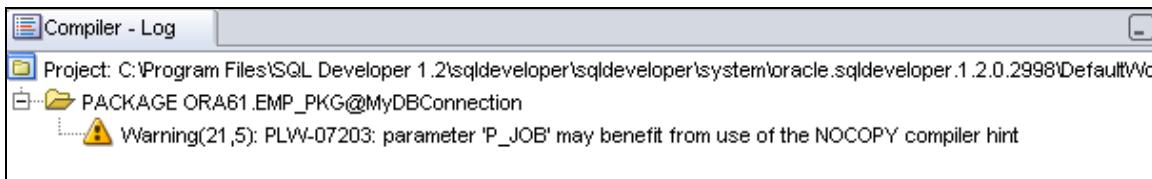
```

- b. Um die Packagespezifikation neu zu erstellen und zu kompilieren, klicken Sie auf **Run Script**.

**Um die Packagespezifikation neu zu erstellen und zu kompilieren, klicken Sie in der SQL Worksheet-Symbolleiste auf das Symbol "Run Script" (F5). Das Ergebnis ist unten abgebildet:**



**Hinweis:** Wie zuvor erwähnt, können Sie den Code mithilfe der folgenden Prozedur rekomplizieren, wenn er eine Fehlermeldung erhält, um die Details des Fehlers oder der Warnung in der Registerkarte **Compiler – Log** anzuzeigen: Um die Packagespezifikation zu kompilieren, klicken Sie im Object Navigator-Baum mit der rechten Maustaste auf den Namen der Packagespezifikation (oder des gesamten Packages) und wählen dann im Kontextmenü die Option **Compile**. Folgende Warnung wird erwartet und dient nur zu Informationszwecken.



- c. Gehen Sie im Packagebody wie folgt vor:
- 1) Um einen Mitarbeiter mithilfe seiner Personalnummer abzufragen, implementieren Sie die erste Funktion `GET_EMPLOYEE`.
  - 2) Um den Gleich-Operator für den Wert zu verwenden, der im Parameter `p_family_name` bereitgestellt wird, implementieren Sie die zweite Funktion `GET_EMPLOYEE`.

**Entfernen Sie die Kommentarzeichen des Codes unter Aufgabe 2\_c, und wählen Sie den Code. Die neu hinzugefügten Funktionen sind im folgenden Codefeld hervorgehoben.**

```
CREATE OR REPLACE PACKAGE emp_pkg IS
    PROCEDURE add_employee(
        p_first_name employees.first_name%TYPE,
        p_last_name employees.last_name%TYPE,
        p_email employees.email%TYPE,
        p_job employees.job_id%TYPE DEFAULT 'SA_REP',
        p_mgr employees.manager_id%TYPE DEFAULT 145,
        p_sal employees.salary%TYPE DEFAULT 1000,
        p_comm employees.commission_pct%TYPE DEFAULT 0,
        p_deptid employees.department_id%TYPE DEFAULT 30);
```

```

PROCEDURE add_employee(
    p_first_name employees.first_name%TYPE,
    p_last_name employees.last_name%TYPE,
    p_deptid employees.department_id%TYPE);

PROCEDURE get_employee(
    p.empid IN employees.employee_id%TYPE,
    p_sal OUT employees.salary%TYPE,
    p_job OUT employees.job_id%TYPE);

-- New overloaded get_employees functions specs starts here:

FUNCTION get_employee(p_emp_id employees.employee_id%type)
    return employees%rowtype;

FUNCTION get_employee(p_family_name
employees.last_name%type)
    return employees%rowtype;

-- New overloaded get_employees functions specs ends here.

END emp_pkg;
/
SHOW ERRORS

-- package body

CREATE OR REPLACE PACKAGE BODY emp_pkg IS
    FUNCTION valid_deptid(p_deptid IN
departments.department_id%TYPE) RETURN BOOLEAN IS
        v_dummy PLS_INTEGER;
    BEGIN
        SELECT 1
        INTO v_dummy
        FROM departments
        WHERE department_id = p_deptid;
        RETURN TRUE;
    EXCEPTION
        WHEN NO_DATA_FOUND THEN
            RETURN FALSE;
    END;

```

```

END valid_deptid;

PROCEDURE add_employee(
    p_first_name employees.first_name%TYPE,
    p_last_name employees.last_name%TYPE,
    p_email employees.email%TYPE,
    p_job employees.job_id%TYPE DEFAULT 'SA_REP',
    p_mgr employees.manager_id%TYPE DEFAULT 145,
    p_sal employees.salary%TYPE DEFAULT 1000,
    p_comm employees.commission_pct%TYPE DEFAULT 0,
    p_deptid employees.department_id%TYPE DEFAULT 30) IS
BEGIN
    IF valid_deptid(p_deptid) THEN
        INSERT INTO employees(employee_id, first_name,
last_name,
                           email, job_id, manager_id, hire_date, salary,
                           commission_pct, department_id)
VALUES (employees_seq.NEXTVAL, p_first_name,
p_last_name,
                           p_email, p_job, p_mgr, TRUNC(SYSDATE), p_sal, p_comm,
                           p_deptid);
    ELSE
        RAISE_APPLICATION_ERROR (-20204, 'Invalid department
ID.
                                         Try again.');
    END IF;
END add_employee;

PROCEDURE add_employee(
    p_first_name employees.first_name%TYPE,
    p_last_name employees.last_name%TYPE,
    p_deptid employees.department_id%TYPE) IS
    p_email employees.email%type;
BEGIN
    p_email := UPPER(SUBSTR(p_first_name, 1,
1) || SUBSTR(p_last_name, 1, 7));
    add_employee(p_first_name, p_last_name, p_email, p_deptid
=> p_deptid);
END;

PROCEDURE get_employee(
    p.empid IN employees.employee_id%TYPE,

```

```

    p_sal OUT employees.salary%TYPE,
    p_job OUT employees.job_id%TYPE) IS
BEGIN
    SELECT salary, job_id
    INTO p_sal, p_job
    FROM employees
    WHERE employee_id = p_empid;
END get_employee;

-- New get_employee function declaration starts here

FUNCTION get_employee(p_emp_id employees.employee_id%type)
    return employees%rowtype IS
    rec_emp employees%rowtype;
BEGIN
    SELECT * INTO rec_emp
    FROM employees
    WHERE employee_id = p_emp_id;
    RETURN rec_emp;
END;

FUNCTION get_employee(p_family_name
employees.last_name%type)
    return employees%rowtype IS
    rec_emp employees%rowtype;
BEGIN
    SELECT * INTO rec_emp
    FROM employees
    WHERE last_name = p_family_name;
    RETURN rec_emp;
END;

-- New overloaded get_employee function declaration ends here

END emp_pkg;
/
SHOW ERRORS

```

- d. Um das Package neu zu erstellen, klicken Sie auf **Run Script**. Kompilieren Sie das Package.

**Um das Package neu zu erstellen und zu kompilieren, klicken Sie in der SQL Worksheet-Symbolleiste auf das Symbol "Run Script" (F5). Das Ergebnis ist unten abgebildet:**

```

  Script Output X
  | Task completed in 0.123 seconds
  PACKAGE EMP_PKG compiled
  No Errors.
  PACKAGE BODY EMP_PKG compiled
  No Errors.

```

- e. Fügen Sie dem Package `EMP_PKG` die Utilityprozedur `PRINT_EMPLOYEE` wie folgt hinzu:
- 1) Die Prozedur nimmt `EMPLOYEES%ROWTYPE` als Parameter an.
  - 2) Die Prozedur zeigt mithilfe des Packages `DBMS_OUTPUT` folgende Elemente für einen Mitarbeiter in einer Zeile an:

- `department_id`
- `employee_id`
- `first_name`
- `last_name`
- `job_id`
- `salary`

**Entfernen Sie die Kommentarzeichen des Codes unter Aufgabe 2\_e, und wählen Sie den Code. Der neu hinzugefügte Code ist im folgenden Codefeld hervorgehoben.**

**-- Package SPECIFICATION**

```

CREATE OR REPLACE PACKAGE emp_pkg IS
  PROCEDURE add_employee(
    p_first_name employees.first_name%TYPE,
    p_last_name employees.last_name%TYPE,
    p_email employees.email%TYPE,
    p_job employees.job_id%TYPE DEFAULT 'SA_REP',
    p_mgr employees.manager_id%TYPE DEFAULT 145,
    p_sal employees.salary%TYPE DEFAULT 1000,
    p_comm employees.commission_pct%TYPE DEFAULT 0,
    p_deptid employees.department_id%TYPE DEFAULT 30);

  PROCEDURE add_employee(
    p_first_name employees.first_name%TYPE,
    p_last_name employees.last_name%TYPE,

```

```

    p_deptid employees.department_id%TYPE);

PROCEDURE get_employee(
    p.empid IN employees.employee_id%TYPE,
    p_sal OUT employees.salary%TYPE,
    p_job OUT employees.job_id%TYPE);

FUNCTION get_employee(p.emp_id employees.employee_id%type)
    return employees%rowtype;

FUNCTION get_employee(p.family_name employees.last_name%type)
    return employees%rowtype;

-- New print_employee print_employee procedure spec

PROCEDURE print_employee(p_rec.emp employees%rowtype);

END emp_pkg;
/
SHOW ERRORS

-- Package BODY

CREATE OR REPLACE PACKAGE BODY emp_pkg IS
    FUNCTION valid_deptid(p_deptid IN
departments.department_id%TYPE) RETURN BOOLEAN IS
        v_dummy PLS_INTEGER;
    BEGIN
        SELECT 1
        INTO v_dummy
        FROM departments
        WHERE department_id = p_deptid;
        RETURN TRUE;
    EXCEPTION
        WHEN NO_DATA_FOUND THEN
            RETURN FALSE;
    END valid_deptid;

PROCEDURE add_employee(
    p_first_name employees.first_name%TYPE,
    p_last_name employees.last_name%TYPE,
    p_email employees.email%TYPE,
    p_job employees.job_id%TYPE DEFAULT 'SA_REP',

```

```

    p_mgr employees.manager_id%TYPE DEFAULT 145,
    p_sal employees.salary%TYPE DEFAULT 1000,
    p_commm employees.commission_pct%TYPE DEFAULT 0,
    p_deptid employees.department_id%TYPE DEFAULT 30) IS
BEGIN
    IF valid_deptid(p_deptid) THEN
        INSERT INTO employees(employee_id, first_name, last_name,
email,
                           job_id, manager_id, hire_date, salary, commission_pct,
department_id)
        VALUES (employees_seq.NEXTVAL, p_first_name, p_last_name,
p_email,
               p_job, p_mgr, TRUNC(SYSDATE), p_sal, p_commm, p_deptid);
    ELSE
        RAISE_APPLICATION_ERROR (-20204, 'Invalid department ID.
Try again.');
    END IF;
END add_employee;

PROCEDURE add_employee(
    p_first_name employees.first_name%TYPE,
    p_last_name employees.last_name%TYPE,
    p_deptid employees.department_id%TYPE) IS
    p_email employees.email%type;
BEGIN
    p_email := UPPER(SUBSTR(p_first_name, 1,
1)||SUBSTR(p_last_name, 1, 7));
    add_employee(p_first_name, p_last_name, p_email, p_deptid =>
p_deptid);
END;

PROCEDURE get_employee(
    p.empid IN employees.employee_id%TYPE,
    p_sal OUT employees.salary%TYPE,
    p_job OUT employees.job_id%TYPE) IS
BEGIN
    SELECT salary, job_id
    INTO p_sal, p_job
    FROM employees
    WHERE employee_id = p.empid;
END get_employee;

FUNCTION get_employee(p.emp_id employees.employee_id%type)
    return employees%rowtype IS

```

```

rec_emp employees%rowtype;
BEGIN
  SELECT * INTO rec_emp
  FROM employees
  WHERE employee_id = p_emp_id;
  RETURN rec_emp;
END;

FUNCTION get_employee(p_family_name employees.last_name%type)
  return employees%rowtype IS
  rec_emp employees%rowtype;
BEGIN
  SELECT * INTO rec_emp
  FROM employees
  WHERE last_name = p_family_name;
  RETURN rec_emp;
END;

-- New print_employees procedure declaration.

PROCEDURE print_employee(p_rec_emp employees%rowtype) IS
BEGIN
  DBMS_OUTPUT.PUT_LINE(p_rec_emp.department_id ||' ' ||
                       p_rec_emp.employee_id||' ' ||
                       p_rec_emp.first_name||' ' ||
                       p_rec_emp.last_name||' ' ||
                       p_rec_emp.job_id||' ' ||
                       p_rec_emp.salary);
END;

END emp_pkg;
/
SHOW ERRORS

```

- f. Um das Package zu erstellen und zu kompilieren, klicken Sie auf **Run Script** (F5).

**Um das Package neu zu erstellen und zu kompilieren, klicken Sie in der SQL Worksheet-Symbolleiste auf das Symbol "Run Script" (F5).**

```

  Script Output X
  | Task completed in 0.125 seconds
  PACKAGE EMP_PKG compiled
  No Errors.
  PACKAGE BODY EMP_PKG compiled
  No Errors.

```

- g. Rufen Sie mit einem anonymen Block die Funktion `EMP_PKG.GET_EMPLOYEE` mit der Personalnummer 100 und dem Nachnamen 'Joplin' auf. Zeigen Sie die Ergebnisse der zurückgegebenen Zeilen mithilfe der Prozedur `PRINT_EMPLOYEE` an. Zuvor müssen Sie jedoch `SET SERVEROUTPUT ON` eingeben.

**Entfernen Sie die Kommentarzeichen des Codes unter Aufgabe 2\_g, und wählen Sie den Code.**

```

SET SERVEROUTPUT ON
BEGIN
    emp_pkg.print_employee(emp_pkg.get_employee(100));
    emp_pkg.print_employee(emp_pkg.get_employee('Joplin'));
END;
/

```

```

  Script Output X
  | Task completed in 0.018 seconds
  anonymous block completed
  90 100 Steven King AD_PRES 24000
  30 209 Samuel Joplin SA_REP 1000

```

3. Da das Unternehmen die Abteilungsdaten nicht oft ändert, können Sie die Performance von `EMP_PKG` steigern, indem Sie eine öffentliche Prozedur namens `INIT_DEPARTMENTS` hinzufügen, um eine private PL/SQL-Tabelle mit gültigen Abteilungsnummern zu füllen. Ändern Sie die Funktion `VALID_DEPTID` so, dass die Werte der Abteilungsnummer mit dem Inhalt der privaten PL/SQL-Tabelle validiert werden.

**Hinweis:** Der Code unter der 3. Aufgabe enthält die Lösung für die Schritte a, b und c.

- Erstellen Sie in der Packagespezifikation eine Prozedur namens `INIT_DEPARTMENTS` ohne Parameter. Fügen Sie im Packagespezifikationsbereich dazu folgenden Code vor der Spezifikation `PRINT_EMPLOYEES` hinzu:
 

```
PROCEDURE init_departments;
```
- Um alle Abteilungsnummern in einer privaten PL/SQL-INDEX BY-Tabelle namens `valid_departments` zu speichern, die `BOOLEAN`-Werte enthält, implementieren Sie im Packagebody die Prozedur `INIT_DEPARTMENTS`.

- 1) Deklarieren Sie die Variable `valid_departments` und ihre Typdefinition `boolean_tab_type` vor allen Prozeduren des Bodys. Geben Sie folgenden Code am Anfang des Packagebodys ein:

```
TYPE boolean_tab_type IS TABLE OF BOOLEAN
INDEX BY BINARY_INTEGER;
valid_departments boolean_tab_type;
```

- 2) Verwenden Sie den Wert der Spalte `department_id` als Index für die Erstellung des Eintrags in der INDEX BY-Tabelle, um dessen Vorhandensein anzuzeigen, und weisen Sie dem Eintrag den Wert `TRUE` zu. Geben Sie die Prozedurdeklaration `INIT_DEPARTMENTS` am Ende des Packagebodys (unmittelbar nach der Prozedur `print_employees`) wie folgt ein:

```
PROCEDURE init_departments IS
BEGIN
  FOR rec IN (SELECT department_id FROM departments)
  LOOP
    valid_departments(rec.department_id) := TRUE;
  END LOOP;
END;
```

- c. Um die Prozedur `INIT_DEPARTMENTS` zur Initialisierung der Tabelle aufzurufen, erstellen Sie im Body den folgenden Initialisierungsblock:

```
BEGIN
  init_departments;
END;
```

**Entfernen Sie die Kommentarzeichen des Codes unter der 3. Aufgabe, und wählen Sie den Code. Der neu hinzugefügte Code ist im folgenden Codefeld hervorgehoben.**

```
-- Package SPECIFICATION

CREATE OR REPLACE PACKAGE emp_pkg IS
  PROCEDURE add_employee(
    p_first_name employees.first_name%TYPE,
    p_last_name employees.last_name%TYPE,
    p_email employees.email%TYPE,
    p_job employees.job_id%TYPE DEFAULT 'SA_REP',
    p_mgr employees.manager_id%TYPE DEFAULT 145,
    p_sal employees.salary%TYPE DEFAULT 1000,
    p_commission employees.commission_pct%TYPE DEFAULT 0,
    p_deptid employees.department_id%TYPE DEFAULT 30);

  PROCEDURE add_employee(
    p_first_name employees.first_name%TYPE,
    p_last_name employees.last_name%TYPE,
    p_deptid employees.department_id%TYPE);
```

```

PROCEDURE get_employee(
    p.empid IN employees.employee_id%TYPE,
    p_sal OUT employees.salary%TYPE,
    p_job OUT employees.job_id%TYPE);

FUNCTION get_employee(p_emp_id employees.employee_id%type)
    return employees%rowtype;

FUNCTION get_employee(p_family_name employees.last_name%type)
    return employees%rowtype;

-- New procedure init_departments spec
PROCEDURE init_departments;

PROCEDURE print_employee(p_rec_emp employees%rowtype);

END emp_pkg;
/
SHOW ERRORS

-- Package BODY

CREATE OR REPLACE PACKAGE BODY emp_pkg IS

-- New type
TYPE boolean_tab_type IS TABLE OF BOOLEAN
    INDEX BY BINARY_INTEGER;
    valid_departments boolean_tab_type;

FUNCTION valid_deptid(p_deptid IN
departments.department_id%TYPE) RETURN BOOLEAN IS
    v_dummy PLS_INTEGER;
BEGIN
    SELECT 1
    INTO v_dummy
    FROM departments
    WHERE department_id = p_deptid;
    RETURN TRUE;
EXCEPTION
    WHEN NO_DATA_FOUND THEN

```

```

        RETURN FALSE;
END valid_deptid;

PROCEDURE add_employee(
    p_first_name employees.first_name%TYPE,
    p_last_name employees.last_name%TYPE,
    p_email employees.email%TYPE,
    p_job employees.job_id%TYPE DEFAULT 'SA_REP',
    p_mgr employees.manager_id%TYPE DEFAULT 145,
    p_sal employees.salary%TYPE DEFAULT 1000,
    p_comm employees.commission_pct%TYPE DEFAULT 0,
    p_deptid employees.department_id%TYPE DEFAULT 30) IS
BEGIN
    IF valid_deptid(p_deptid) THEN

        INSERT INTO employees(employee_id, first_name, last_name,
            email, job_id, manager_id, hire_date, salary,
            commission_pct, department_id)
        VALUES (employees_seq.NEXTVAL, p_first_name, p_last_name,
            p_email, p_job, p_mgr, TRUNC(SYSDATE), p_sal, p_comm,
            p_deptid);
    ELSE
        RAISE_APPLICATION_ERROR (-20204, 'Invalid department ID.
            Try again.');
    END IF;
END add_employee;

PROCEDURE add_employee(
    p_first_name employees.first_name%TYPE,
    p_last_name employees.last_name%TYPE,
    p_deptid employees.department_id%TYPE) IS
    p_email employees.email%TYPE;
BEGIN
    p_email := UPPER(SUBSTR(p_first_name, 1,
    1) || SUBSTR(p_last_name, 1, 7));
    add_employee(p_first_name, p_last_name, p_email, p_deptid =>
    p_deptid);
END;

PROCEDURE get_employee(
    p.empid IN employees.employee_id%TYPE,
    p_sal OUT employees.salary%TYPE,
    p_job OUT employees.job_id%TYPE) IS

```

```

BEGIN
    SELECT salary, job_id
    INTO p_sal, p_job
    FROM employees
    WHERE employee_id = p_empid;
END get_employee;

FUNCTION get_employee(p_emp_id employees.employee_id%type)
    return employees%rowtype IS
    rec_emp employees%rowtype;
BEGIN
    SELECT * INTO rec_emp
    FROM employees
    WHERE employee_id = p_emp_id;
    RETURN rec_emp;
END;

FUNCTION get_employee(p_family_name employees.last_name%type)
    return employees%rowtype IS
    rec_emp employees%rowtype;
BEGIN
    SELECT * INTO rec_emp
    FROM employees
    WHERE last_name = p_family_name;
    RETURN rec_emp;
END;

PROCEDURE print_employee(p_rec_emp employees%rowtype) IS
BEGIN
    DBMS_OUTPUT.PUT_LINE(p_rec_emp.department_id || ' ' ||
                           p_rec_emp.employee_id|| ' ' ||
                           p_rec_emp.first_name|| ' ' ||
                           p_rec_emp.last_name|| ' ' ||
                           p_rec_emp.job_id|| ' ' ||
                           p_rec_emp.salary);
END;

-- New init_departments procedure declaration.

PROCEDURE init_departments IS
BEGIN
    FOR rec IN (SELECT department_id FROM departments)

```

```

LOOP
    valid_departments(rec.department_id) := TRUE;
END LOOP;
END;

-- call the new init_departments procedure.

BEGIN
    init_departments;
END emp_pkg;
/
SHOW ERRORS

CREATE OR REPLACE PACKAGE emp_pkg IS
    PROCEDURE add_employee(
        p_first_name employees.first_name%TYPE,
        p_last_name employees.last_name%TYPE,
        p_email employees.email%TYPE,
        p_job employees.job_id%TYPE DEFAULT 'SA_REP',
        p_mgr employees.manager_id%TYPE DEFAULT 145,
        p_sal employees.salary%TYPE DEFAULT 1000,
        p_comm employees.commission_pct%TYPE DEFAULT 0,
        p_deptid employees.department_id%TYPE DEFAULT 30);

    PROCEDURE add_employee(
        p_first_name employees.first_name%TYPE,
        p_last_name employees.last_name%TYPE,
        p_deptid employees.department_id%TYPE);

    PROCEDURE get_employee(
        p.empid IN employees.employee_id%TYPE,
        p_sal OUT employees.salary%TYPE,
        p_job OUT employees.job_id%TYPE);

    FUNCTION get_employee(p.emp_id employees.employee_id%type)
    return employees%rowtype;

    FUNCTION get_employee(p.family_name
                           employees.last_name%type)
    return employees%rowtype;

--New procedure init_departments spec

PROCEDURE init_departments;

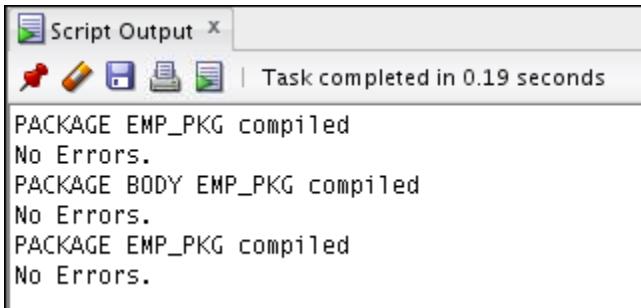
PROCEDURE print_employee(p_rec_emp employees%rowtype);

END emp_pkg;
/
SHOW ERRORS

```

- d. Um das Package neu zu erstellen und zu komplizieren, klicken Sie auf **Run Script** (F5).

**Um das Package neu zu erstellen und zu komplizieren, klicken Sie in der SQL Worksheet-Symbolleiste auf das Symbol "Run Script" (F5).**



```
Script Output X
Task completed in 0.19 seconds
PACKAGE EMP_PKG compiled
No Errors.
PACKAGE BODY EMP_PKG compiled
No Errors.
PACKAGE EMP_PKG compiled
No Errors.
```

4. Ändern Sie die Validierungsverarbeitungsfunktion VALID\_DEPTID so, dass die private PL/SQL-Tabelle der Abteilungsnummern verwendet wird.
- a. Ändern Sie die Funktion VALID\_DEPTID so, dass die Validierung mithilfe der Werte in der PL/SQL-Tabelle der Abteilungsnummern erfolgt. Um das Package zu erstellen und zu komplizieren, klicken Sie auf **Run Script** (F5).
- Entfernen Sie die Kommentarzeichen des Codes unter Aufgabe 4\_a, und wählen Sie den Code. Um das Package zu erstellen und zu komplizieren, klicken Sie auf "Run Script" (F5). Der neu hinzugefügte Code ist im folgenden Codefeld hervorgehoben.**

-- Package SPECIFICATION

```
CREATE OR REPLACE PACKAGE emp_pkg IS
  PROCEDURE add_employee(
    p_first_name employees.first_name%TYPE,
    p_last_name employees.last_name%TYPE,
    p_email employees.email%TYPE,
    p_job employees.job_id%TYPE DEFAULT 'SA_REP',
    p_mgr employees.manager_id%TYPE DEFAULT 145,
    p_sal employees.salary%TYPE DEFAULT 1000,
    p_comm employees.commission_pct%TYPE DEFAULT 0,
    p_deptid employees.department_id%TYPE DEFAULT 30);

  PROCEDURE add_employee(
    p_first_name employees.first_name%TYPE,
    p_last_name employees.last_name%TYPE,
    p_deptid employees.department_id%TYPE);

  PROCEDURE get_employee(
    p.empid IN employees.employee_id%TYPE,
    p_sal OUT employees.salary%TYPE,
    p_job OUT employees.job_id%TYPE);
```

```

FUNCTION get_employee(p_emp_id employees.employee_id%type)
    return employees%rowtype;

FUNCTION get_employee(p_family_name
    employees.last_name%type)
    return employees%rowtype;

-- New procedure init_departments spec
PROCEDURE init_departments;

PROCEDURE print_employee(p_rec_emp employees%rowtype);

END emp_pkg;
/
SHOW ERRORS

-- Package BODY

CREATE OR REPLACE PACKAGE BODY emp_pkg IS

TYPE boolean_tab_type IS TABLE OF BOOLEAN
    INDEX BY BINARY_INTEGER;
valid_departments boolean_tab_type;

FUNCTION valid_deptid(p_deptid IN
departments.department_id%TYPE) RETURN BOOLEAN IS
    v_dummy PLS_INTEGER;
BEGIN
    RETURN valid_departments.exists(p_deptid);
EXCEPTION
    WHEN NO_DATA_FOUND THEN
        RETURN FALSE;
END valid_deptid;

PROCEDURE add_employee(
    p_first_name employees.first_name%TYPE,
    p_last_name employees.last_name%TYPE,
    p_email employees.email%TYPE,
    p_job employees.job_id%TYPE DEFAULT 'SA_REP',
    p_mgr employees.manager_id%TYPE DEFAULT 145,
    p_sal employees.salary%TYPE DEFAULT 1000,

```

```

    p_comm employees.commission_pct%TYPE DEFAULT 0,
    p_deptid employees.department_id%TYPE DEFAULT 30) IS
BEGIN
    IF valid_deptid(p_deptid) THEN
        INSERT INTO employees(employee_id, first_name,
        last_name, email, job_id, manager_id, hire_date,
        salary, commission_pct, department_id)
        VALUES (employees_seq.NEXTVAL, p_first_name,
        p_last_name, p_email,
        p_job, p_mgr, TRUNC(SYSDATE), p_sal, p_comm,p_deptid);
    ELSE
        RAISE_APPLICATION_ERROR (-20204, 'Invalid department ID.
                                         Try again.');
    END IF;
END add_employee;

PROCEDURE add_employee(
    p_first_name employees.first_name%TYPE,
    p_last_name employees.last_name%TYPE,
    p_deptid employees.department_id%TYPE) IS
    p_email employees.email%type;
BEGIN
    p_email := UPPER(SUBSTR(p_first_name, 1,
1)||SUBSTR(p_last_name, 1, 7));
    add_employee(p_first_name, p_last_name, p_email, p_deptid =>
p_deptid);
END;

PROCEDURE get_employee(
    p.empid IN employees.employee_id%TYPE,
    p_sal OUT employees.salary%TYPE,
    p.job OUT employees.job_id%TYPE) IS
BEGIN
    SELECT salary, job_id
    INTO p_sal, p_job
    FROM employees
    WHERE employee_id = p.empid;
END get_employee;

FUNCTION get_employee(p.emp_id employees.employee_id%type)
    return employees%rowtype IS
    rec_emp employees%rowtype;
BEGIN

```

```

        SELECT * INTO rec_emp
        FROM employees
        WHERE employee_id = p_emp_id;
        RETURN rec_emp;
    END;

FUNCTION get_employee(p_family_name employees.last_name%type)
    return employees%rowtype IS
    rec_emp employees%rowtype;
BEGIN
    SELECT * INTO rec_emp
    FROM employees
    WHERE last_name = p_family_name;
    RETURN rec_emp;
END;

PROCEDURE print_employee(p_rec_emp employees%rowtype) IS
BEGIN
    DBMS_OUTPUT.PUT_LINE(p_rec_emp.department_id || ' ' ||
                         p_rec_emp.employee_id|| ' ' ||
                         p_rec_emp.first_name|| ' ' ||
                         p_rec_emp.last_name|| ' ' ||
                         p_rec_emp.job_id|| ' ' ||
                         p_rec_emp.salary);
END;

-- New init_departments procedure declaration.

PROCEDURE init_departments IS
BEGIN
    FOR rec IN (SELECT department_id FROM departments)
    LOOP
        valid_departments(rec.department_id) := TRUE;
    END LOOP;
END;

-- call the new init_departments procedure.

BEGIN
    init_departments;
END emp_pkg;

/
SHOW ERRORS

```

```

Script Output x
PACKAGE EMP_PKG compiled
No Errors.
PACKAGE BODY EMP_PKG compiled
No Errors.

```

- b. Testen Sie den Code, indem Sie ADD\_EMPLOYEE mit dem Namen James Bond in Abteilung 15 aufrufen. Was geschieht?

**Entfernen Sie die Kommentarzeichen des Codes unter Aufgabe 4\_b, und wählen Sie den Code.**

```
EXECUTE emp_pkg.add_employee('James', 'Bond', 15)
```

**Um das Einfügen eines neuen Mitarbeiters zu testen, klicken Sie in der SQL Worksheet-Symbolleiste auf das Symbol "Run Script" (F5). Der INSERT-Vorgang zum Einfügen des Mitarbeiters wird mit einer Exception abgebrochen, da Abteilung 15 nicht vorhanden ist.**

```

Script Output x
Error starting at line 788 in command:
EXECUTE emp_pkg.add_employee('James', 'Bond', 15)
Error report:
ORA-20204: Invalid department ID.
          Try again.
ORA-06512: at "ORA61.EMP_PKG", line 34
ORA-06512: at "ORA61.EMP_PKG", line 46
ORA-06512: at line 1

```

- c. Fügen Sie eine neue Abteilung ein. Geben Sie die Abteilungsnummer 15 und den Abteilungsnamen 'Security' ein. Schreiben Sie die Änderungen fest, und verifizieren Sie sie.

**Entfernen Sie die Kommentarzeichen des Codes unter Aufgabe 4\_c, und wählen Sie den Code. Der Code und das Ergebnis werden wie folgt angezeigt:**

```
INSERT INTO departments (department_id, department_name)
VALUES (15, 'Security');
COMMIT;
```

```

Script Output x
Task completed in 0.032 seconds
1 rows inserted.
committed.

```

- d. Testen Sie den Code erneut, indem Sie ADD\_EMPLOYEE mit dem Namen James Bond in Abteilung 15 aufrufen. Was geschieht?

**Entfernen Sie die Kommentarzeichen des Codes unter Aufgabe 4\_d, und wählen Sie den Code. Der Code und das Ergebnis werden wie folgt angezeigt:**

```
EXECUTE emp_pkg.add_employee( 'James' , 'Bond' , 15)
```

The screenshot shows the 'Script Output' window from Oracle SQL Developer. It displays the following text:  
 Task completed in 1.925 seconds  
 Error starting at line 802 in command:  
 EXECUTE emp\_pkg.add\_employee('James', 'Bond', 15)  
 Error report:  
 ORA-20204: Invalid department ID.  
 Try again.  
 ORA-06512: at "ORA61.EMP\_PKG", line 34  
 ORA-06512: at "ORA61.EMP\_PKG", line 46  
 ORA-06512: at line 1

**Der INSERT-Vorgang zum Einfügen des Mitarbeiters wird mit einer Exception abgebrochen. Abteilung 15 ist im assoziativen Array der PL/SQL-Package-zustandsvariablen (INDEX BY-Tabelle) nicht als Eintrag vorhanden.**

- e. Um die interne PL/SQL-Tabelle mit den neuesten Abteilungsdaten zu aktualisieren, führen Sie die Prozedur EMP\_PKG.INIT\_DEPARTMENTS aus.

**Entfernen Sie die Kommentarzeichen des Codes unter Aufgabe 4\_e, und wählen Sie den Code. Der Code und das Ergebnis werden wie folgt angezeigt:**

```
EXECUTE EMP_PKG.INIT_DEPARTMENTS
```

The screenshot shows the 'Script Output' window from Oracle SQL Developer. It displays the following text:  
 Task completed in 0.016 seconds  
 anonymous block completed

- f. Testen Sie den Code, indem Sie ADD\_EMPLOYEE mit dem Mitarbeiternamen James Bond aufrufen, der in Abteilung 15 arbeitet. Was geschieht?

**Entfernen Sie die Kommentarzeichen des Codes unter Aufgabe 4\_f, und wählen Sie den Code. Der Code und das Ergebnis werden wie folgt angezeigt.**

```
EXECUTE emp_pkg.add_employee( 'James' , 'Bond' , 15)
```

**Die Zeile wird schließlich eingefügt, da der Record für Abteilung 15 in der Datenbank und der INDEX BY-Tabelle des PL/SQL-Packages vorhanden ist. Zu diesem Zweck wurde die Prozedur EMP\_PKG.INIT\_DEPARTMENTS aufgerufen, die die Packagezustandsdaten aktualisiert.**

The screenshot shows the 'Script Output' window from Oracle SQL Developer. It displays the following text:  
 Task completed in 0.016 seconds  
 anonymous block completed

- g. Löschen Sie den Mitarbeiter James Bond und die Abteilung 15 aus den entsprechenden Tabellen, schreiben Sie die Änderungen fest, und aktualisieren Sie die Abteilungsdaten, indem Sie die Prozedur EMP\_PKG.INIT\_DEPARTMENTS aufrufen.

**Entfernen Sie die Kommentarzeichen des Codes unter Aufgabe 4\_g, und wählen Sie den Code. Der Code und das Ergebnis werden wie folgt angezeigt.**

```
DELETE FROM employees
WHERE first_name = 'James' AND last_name = 'Bond';
DELETE FROM departments WHERE department_id = 15;
COMMIT;
EXECUTE EMP_PKG.INIT_DEPARTMENTS
```

The screenshot shows the 'Script Output' window with the following content:

```
Script Output X
Task completed in 0.027 seconds
1 rows deleted.
1 rows deleted.
committed.
anonymous block completed
```

5. Reorganisieren Sie die Unterprogramme in der Packagespezifikation und im Body so, dass sie in alphabetischer Reihenfolge angezeigt werden.

- a. Bearbeiten Sie die Packagespezifikation so, dass die Unterprogramme in alphabetischer Reihenfolge neu angeordnet werden. Um die Packagespezifikation neu zu erstellen, klicken Sie auf **Run Script**. Kompilieren Sie die Packagespezifikation. Was geschieht?

**Entfernen Sie die Kommentarzeichen des Codes unter Aufgabe 5\_a, und wählen Sie den Code. Um das Package neu zu erstellen und zu kompilieren, klicken Sie in der SQL Worksheet-Symbolleiste auf das Symbol "Run Script" (F5). Der Code und das Ergebnis werden wie folgt angezeigt. Die Unterprogramme der Packagespezifikation sind bereits alphabetisch sortiert.**

```
CREATE OR REPLACE PACKAGE emp_pkg IS

-- the package spec is already in an alphabetical order.

PROCEDURE add_employee(
    p_first_name employees.first_name%TYPE,
    p_last_name employees.last_name%TYPE,
    p_email employees.email%TYPE,
    p_job employees.job_id%TYPE DEFAULT 'SA_REP',
    p_mgr employees.manager_id%TYPE DEFAULT 145,
    p_sal employees.salary%TYPE DEFAULT 1000,
    p_comm employees.commission_pct%TYPE DEFAULT 0,
    p_deptid employees.department_id%TYPE DEFAULT 30);
```

```

PROCEDURE add_employee(
    p_first_name employees.first_name%TYPE,
    p_last_name employees.last_name%TYPE,
    p_deptid employees.department_id%TYPE);

PROCEDURE get_employee(
    p.empid IN employees.employee_id%TYPE,
    p_sal OUT employees.salary%TYPE,
    p_job OUT employees.job_id%TYPE);

FUNCTION get_employee(p.emp_id employees.employee_id%type)
    return employees%rowtype;

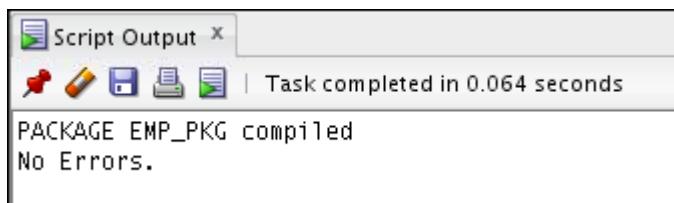
FUNCTION get_employee(p.family_name employees.last_name%type)
    return employees%rowtype;

PROCEDURE init_departments;

PROCEDURE print_employee(p_rec_emp employees%rowtype);

END emp_pkg;
/
SHOW ERRORS

```



- b. Bearbeiten Sie den Packagebody so, dass alle Unterprogramme in alphabetischer Reihenfolge neu angeordnet werden. Um die Packagespezifikation neu zu erstellen, klicken Sie auf **Run Script**. Rekompilieren Sie die Packagespezifikation. Was geschieht?

**Entfernen Sie die Kommentarzeichen des Codes unter Aufgabe 5\_b, und wählen Sie den Code. Um das Package neu zu erstellen, klicken Sie in der SQL Worksheet-Symbolleiste auf das Symbol "Run Script" (F5). Der Code und das Ergebnis werden wie folgt angezeigt.**

```
-- Package BODY
CREATE OR REPLACE PACKAGE BODY emp_pkg IS
    TYPE boolean_tab_type IS TABLE OF BOOLEAN
        INDEX BY BINARY_INTEGER;
    valid_departments boolean_tab_type;

    PROCEDURE add_employee(
        p_first_name employees.first_name%TYPE,
```

```

    p_last_name employees.last_name%TYPE,
    p_email employees.email%TYPE,
    p_job employees.job_id%TYPE DEFAULT 'SA_REP',
    p_mgr employees.manager_id%TYPE DEFAULT 145,
    p_sal employees.salary%TYPE DEFAULT 1000,
    p_comm employees.commission_pct%TYPE DEFAULT 0,
    p_deptid employees.department_id%TYPE DEFAULT 30) IS
BEGIN
    IF valid_deptid(p_deptid) THEN
        INSERT INTO employees(employee_id, first_name, last_name,
email,
                           job_id, manager_id, hire_date, salary, commission_pct,
department_id)
        VALUES (employees_seq.NEXTVAL, p_first_name, p_last_name,
p_email,
               p_job, p_mgr, TRUNC(SYSDATE), p_sal, p_comm, p_deptid);
    ELSE
        RAISE_APPLICATION_ERROR (-20204, 'Invalid department ID.
Try again.');
    END IF;
END add_employee;

PROCEDURE add_employee(
    p_first_name employees.first_name%TYPE,
    p_last_name employees.last_name%TYPE,
    p_deptid employees.department_id%TYPE) IS
    p_email employees.email%TYPE;
BEGIN
    p_email := UPPER(SUBSTR(p_first_name, 1,
1) || SUBSTR(p_last_name, 1, 7));
    add_employee(p_first_name, p_last_name, p_email, p_deptid =>
p_deptid);
END;

PROCEDURE get_employee(
    p.empid IN employees.employee_id%TYPE,
    p_sal OUT employees.salary%TYPE,
    p_job OUT employees.job_id%TYPE) IS
BEGIN
    SELECT salary, job_id
    INTO p_sal, p_job
    FROM employees
    WHERE employee_id = p.empid;
END get_employee;

```

```

FUNCTION get_employee(p_emp_id employees.employee_id%type)
    return employees%rowtype IS
    rec_emp employees%rowtype;
BEGIN
    SELECT * INTO rec_emp
    FROM employees
    WHERE employee_id = p_emp_id;
    RETURN rec_emp;
END;

FUNCTION get_employee(p_family_name employees.last_name%type)
    return employees%rowtype IS
    rec_emp employees%rowtype;
BEGIN
    SELECT * INTO rec_emp
    FROM employees
    WHERE last_name = p_family_name;
    RETURN rec_emp;
END;

PROCEDURE init_departments IS
BEGIN
    FOR rec IN (SELECT department_id FROM departments)
    LOOP
        valid_departments(rec.department_id) := TRUE;
    END LOOP;
END;

PROCEDURE print_employee(p_rec_emp employees%rowtype) IS
BEGIN
    DBMS_OUTPUT.PUT_LINE(p_rec_emp.department_id || ' ' ||
                           p_rec_emp.employee_id|| ' ' ||
                           p_rec_emp.first_name|| ' ' ||
                           p_rec_emp.last_name|| ' ' ||
                           p_rec_emp.job_id|| ' ' ||
                           p_rec_emp.salary);
END;

FUNCTION valid_deptid(p_deptid IN
departments.department_id%TYPE) RETURN BOOLEAN IS
    v_dummy PLS_INTEGER;
BEGIN

```

```

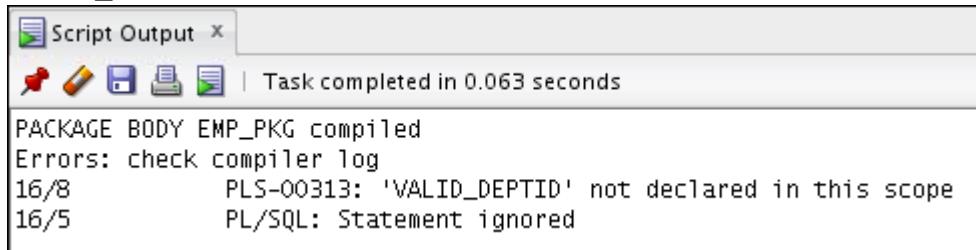
        RETURN valid_departments.exists(p_deptid);
EXCEPTION
    WHEN NO_DATA_FOUND THEN
        RETURN FALSE;
END valid_deptid;

BEGIN
    init_departments;
END emp_pkg;

/
SHOW ERRORS

```

**Bei der Kompilierung des Packages tritt ein Fehler auf, da die Funktion VALID\_DEPTID vor ihrer Deklaration referenziert wurde.**



The screenshot shows the 'Script Output' window from Oracle SQL Developer. It displays the following text:

```

Script Output x
Task completed in 0.063 seconds
PACKAGE BODY EMP_PKG compiled
Errors: check compiler log
16/8      PLS-00313: 'VALID_DEPTID' not declared in this scope
16/5      PL/SQL: Statement ignored

```

- c. Beheben Sie den Kompilierungsfehler mithilfe einer Vorwärtsdeklaration im Body für die entsprechende Unterprogrammreferenz. Klicken Sie auf **Run Script**, um das Package neu zu erstellen, und rekomplizieren Sie dann das Package. Was geschieht?  
**Entfernen Sie die Kommentarzeichen des Codes unter Aufgabe 5\_c, und wählen Sie den Code. Die Vorwärtsdeklaration der Funktion ist im folgenden Codefeld hervorgehoben. Um das Package neu zu erstellen und zu kompilieren, klicken Sie in der SQL Worksheet-Symbolleiste auf das Symbol "Run Script" (F5). Der Code und das Ergebnis werden wie folgt angezeigt.**

```

-- Package BODY

CREATE OR REPLACE PACKAGE BODY emp_pkg IS
    TYPE boolean_tab_type IS TABLE OF BOOLEAN
        INDEX BY BINARY_INTEGER;
    valid_departments boolean_tab_type;

-- forward declaration of valid_deptid

FUNCTION valid_deptid(p_deptid IN
    departments.department_id%TYPE)
    RETURN BOOLEAN;

```

```

PROCEDURE add_employee(
    p_first_name employees.first_name%TYPE,
    p_last_name employees.last_name%TYPE,
    p_email employees.email%TYPE,
    p_job employees.job_id%TYPE DEFAULT 'SA_REP',
    p_mgr employees.manager_id%TYPE DEFAULT 145,
    p_sal employees.salary%TYPE DEFAULT 1000,
    p_comm employees.commission_pct%TYPE DEFAULT 0,
    p_deptid employees.department_id%TYPE DEFAULT 30) IS
BEGIN
    IF valid_deptid(p_deptid) THEN -- valid_deptid function
refernced
        INSERT INTO employees(employee_id, first_name, last_name,
email,
                job_id, manager_id, hire_date, salary, commission_pct,
department_id)
        VALUES (employees_seq.NEXTVAL, p_first_name, p_last_name,
p_email,
                p_job, p_mgr, TRUNC(SYSDATE), p_sal, p_comm, p_deptid);
    ELSE
        RAISE_APPLICATION_ERROR (-20204, 'Invalid department ID.
Try again.');
    END IF;
END add_employee;

PROCEDURE add_employee(
    p_first_name employees.first_name%TYPE,
    p_last_name employees.last_name%TYPE,
    p_deptid employees.department_id%TYPE) IS
    p_email employees.email%type;
BEGIN
    p_email := UPPER(SUBSTR(p_first_name, 1,
1) || SUBSTR(p_last_name, 1, 7));
    add_employee(p_first_name, p_last_name, p_email, p_deptid =>
p_deptid);
END;

PROCEDURE get_employee(
    p.empid IN employees.employee_id%TYPE,
    p_sal OUT employees.salary%TYPE,
    p_job OUT employees.job_id%TYPE) IS
BEGIN
    SELECT salary, job_id
    INTO p_sal, p_job

```

```

        FROM employees
        WHERE employee_id = p_empid;
    END get_employee;

FUNCTION get_employee(p_emp_id employees.employee_id%type)
    return employees%rowtype IS
    rec_emp employees%rowtype;
BEGIN
    SELECT * INTO rec_emp
    FROM employees
    WHERE employee_id = p_emp_id;
    RETURN rec_emp;
END;

FUNCTION get_employee(p_family_name employees.last_name%type)
    return employees%rowtype IS
    rec_emp employees%rowtype;
BEGIN
    SELECT * INTO rec_emp
    FROM employees
    WHERE last_name = p_family_name;
    RETURN rec_emp;
END;

-- New alphabetical location of function init_departments.

PROCEDURE init_departments IS
BEGIN
    FOR rec IN (SELECT department_id FROM departments)
    LOOP
        valid_departments(rec.department_id) := TRUE;
    END LOOP;
END;

PROCEDURE print_employee(p_rec_emp employees%rowtype) IS
BEGIN
    DBMS_OUTPUT.PUT_LINE(p_rec_emp.department_id || ' ' ||
                           p_rec_emp.employee_id|| ' ' ||
                           p_rec_emp.first_name|| ' ' ||
                           p_rec_emp.last_name|| ' ' ||
                           p_rec_emp.job_id|| ' ' ||
                           p_rec_emp.salary);

```

```
END;

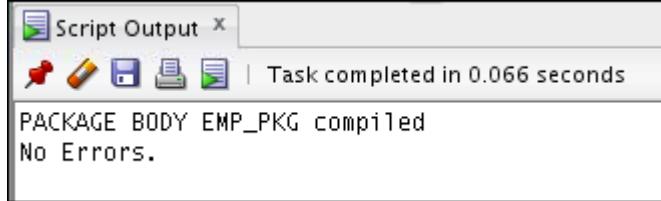
-- New alphabetical location of function valid_deptid.

FUNCTION valid_deptid(p_deptid IN
departments.department_id%TYPE) RETURN BOOLEAN IS
    v_dummy PLS_INTEGER;
BEGIN
    RETURN valid_departments.exists(p_deptid);
EXCEPTION
    WHEN NO_DATA_FOUND THEN
        RETURN FALSE;
END valid_deptid;

BEGIN
    init_departments;
END emp_pkg;

/
SHOW ERRORS
```

**Eine Vorwärtsdeklaration der Funktion VALID\_DEPTID ermöglicht die erfolgreiche Komplilierung des Packagebodys wie unten abgebildet:**



The screenshot shows the 'Script Output' window from Oracle SQL Developer. It displays the message 'Task completed in 0.066 seconds' and 'PACKAGE BODY EMP\_PKG compiled No Errors.' This indicates that the package body was successfully compiled without any errors.

```
Script Output x
| Task completed in 0.066 seconds
PACKAGE BODY EMP_PKG compiled
No Errors.
```

# **Übungen zu Lektion 6 – Von Oracle bereitgestellte Packages zur Anwendungsentwicklung**

**Kapitel 6**

# Übungen zu Lektion 6 – Überblick

---

## Lektionsüberblick

In dieser Übung generieren Sie mit dem Package UTL\_FILE einen Bericht über die Mitarbeiter der einzelnen Abteilungen in Form einer Textdatei.

### Hinweis:

1. Führen Sie vor Beginn dieser Übung das Skript  
`/home/oracle/plpu/code_ex/code_ex_scripts/clean_up_scripts/  
cleanup_06.sql` aus.
2. Wenn Sie einen Schritt in einer Übung ausgelassen haben, führen Sie das entsprechende Lösungsskript zu diesem Übungsschritt aus, bevor Sie mit dem nächsten Schritt oder der nächsten Übung fortfahren.

# Übung 1 zu Lektion 6 – Package UTL\_FILE

---

## Überblick

In dieser Übung generieren Sie mit dem Package UTL\_FILE einen Bericht über die Mitarbeiter der einzelnen Abteilungen in Form einer Textdatei. Zuerst erstellen Sie mithilfe des Packages UTL\_FILE eine Prozedur namens EMPLOYEE\_REPORT, die in einer Betriebssystemdatei einen Mitarbeiterbericht generiert. Der Bericht soll eine Liste der Mitarbeiter generieren, deren Gehalt über dem Durchschnittsgehalt ihrer jeweiligen Abteilung liegt. Schließlich sehen Sie sich die generierte Ausgabe der Textdatei an.

**Hinweis:** Führen Sie vor den folgenden Aufgaben das Skript cleanup\_06.sql in /home/oracle/plpu/code\_ex/code\_ex\_scripts/clean\_up\_scripts/ aus.

## Aufgabe

1. Erstellen Sie mithilfe des Packages UTL\_FILE eine Prozedur namens EMPLOYEE\_REPORT, die in einer Betriebssystemdatei einen Mitarbeiterbericht generiert. Der Bericht soll eine Liste der Mitarbeiter generieren, deren Gehalt über dem Durchschnittsgehalt ihrer jeweiligen Abteilung liegt.
  - a. Das Programm soll zwei Parameter annehmen: Der erste Parameter ist das Ausgabeverzeichnis. Der zweite Parameter ist der Name der geschriebenen Textdatei.  
**Hinweis:** Verwenden Sie den Verzeichnispfadwert UTL\_FILE. Fügen Sie einen Bereich zur Exception-Behandlung hinzu, um Fehler zu behandeln, die möglicherweise bei Verwendung des Packages UTL\_FILE auftreten.
  - b. Um die Prozedur zu erstellen und zu kompilieren, klicken Sie in der SQL Worksheet-Symbolleiste auf das Symbol **Run Script** (F5).
2. Rufen Sie die Prozedur mithilfe der folgenden beiden Argumente auf:
  - a. Verwenden Sie REPORTS\_DIR als Alias für das Directory-Objekt als ersten Parameter.
  - b. Verwenden Sie sal\_rpt61.txt als zweiten Parameter.
3. Zeigen Sie die generierte Ausgabedatei wie folgt an:
  - a. Doppelklicken Sie auf Ihrem Desktop auf das Symbol **Terminal**. Das Fenster **Terminal** wird geöffnet.
  - b. Wechseln Sie an der Eingabeaufforderung \$ mit dem Befehl cd in das Verzeichnis /home/oracle/labs/plpu/reports, das die generierte Ausgabedatei sal\_rpt61.txt enthält.  
**Hinweis:** Mit dem Befehl pwd können Sie das aktuelle Arbeitsverzeichnis auflisten.
  - c. Listen Sie den Inhalt des aktuellen Verzeichnisses mit dem Befehl ls auf.
  - d. Öffnen Sie die übertragene Datei sal\_rpt61.txt mithilfe des Befehls gedit oder mit einem Editor Ihrer Wahl.

## Übung 1 zu Lektion 6 – Lösung: Package UTL\_FILE

In dieser Übung generieren Sie mit dem Package UTL\_FILE einen Bericht über die Mitarbeiter der einzelnen Abteilungen in Form einer Textdatei. Zuerst erstellen Sie mithilfe des Packages UTL\_FILE eine Prozedur namens EMPLOYEE\_REPORT, die in einer Betriebssystemdatei einen Mitarbeiterbericht generiert. Der Bericht soll eine Liste der Mitarbeiter generieren, deren Gehalt über dem Durchschnittsgehalt ihrer jeweiligen Abteilung liegt. Schließlich sehen Sie sich die generierte Ausgabe der Textdatei an.

1. Erstellen Sie mithilfe des Packages UTL\_FILE eine Prozedur namens EMPLOYEE\_REPORT, die in einer Betriebssystemdatei einen Mitarbeiterbericht generiert. Der Bericht soll eine Liste der Mitarbeiter generieren, deren Gehalt über dem Durchschnittsgehalt ihrer jeweiligen Abteilung liegt.
  - a. Das Programm soll zwei Parameter annehmen: Der erste Parameter ist das Ausgabeverzeichnis. Der zweite Parameter ist der Name der geschriebenen Textdatei.

**Hinweis:** Verwenden Sie den Verzeichnispfadwert UTL\_FILE. Fügen Sie einen Bereich zur Exception-Behandlung hinzu, um Fehler zu behandeln, die möglicherweise bei Verwendung des Packages UTL\_FILE auftreten.

**Öffnen Sie die Datei im Skript /home/oracle/labs/plpu/solns/sol\_06.sql.**

**Entfernen Sie die Kommentarzeichen, und wählen Sie den Code unter der 1. Aufgabe.**

```
-- Verify with your instructor that the database initSID.ora
-- file has the directory path you are going to use with this --
procedure.
-- For example, there should be an entry such as:
-- UTL_FILE_DIR = /home1/teachX/UTL_FILE in your initSID.ora
-- (or the SPFILE)
-- HOWEVER: The course has a directory alias provided called
-- "REPORTS_DIR" that is associated with an appropriate
-- directory. Use the directory alias name in quotes for the
-- first parameter to create a file in the appropriate
-- directory.

CREATE OR REPLACE PROCEDURE employee_report(
  p_dir IN VARCHAR2, p_filename IN VARCHAR2) IS
  f UTL_FILE.FILE_TYPE;
  CURSOR cur_avg IS
    SELECT last_name, department_id, salary
    FROM employees outer
    WHERE salary > (SELECT AVG(salary)
                     FROM employees inner
                     Where department_id = outer.department_id)
    ORDER BY department_id;
```

```

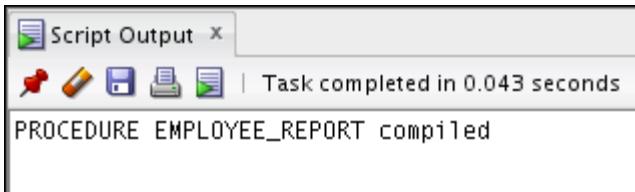
BEGIN
  f := UTL_FILE.FOPEN(p_dir, p_filename,'W');

  UTL_FILE.PUT_LINE(f, 'Employees who earn more than average
    salary: ');
  UTL_FILE.PUT_LINE(f, 'REPORT GENERATED ON ' || SYSDATE);
  UTL_FILE.NEW_LINE(f);
  FOR emp IN cur_avg
  LOOP

    UTL_FILE.PUT_LINE(f,
      RPAD(emp.last_name, 30) || ' ' || 
      LPAD(NVL(TO_CHAR(emp.department_id,'9999'),'-'), 5) || ' '
    ||
      LPAD(TO_CHAR(emp.salary, '$99,999.00'), 12));
  END LOOP;
  UTL_FILE.NEW_LINE(f);
  UTL_FILE.PUT_LINE(f, '*** END OF REPORT ***');
  UTL_FILE.FCLOSE(f);
END employee_report;
/

```

- b. Um die Prozedur zu erstellen und zu kompilieren, klicken Sie in der SQL Worksheet-Symbolleiste auf das Symbol **Run Script** (F5).



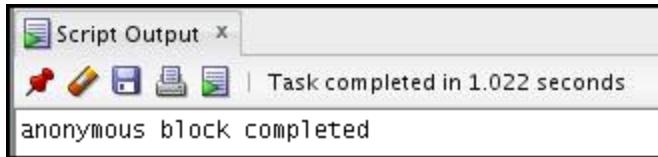
2. Rufen Sie die Prozedur mithilfe der folgenden Argumente auf:

- Verwenden Sie `REPORTS_DIR` als Alias für das Directory-Objekt als ersten Parameter.
- Verwenden Sie `sal_rpt61.txt` als zweiten Parameter.

**Entfernen Sie die Kommentarzeichen, und wählen Sie den Code unter der 2. Aufgabe. Um die Prozedur auszuführen, klicken Sie in der SQL Worksheet-Symbolleiste auf das Symbol "Run Script" (F5). Das Ergebnis ist unten abgebildet. Stellen Sie sicher, dass sich die externe Datei und die Datenbank auf demselben Rechner befinden.**

```
-- For example, if you are student ora61, use 61 as a prefix

EXECUTE employee_report('REPORTS_DIR','sal_rpt61.txt')
```



3. Zeigen Sie die generierte Ausgabedatei wie folgt an:
  - a. Doppelklicken Sie auf Ihrem Desktop auf das Symbol **Terminal**. Das Fenster **Terminal** wird geöffnet.
  - b. Wechseln Sie wie folgt an der Eingabeaufforderung \$ mit dem Befehl cd in das Verzeichnis /home/oracle/labs/plpu/reports, das die generierte Ausgabedatei sal\_rpt61.txt enthält:

The terminal window title is 'oracle@EDRSR15P1:~/labs/plpu/reports'. The command history shows:
 

```
[oracle@EDRSR15P1 Desktop]$ cd /home/oracle/labs/plpu/reports
[oracle@EDRSR15P1 reports]$ pwd
/home/oracle/labs/plpu/reports
[oracle@EDRSR15P1 reports]$
```

**Hinweis:** Mit dem Befehl `pwd` können Sie das aktuelle Arbeitsverzeichnis auflisten, wie im Screenshot oben zu sehen ist.

- c. Listen Sie wie folgt den Inhalt des aktuellen Verzeichnisses mit dem Befehl ls auf:

The terminal window title is 'oracle@EDRSR15P1:~/labs/plpu/reports'. The command history shows:
 

```
[oracle@EDRSR15P1 Desktop]$ cd /home/oracle/labs/plpu/reports
[oracle@EDRSR15P1 reports]$ pwd
/home/oracle/labs/plpu/reports
[oracle@EDRSR15P1 reports]$ ls
instructor.txt salreport2.txt sal_rpt61.txt
```

The file 'sal\_rpt61.txt' is highlighted with a red rectangle.

**Beachten Sie die generierte Ausgabedatei sal\_rpt61.txt.**

Öffnen Sie die übertragene Datei `sal_rpt61.txt` mithilfe des Befehls `gedit` oder mit einem Editor Ihrer Wahl. Der Bericht wird wie folgt angezeigt:

The terminal window title is 'oracle@EDRSR15P1:~/labs/plpu/reports'. The command history shows:
 

```
[oracle@EDRSR15P1 Desktop]$ cd /home/oracle/labs/plpu/reports
[oracle@EDRSR15P1 reports]$ pwd
/home/oracle/labs/plpu/reports
[oracle@EDRSR15P1 reports]$ ls
instructor.txt salreport2.txt sal_rpt61.txt
[oracle@EDRSR15P1 reports]$ gedit sal_rpt61.txt
```

```
sal_rpt61.txt X
Employees who earn more than average salary:
REPORT GENERATED ON 19-NOV-12

Hartstein          20   $13,000.00
Raphaely          30   $11,000.00
Ladwig             50   $3,600.00
Rajs               50   $3,500.00
Sarchand            50   $4,200.00
Bull                50   $4,100.00
Chung               50   $3,800.00
Dilly               50   $3,600.00
Bell                50   $4,000.00
Everett              50   $3,900.00
Mourgos            50   $5,800.00
Vollman             50   $6,500.00
Kaufling            50   $7,900.00
Fripp                50   $8,200.00
Weiss               50   $8,000.00
Hunold               60   $9,000.00
Ernst                60   $6,000.00
Russell              80   $14,000.00
Partners             80   $13,500.00
Errazuriz            80   $12,000.00
Cambrault            80   $11,000.00
Zlotkey              80   $10,500.00
Tucker               80   $10,000.00
Bernstein            80   $9,500.00
Hall                 80   $9,000.00
King                 80   $10,000.00
Sully                80   $9,500.00
McEwen               80   $9,000.00
Vishney              80   $10,500.00
Greene               80   $9,500.00
Ozer                 80   $11,500.00
Bloom                80   $10,000.00
Fox                  80   $9,600.00
Abel                 80   $11,000.00
Hutton               80   $8,800.00
Taylor               80   $10,406.00
King                 90   $24,000.00
Faviet               100   $9,000.00
Greenberg            100   $12,008.00
Higgins              110   $12,008.00

*** END OF REPORT ***
```

**Hinweis:** Die Ausgabe kann je nach den Daten in der Mitarbeitertabelle leicht abweichen.



# **Übungen zu Lektion 7 – Dynamisches SQL**

## **Kapitel 7**

# Übungen zu Lektion 7 – Überblick

---

## Lektionsüberblick

In dieser Übung erstellen Sie ein Package, das mithilfe von nativem dynamischem SQL eine Tabelle erstellt oder löscht und Zeilen in der Tabelle füllt, ändert und löscht. Außerdem erstellen Sie ein Package, das den PL/SQL-Code in Ihrem Schema kompiliert – entweder den gesamten PL/SQL-Code oder nur Code, der in der Tabelle `USER_OBJECTS` den Status `INVALID` hat.

### Hinweis:

1. Führen Sie vor Beginn dieser Übung das Skript  
`/home/oracle/plpu/code_ex/code_ex_scripts/clean_up_scripts/`  
`cleanup_07.sql` aus.
2. Wenn Sie einen Schritt in einer Übung ausgelassen haben, führen Sie das entsprechende Lösungsskript zu diesem Übungsschritt aus, bevor Sie mit dem nächsten Schritt oder der nächsten Übung fortfahren.

# Übung 1 zu Lektion 7 – Natives dynamisches SQL

---

## Überblick

In dieser Übung erstellen Sie ein Package, das mithilfe von nativem dynamischem SQL eine Tabelle erstellt oder löscht und Zeilen in der Tabelle füllt, ändert und löscht. Außerdem erstellen Sie ein Package, das den PL/SQL-Code in Ihrem Schema kompiliert – entweder den gesamten PL/SQL-Code oder nur Code, der in der Tabelle `USER_OBJECTS` den Status `INVALID` hat.

**Hinweis:** Führen Sie vor den folgenden Aufgaben das Skript `cleanup_07.sql` in `/home/oracle/plpu/code_ex/code_ex_scripts/clean_up_scripts/` aus.

## Aufgabe

- Erstellen Sie ein Package namens `TABLE_PKG`, das mithilfe von nativem dynamischem SQL eine Tabelle erstellt oder löscht und Zeilen in der Tabelle füllt, ändert und löscht. Stellen Sie sicher, dass die Unterprogramme optionale Standardparameter mit `NULL`-Werten verarbeiten.

- Erstellen Sie eine Packagespezifikation mit folgenden Prozeduren:

```
PROCEDURE make(p_table_name VARCHAR2, p_col_specs VARCHAR2)
PROCEDURE add_row(p_table_name VARCHAR2, p_col_values
                  VARCHAR2, p_cols VARCHAR2 := NULL)
PROCEDURE upd_row(p_table_name VARCHAR2, p_set_values
                  VARCHAR2, p_conditions VARCHAR2 := NULL)
PROCEDURE del_row(p_table_name VARCHAR2,
                  p_conditions VARCHAR2 := NULL);
PROCEDURE remove(p_table_name VARCHAR2)
```

- Erstellen Sie den Packagebody, der die Parameter annimmt und die entsprechenden SQL-Anweisungen dynamisch erstellt, die mithilfe von nativem dynamischem SQL ausgeführt werden, ausgenommen die Prozedur `remove`. Diese Prozedur sollte mit dem Package `DBMS_SQL` erstellt werden.

- Führen Sie die Packageprozedur `MAKE` aus, um wie folgt eine Tabelle zu erstellen:

```
make('my_contacts', 'id number(4), name varchar2(40)');
```

- Beschreiben Sie die Tabellenstruktur von `MY_CONTACTS`.

- Führen Sie die Packageprozedur `ADD_ROW` aus, um folgende Zeilen einzufügen.

Aktivieren Sie `SERVERTOUTPUT`.

```
add_row('my_contacts', '1, ''Lauran Serhal'', 'id, name');
add_row('my_contacts', '2, ''Nancy'', 'id, name');
add_row('my_contacts', '3, ''Sunitha Patel'', 'id, name');
add_row('my_contacts', '4, ''Valli Pataballa'', 'id, name');
```

- Fragen Sie den Tabelleninhalt von `MY_CONTACTS` ab, um zu prüfen, ob die Zeilen eingefügt wurden.

- Führen Sie die Packageprozedur `DEL_ROW` aus, um die Kontaktperson mit dem ID-Wert 3 zu löschen.

- Führen Sie die Prozedur `UPD_ROW` mit folgenden Zeilendaten aus:

```
upd_row('my_contacts', 'name= ''Nancy Greenberg'', 'id=2');
```

- i. Fragen Sie den Tabelleninhalt von `MY_CONTACTS` ab, um zu prüfen, ob die Änderungen vorgenommen wurden.
  - j. Löschen Sie die Tabelle mit der Prozedur `remove`, und beschreiben Sie die Tabelle `MY_CONTACTS`.
2. Erstellen Sie ein Package namens `COMPILE_PKG`, das den PL/SQL-Code in Ihrem Schema kompiliert.
- a. Erstellen Sie in der Spezifikation eine Packageprozedur namens `MAKE`, die den Namen einer zu kompilierenden PL/SQL-Programmeinheit annimmt.
  - b. Nehmen Sie Folgendes in den Packagebody auf:
    - 1) Die Prozedur `EXECUTE`, die im 1. Schritt dieser Übung in der Prozedur `TABLE_PKG` verwendet wurde
    - 2) Eine private Funktion namens `GET_TYPE`, um den PL/SQL-Objekttyp aus dem Data Dictionary zu ermitteln
      - Die Funktion gibt den Typnamen zurück (verwenden Sie `PACKAGE` für ein Package mit Body), sofern das Objekt vorhanden ist. Andernfalls wird `NULL` zurückgegeben.
      - Fügen Sie in der Klauselbedingung `WHERE` Folgendes ein, um sicherzustellen, dass nur eine Zeile zurückgegeben wird, wenn der Name einem `PACKAGE` entspricht, das auch einen `PACKAGE BODY` haben kann. In diesem Fall können Sie nur das vollständige Package kompilieren, nicht aber die Spezifikation oder den Body als separate Komponenten:

```
rownum = 1
```
  - 3) Erstellen Sie die Prozedur `MAKE` wie folgt:
    - Die Prozedur `MAKE` nimmt ein Argument an (`name`), das den Objektnamen darstellt.
    - Die Prozedur `MAKE` ruft die Funktion `GET_TYPE` auf. Wenn das Objekt vorhanden ist, wird es von `MAKE` mithilfe der Anweisung `ALTER` dynamisch kompiliert.
  - c. Kompilieren Sie mit der Prozedur `COMPILE_PKG.MAKE` Folgendes:
    - 1) Die Prozedur `EMPLOYEE_REPORT`
    - 2) Das Package `EMP_PKG`
    - 3) Ein nicht vorhandenes Objekt namens `EMP_DATA`

## Übung 1 zu Lektion 7 – Lösung: Natives dynamisches SQL

In dieser Übung erstellen Sie ein Package, das mithilfe von nativem dynamischem SQL eine Tabelle erstellt oder löscht und Zeilen in der Tabelle füllt, ändert und löscht. Außerdem erstellen Sie ein Package, das den PL/SQL-Code in Ihrem Schema kompiliert – entweder den gesamten PL/SQL-Code oder nur Code, der in der Tabelle `USER_OBJECTS` den Status `INVALID` hat.

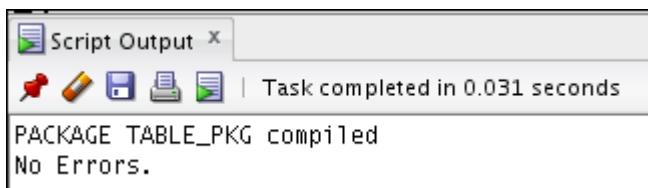
1. Erstellen Sie ein Package namens `TABLE_PKG`, das mithilfe von nativem dynamischem SQL eine Tabelle erstellt oder löscht und Zeilen in der Tabelle füllt, ändert und löscht. Stellen Sie sicher, dass die Unterprogramme optionale Standardparameter mit NULL-Werten verarbeiten.

- a. Erstellen Sie eine Packagespezifikation mit folgenden Prozeduren:

```
PROCEDURE make(p_table_name VARCHAR2, p_col_specs VARCHAR2)
PROCEDURE add_row(p_table_name VARCHAR2, p_col_values
                  VARCHAR2, p_cols VARCHAR2 := NULL)
PROCEDURE upd_row(p_table_name VARCHAR2, p_set_values
                  VARCHAR2, p_conditions VARCHAR2 := NULL)
PROCEDURE del_row(p_table_name VARCHAR2,
                  p_conditions VARCHAR2 := NULL);
PROCEDURE remove(p_table_name VARCHAR2)
```

**Öffnen Sie das Skript `/home/oracle/labs/plpu/solns/sol_07.sql`. Entfernen Sie die Kommentarzeichen, und wählen Sie den Code unter der Aufgabe 1\_a. Um die Packagespezifikation zu erstellen und zu kompilieren, klicken Sie in der SQL Worksheet-Symbolleiste auf das Symbol "Run Script" (F5). Der Code und das Ergebnis werden wie folgt angezeigt:**

```
CREATE OR REPLACE PACKAGE table_pkg IS
    PROCEDURE make(p_table_name VARCHAR2, p_col_specs
                   VARCHAR2);
    PROCEDURE add_row(p_table_name VARCHAR2, p_col_values
                      VARCHAR2, p_cols VARCHAR2 := NULL);
    PROCEDURE upd_row(p_table_name VARCHAR2, p_set_values
                      VARCHAR2, p_conditions VARCHAR2 := NULL);
    PROCEDURE del_row(p_table_name VARCHAR2, p_conditions
                      VARCHAR2 := NULL);
    PROCEDURE remove(p_table_name VARCHAR2);
END table_pkg;
/
SHOW ERRORS
```



- b. Erstellen Sie den Packagebody, der die Parameter annimmt und die entsprechenden SQL-Anweisungen dynamisch erstellt, die mithilfe von nativem dynamischem SQL ausgeführt werden. Davon ausgenommen ist die Prozedur `remove` die mit dem Package DBMS\_SQL erstellt werden sollte.

**Entfernen Sie die Kommentarzeichen, und wählen Sie den Code unter der Aufgabe 1\_b. Um die Packagespezifikation zu erstellen und zu kompilieren, klicken Sie in der SQL Worksheet-Symbolleiste auf das Symbol "Run Script" (F5). Der Code und das Ergebnis sind unten abgebildet.**

```

CREATE OR REPLACE PACKAGE BODY table_pkg IS
    PROCEDURE execute(p_stmt VARCHAR2) IS
        BEGIN
            DBMS_OUTPUT.PUT_LINE(p_stmt);
            EXECUTE IMMEDIATE p_stmt;
        END;

    PROCEDURE make(p_table_name VARCHAR2, p_col_specs VARCHAR2)
    IS
        v_stmt VARCHAR2(200) := 'CREATE TABLE '|| p_table_name ||
                               ' (' || p_col_specs || ')';
    BEGIN
        execute(v_stmt);
    END;

    PROCEDURE add_row(p_table_name VARCHAR2, p_col_values
                      VARCHAR2, p_cols VARCHAR2 := NULL) IS
        v_stmt VARCHAR2(200) := 'INSERT INTO '|| p_table_name;
    BEGIN
        IF p_cols IS NOT NULL THEN
            v_stmt := v_stmt || ' (' || p_cols || ')';
        END IF;
        v_stmt := v_stmt || ' VALUES (' || p_col_values || ')';
        execute(v_stmt);
    END;

    PROCEDURE upd_row(p_table_name VARCHAR2, p_set_values
                      VARCHAR2, p_conditions VARCHAR2 := NULL) IS
        v_stmt VARCHAR2(200) := 'UPDATE '|| p_table_name || ' SET ' ||
                               p_set_values;
    BEGIN
        IF p_conditions IS NOT NULL THEN

```

```

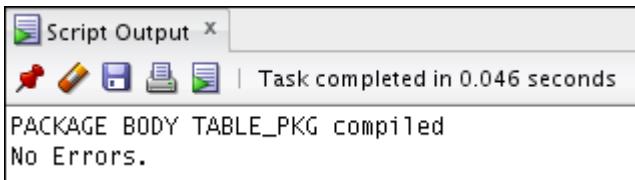
        v_stmt := v_stmt || ' WHERE ' || p_conditions;
    END IF;
    execute(v_stmt);
END;

PROCEDURE del_row(p_table_name VARCHAR2, p_conditions
                  VARCHAR2 := NULL) IS
    v_stmt VARCHAR2(200) := 'DELETE FROM '|| p_table_name;
BEGIN
    IF p_conditions IS NOT NULL THEN
        v_stmt := v_stmt || ' WHERE ' || p_conditions;
    END IF;
    execute(v_stmt);
END;

PROCEDURE remove(p_table_name VARCHAR2) IS
    cur_id INTEGER;
    v_stmt VARCHAR2(100) := 'DROP TABLE '||p_table_name;
BEGIN
    cur_id := DBMS_SQL.OPEN_CURSOR;
    DBMS_OUTPUT.PUT_LINE(v_stmt);
    DBMS_SQLPARSE(cur_id, v_stmt, DBMS_SQL.NATIVE);
    -- Parse executes DDL statements,no EXECUTE is required.
    DBMS_SQLCLOSE_CURSOR(cur_id);
END;

END table_pkg;
/
SHOW ERRORS

```



- c. Führen Sie die Packageprozedur `MAKE` aus, um wie folgt eine Tabelle zu erstellen:
- ```
make('my_contacts', 'id number(4), name varchar2(40)');
```

**Entfernen Sie die Kommentarzeichen, und wählen Sie den Code unter der Aufgabe 1\_c. Um die Packagespezifikation zu erstellen, klicken Sie in der SQL Worksheet-Symbolleiste auf das Symbol "Run Script" (F5). Der Code und die Ergebnisse werden wie folgt angezeigt:**

```
EXECUTE table_pkg.make('my_contacts', 'id number(4), name
varchar2(40)');
```

```
Script Output X
| Task completed in 0.993 seconds
anonymous block completed
CREATE TABLE my_contacts (id number(4), name varchar2(40))
```

- d. Beschreiben Sie die Tabellenstruktur von `MY_CONTACTS`.

```
DESCRIBE my_contacts
```

**Das Ergebnis wird wie folgt angezeigt:**

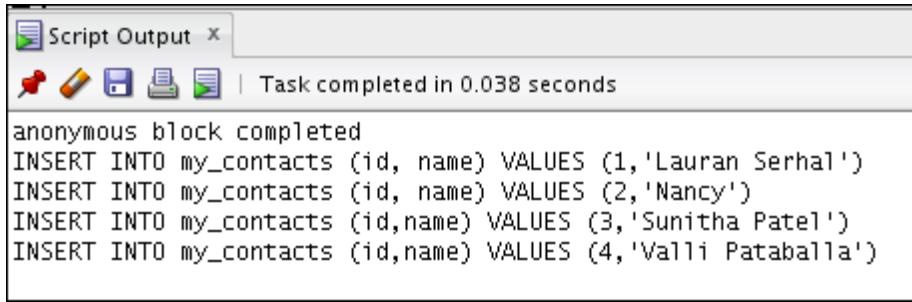
```
Script Output X
| Task completed in 0.529 seconds
DESCRIBE my_contacts
Name Null Type
-----
ID      NUMBER(4)
NAME    VARCHAR2(40)
```

- e. Führen Sie die Packageprozedur `ADD_ROW` aus, um folgende Zeilen einzufügen.

```
SET SERVEROUTPUT ON
```

```
BEGIN
    table_pkg.add_row('my_contacts', '1', ''Lauran Serhal'', 'id,
                      name');
    table_pkg.add_row('my_contacts', '2', ''Nancy'', 'id, name');
    table_pkg.add_row('my_contacts', '3', ''Sunita
                      Patel'', 'id, name');
    table_pkg.add_row('my_contacts', '4', ''Valli
                      Pataballa'', 'id, name');
END;
/
```

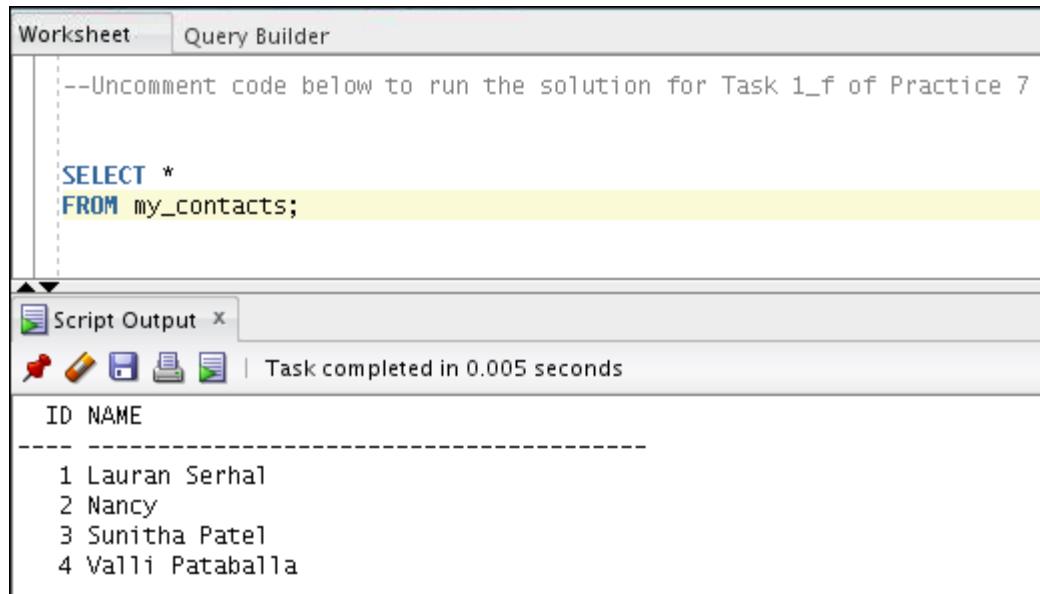
Entfernen Sie die Kommentarzeichen, und wählen Sie den Code unter der Aufgabe 1\_e. Um das Skript auszuführen, klicken Sie in der SQL Worksheet-Symbolleiste auf das Symbol "Run Script" (F5).



```
anonymous block completed
INSERT INTO my_contacts (id, name) VALUES (1,'Lauran Serhal')
INSERT INTO my_contacts (id, name) VALUES (2,'Nancy')
INSERT INTO my_contacts (id,name) VALUES (3,'Sunitha Patel')
INSERT INTO my_contacts (id,name) VALUES (4,'Valli Pataballa')
```

- f. Fragen Sie den Tabelleninhalt von MY\_CONTACTS ab, um zu prüfen, ob die Zeilen eingefügt wurden.

**Der Code und die Ergebnisse werden wie folgt angezeigt:**



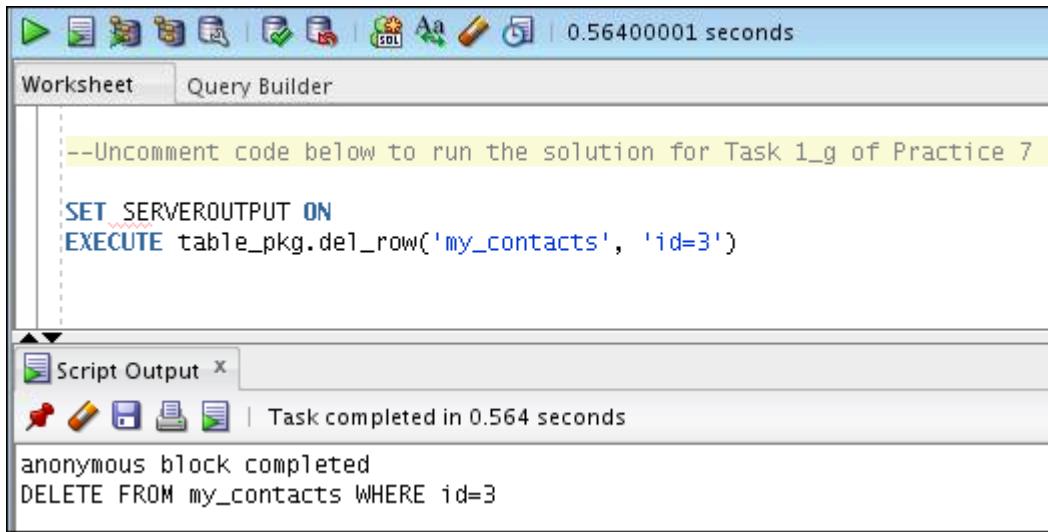
```
--Uncomment code below to run the solution for Task 1_f of Practice 7

SELECT *
FROM my_contacts;
```

| ID | NAME            |
|----|-----------------|
| 1  | Lauran Serhal   |
| 2  | Nancy           |
| 3  | Sunitha Patel   |
| 4  | Valli Pataballa |

- g. Führen Sie die Packageprozedur `DEL_ROW` aus, um die Kontaktperson mit dem ID-Wert 3 zu löschen.

**Der Code und die Ergebnisse werden wie folgt angezeigt:**



The screenshot shows the Oracle SQL Developer interface. The top toolbar has various icons and the text "0.56400001 seconds". Below it is a tab bar with "Worksheet" and "Query Builder", with "Worksheet" selected. The main workspace contains the following code:

```
--Uncomment code below to run the solution for Task 1_g of Practice 7
SET SERVEROUTPUT ON
EXECUTE table_pkg.del_row('my_contacts', 'id=3')
```

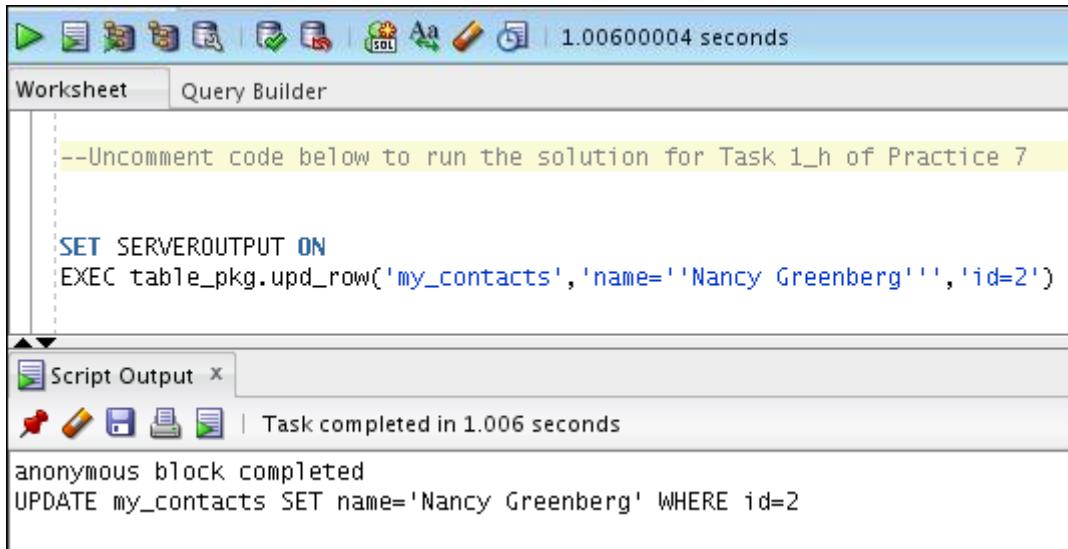
Below the workspace is a "Script Output" window with the following log:

```
anonymous block completed
DELETE FROM my_contacts WHERE id=3
```

- h. Führen Sie die Prozedur `UPD_ROW` mit folgenden Zeilendaten aus:

`upd_row('my_contacts', 'name='Nancy Greenberg''', 'id=2');`

**Der Code und die Ergebnisse werden wie folgt angezeigt:**



The screenshot shows the Oracle SQL Developer interface. The top toolbar has various icons and the text "1.00600004 seconds". Below it is a tab bar with "Worksheet" and "Query Builder", with "Worksheet" selected. The main workspace contains the following code:

```
--Uncomment code below to run the solution for Task 1_h of Practice 7
SET SERVEROUTPUT ON
EXEC table_pkg.upd_row('my_contacts', 'name='Nancy Greenberg''', 'id=2')
```

Below the workspace is a "Script Output" window with the following log:

```
anonymous block completed
UPDATE my_contacts SET name='Nancy Greenberg' WHERE id=2
```

- i. Fragen Sie den Tabelleninhalt von MY\_CONTACTS ab, um zu prüfen, ob die Änderungen vorgenommen wurden.

**Der Code und die Ergebnisse werden wie folgt angezeigt:**

The screenshot shows the Oracle SQL Developer interface. In the top navigation bar, the 'Worksheet' tab is selected. Below it, the 'Script Output' tab is also visible. The main workspace contains a SQL query:

```
--Uncomment code below to run the solution for Task 1_i of Practice 7
SELECT *
FROM my_contacts;
```

Below the query, the 'Script Output' tab displays the results of the execution:

```
Task completed in 0.002 seconds
```

| ID | NAME            |
|----|-----------------|
| 1  | Lauran Serhal   |
| 2  | Nancy Greenberg |
| 4  | Valli Pataballa |

- j. Löschen Sie die Tabelle mit der Prozedur `remove`, und beschreiben Sie die Tabelle MY\_CONTACTS.

**Der Code und die Ergebnisse werden wie folgt angezeigt:**

The screenshot shows the Oracle SQL Developer interface. The 'Worksheet' tab is selected. The workspace contains the following code:

```
--Uncomment code below to run the solution for Task 1_j of Practice 7
EXECUTE table_pkg.remove('my_contacts')
DESCRIBE my_contacts
```

Below the code, the 'Script Output' tab shows the execution results:

```
anonymous block completed
DROP TABLE my_contacts

DESCRIBE my_contacts
ERROR:
-----
```

At the bottom of the output, there is an additional error message:

```
ERROR: object MY_CONTACTS does not exist
```

2. Erstellen Sie ein Package namens `COMPILE_PKG`, das den PL/SQL-Code in Ihrem Schema kompiliert.
- Erstellen Sie in der Spezifikation eine Packageprozedur namens `MAKE`, die den Namen einer zu kompilierenden PL/SQL-Programmeinheit annimmt.
- Entfernen Sie die Kommentarzeichen, und wählen Sie den Code unter der Aufgabe 2\_a. Um die Packagespezifikation zu erstellen und zu kompilieren, klicken Sie in der SQL Worksheet-Symbolleiste auf das Symbol "Run Script" (F5). Der Code und die Ergebnisse sind unten abgebildet.**

```
CREATE OR REPLACE PACKAGE compile_pkg IS
  PROCEDURE make(p_name VARCHAR2);
END compile_pkg;
/
SHOW ERRORS
```

The screenshot shows the Oracle SQL Worksheet interface. The top bar includes icons for file operations, a search function, and a toolbar with various symbols. The status bar indicates "0.041 seconds". The main area is titled "Worksheet" and contains the PL/SQL code for creating the package. Below the code is a "Script Output" window showing the results of the compilation: "PACKAGE COMPILE\_PKG compiled" and "No Errors.".

```
--Uncomment code below to run the solution for Task 2_a of Practice 7
CREATE OR REPLACE PACKAGE compile_pkg IS
  PROCEDURE make(p_name VARCHAR2);
END compile_pkg;
/
SHOW ERRORS
```

Script Output

PACKAGE COMPILE\_PKG compiled  
No Errors.

- Nehmen Sie Folgendes in den Packagebody auf:
  - Die Prozedur `EXECUTE`, die im 1. Schritt dieser Übung in der Prozedur `TABLE_PKG` verwendet wurde
  - Eine private Funktion namens `GET_TYPE`, um den PL/SQL-Objekttyp aus dem Data Dictionary zu ermitteln
    - Die Funktion gibt den Typnamen zurück (verwenden Sie `PACKAGE` für ein Package mit Body), sofern das Objekt vorhanden ist. Andernfalls wird `NULL` zurückgegeben.
    - Nehmen Sie den folgenden Eintrag in die Klauselbedingung `WHERE` auf, um sicherzustellen, dass nur eine Zeile zurückgegeben wird, wenn der Name einem `PACKAGE` entspricht, das ebenfalls einen `PACKAGE BODY` enthalten kann. In diesem Fall können Sie nur das vollständige Package kompilieren, nicht aber die Spezifikation oder den Body als separate Komponenten:

```
rownum = 1
```

3) Erstellen Sie die Prozedur `MAKE` wie folgt:

- Die Prozedur `MAKE` nimmt ein Argument an (`name`), das den Objektnamen darstellt.
- Die Prozedur `MAKE` ruft die Funktion `GET_TYPE` auf. Wenn das Objekt vorhanden ist, wird es von `MAKE` mithilfe der Anweisung `ALTER` dynamisch kompiliert.

**Entfernen Sie die Kommentarzeichen, und wählen Sie den Code unter der Aufgabe 2\_b. Um den Packagebody zu erstellen und zu kompilieren, klicken Sie in der SQL Worksheet-Symbolleiste auf das Symbol "Run Script" (F5). Der Code und die Ergebnisse werden wie folgt angezeigt:**

```
CREATE OR REPLACE PACKAGE BODY compile_pkg IS

PROCEDURE execute(p_stmt VARCHAR2) IS
BEGIN
    DBMS_OUTPUT.PUT_LINE(p_stmt);
    EXECUTE IMMEDIATE p_stmt;
END;

FUNCTION get_type(p_name VARCHAR2) RETURN VARCHAR2 IS
    v_proc_type VARCHAR2(30) := NULL;
BEGIN

    -- The ROWNUM = 1 is added to the condition
    -- to ensure only one row is returned if the
    -- name represents a PACKAGE, which may also
    -- have a PACKAGE BODY. In this case, we can
    -- only compile the complete package, but not
    -- the specification or body as separate
    -- components.

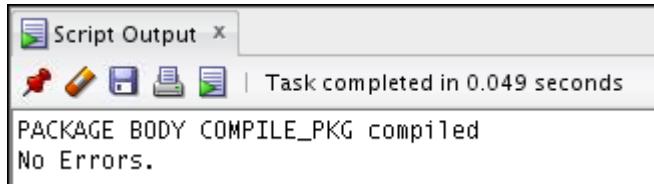
    SELECT object_type INTO v_proc_type
    FROM user_objects
    WHERE object_name = UPPER(p_name)
    AND ROWNUM = 1;
    RETURN v_proc_type;
EXCEPTION
    WHEN NO_DATA_FOUND THEN
        RETURN NULL;
END;

PROCEDURE make(p_name VARCHAR2) IS
    v_stmt          VARCHAR2(100);
    v_proc_type    VARCHAR2(30) := get_type(p_name);
```

```

BEGIN
    IF v_proc_type IS NOT NULL THEN
        v_stmt := 'ALTER ' || v_proc_type || ' ' || p_name || '
COMPILE';
        execute(v_stmt);
    ELSE
        RAISE_APPLICATION_ERROR(-20001,
            'Subprogram ' || p_name || ' does not exist');
    END IF;
END make;
END compile_pkg;
/
SHOW ERRORS

```



c. Kompilieren Sie mit der Prozedur `COMPILE_PKG.MAKE` Folgendes:

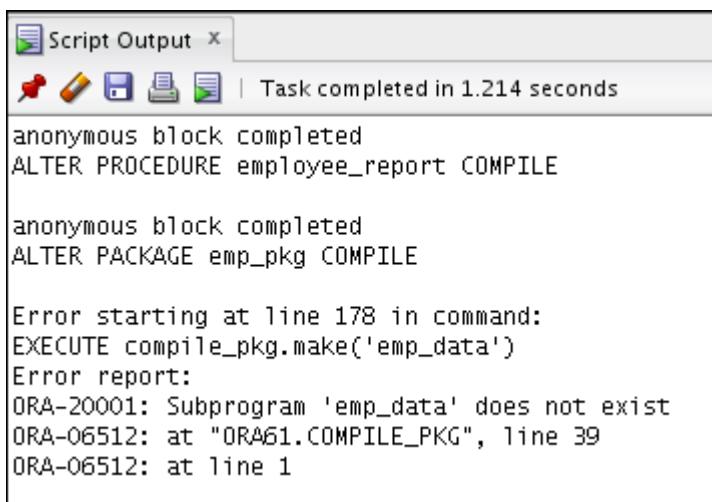
- 1) Die Prozedur `EMPLOYEE_REPORT`
- 2) Das Package `EMP_PKG`
- 3) Ein nicht vorhandenes Objekt namens `EMP_DATA`

**Entfernen Sie die Kommentarzeichen, und wählen Sie den Code unter der Aufgabe 2\_c. Um die Packageprozedur aufzurufen, klicken Sie in der SQL Worksheet-Symbolleiste auf das Symbol "Run Script" (F5). Das Ergebnis ist unten abgebildet.**

```

SET SERVEROUTPUT ON
EXECUTE compile_pkg.make('employee_report')
EXECUTE compile_pkg.make('emp_pkg')
EXECUTE compile_pkg.make('emp_data')

```



# **Übungen zu Lektion 8 – Überlegungen zum Design von PL/SQL-Code**

## **Kapitel 8**

# Übungen zu Lektion 8 – Überblick

---

## Lektionsüberblick

In dieser Übung erstellen Sie ein Package, das einen Bulk-Abruf der Mitarbeiter aus einer bestimmten Abteilung durchführt. Die Daten werden im Package in einer PL/SQL-Tabelle gespeichert. Außerdem stellen Sie eine Prozedur zum Anzeigen des Tabelleninhalts bereit. Darüber hinaus erstellen Sie die Prozedur `add_employee`, mit der neue Mitarbeiter eingefügt werden. Diese Prozedur schreibt mit einem lokalen, autonomen Unterprogramm bei jedem Aufruf der Prozedur `add_employee` einen Log-Record, unabhängig davon, ob ein Record erfolgreich hinzugefügt wurde oder nicht.

### Hinweis:

1. Führen Sie vor Beginn dieser Übung das Skript  
`/home/oracle/plpu/code_ex/code_ex_scripts/clean_up_scripts/cleanup_08.sql` aus.
2. Wenn Sie einen Schritt in einer Übung ausgelassen haben, führen Sie das entsprechende Lösungsskript zu diesem Übungsschritt aus, bevor Sie mit dem nächsten Schritt oder der nächsten Übung fortfahren.

# Übung 1 zu Lektion 8 – Bulk Binding und autonome Transaktionen

---

## Überblick

In dieser Übung erstellen Sie ein Package, das einen Bulk-Abruf der Mitarbeiter aus einer bestimmten Abteilung durchführt. Die Daten werden im Package in einer PL/SQL-Tabelle gespeichert. Außerdem stellen Sie eine Prozedur zum Anzeigen des Tabelleninhalts bereit. Darüber hinaus erstellen Sie die Prozedur `add_employee`, mit der neue Mitarbeiter eingefügt werden. Diese Prozedur schreibt mit einem lokalen, autonomen Unterprogramm bei jedem Aufruf der Prozedur `add_employee` einen Log-Record, unabhängig davon, ob ein Record erfolgreich hinzugefügt wurde oder nicht.

**Hinweis:** Führen Sie vor den folgenden Aufgaben das Skript `cleanup_08.sql` in `/home/oracle/plpu/code_ex/code_ex_scripts/clean_up_scripts/` aus.

## Aufgabe

1. Aktualisieren Sie das Package `EMP_PKG` mit einer neuen Prozedur, um die Mitarbeiter aus einer bestimmten Abteilung abzufragen.
  - a. Gehen Sie in der Packagespezifikation wie folgt vor:
    - 1) Deklarieren Sie eine `get_employees`-Prozedur mit einem Parameter namens `dept_id`, der auf dem Spaltentyp `employees.department_id` basiert.
    - 2) Definieren Sie einen verschachtelten PL/SQL-Typ als `TABLE OF EMPLOYEES%ROWTYPE`.
  - b. Gehen Sie im Packagebody wie folgt vor:
    - 1) Definieren Sie eine private Variable namens `emp_table` auf Basis des in der Spezifikation für Mitarbeiter-Records definierten Typs.
    - 2) Implementieren Sie die Prozedur `get_employees`, um die Daten als Ganzes mit einem einzigen Vorgang in die Tabelle abzurufen.
  - c. Erstellen Sie in der Spezifikation und im Body eine neue Prozedur namens `show_employees`, die keine Argumente annimmt. Diese Prozedur zeigt den Inhalt der privaten PL/SQL-Tabellenvariablen an (sofern Daten vorhanden sind). Verwenden Sie die in einer der vorherigen Übungen erstellte Prozedur `print_employee`. Um die Ergebnisse anzuzeigen, klicken Sie in SQL Developer in der Registerkarte **DBMS Output** auf das Symbol **Enable DBMS Output**, sofern Sie dies nicht bereits getan haben.
  - d. Aktivieren Sie `SERVEROUTPUT`. Rufen Sie die Prozedur `emp_pkg.get_employees` für Abteilung 30 und anschließend `emp_pkg.show_employees` auf. Wiederholen Sie den Vorgang für Abteilung 60.
2. Ihr Chef möchte jeden Aufruf der Prozedur `add_employee` im Package protokollieren, bei dem ein neuer Mitarbeiter in die Tabelle `EMPLOYEES` eingefügt wird.
  - a. Laden und führen Sie zuerst den Code unter der Aufgabe 2\_a im Skript `/home/oracle/labs/plpu/solns/sol_08.sql` aus, um eine Logtabelle namens `LOG_NEWEMP` und eine Sequence namens `log_newemp_seq` zu erstellen.

- b. Ändern Sie im Body des Packages `EMP_PKG` die Prozedur `add_employee`, die den tatsächlichen `INSERT`-Vorgang ausführt. Fügen Sie wie folgt eine lokale Prozedur namens `audit_newemp` hinzu:
- 1) Die Prozedur `audit_newemp` muss mithilfe einer autonomen Transaktion einen Log-Record in die Tabelle `LOG_NEWEMP` einfügen.
  - 2) Speichern Sie den Benutzer (`USER`), die aktuelle Uhrzeit und den Namen des neuen Mitarbeiters in der Zeile der Logtabelle.
  - 3) Legen Sie die Spalte `entry_id` mit `log_newemp_seq` fest.
- Hinweis:** Denken Sie daran, in einer Prozedur mit einer autonomen Transaktion einen `COMMIT`-Vorgang auszuführen.
- c. Ändern Sie die Prozedur `add_employee`, um `audit_emp` aufzurufen, bevor der `INSERT`-Vorgang ausgeführt wird.
- d. Rufen Sie die Prozedur `add_employee` für die nachstehenden neuen Mitarbeiter auf:  
Max Smart in Abteilung 20 und Clark Kent in Abteilung 10. Was geschieht?
- e. Fragen Sie die beiden hinzugefügten `EMPLOYEES`-Records und die Records in der Tabelle `LOG_NEWEMP` ab. Wie viele Log-Records sind vorhanden?
- f. Führen Sie eine `ROLLBACK`-Anweisung aus, um die nicht festgeschriebenen `INSERT`-Vorgänge rückgängig zu machen. Verwenden Sie die Abfragen aus Schritt 2e wie folgt:
- 1) Prüfen Sie anhand der ersten Abfrage, ob die Zeilen für die Mitarbeiter `Smart` und `Kent` entfernt wurden.
  - 2) Prüfen Sie anhand der zweiten Abfrage die Log-Records in der Tabelle `LOG_NEWEMP`. Wie viele Log-Records sind vorhanden? Warum?

## Übung 1 zu Lektion 8 – Lösung: Bulk Binding und autonome Transaktionen

In dieser Übung erstellen Sie ein Package, das einen Bulk-Abruf der Mitarbeiter aus einer bestimmten Abteilung durchführt. Die Daten werden im Package in einer PL/SQL-Tabelle gespeichert. Außerdem stellen Sie eine Prozedur zum Anzeigen des Tabelleninhalts bereit. Darüber hinaus erstellen Sie die Prozedur `add_employee`, mit der neue Mitarbeiter eingefügt werden. Diese Prozedur schreibt mit einem lokalen, autonomen Unterprogramm bei jedem Aufruf der Prozedur `add_employee` einen Log-Record, unabhängig davon, ob ein Record erfolgreich hinzugefügt wurde oder nicht.

1. Aktualisieren Sie das Package `EMP_PKG` mit einer neuen Prozedur, um die Mitarbeiter aus einer bestimmten Abteilung abzufragen.
  - a. Gehen Sie in der Packagespezifikation wie folgt vor:
    - 1) Deklarieren Sie eine `get_employees`-Prozedur mit einem Parameter namens `dept_id`, der auf dem Spaltentyp `employees.department_id` basiert
    - 2) Definieren Sie einen verschachtelten PL/SQL-Typ als `TABLE OF EMPLOYEES%ROWTYPE`

**Öffnen Sie das Skript `/home/oracle/labs/plpu/solns/sol_08.sql`. Entfernen Sie die Kommentarzeichen, und wählen Sie den Code unter der Aufgabe 1\_a. Um die Spezifikation zu erstellen und zu kompilieren, klicken Sie in der SQL Worksheet-Symbolleiste auf das Symbol "Run Script" (F5). Der Code und die Ergebnisse werden wie folgt angezeigt. Der neu hinzugefügte Code ist im Codefeld unten durch Fetschrift hervorgehoben.**

```
CREATE OR REPLACE PACKAGE emp_pkg IS

  TYPE emp_tab_type IS TABLE OF employees%ROWTYPE;

  PROCEDURE add_employee(
    p_first_name employees.first_name%TYPE,
    p_last_name employees.last_name%TYPE,
    p_email employees.email%TYPE,
    p_job employees.job_id%TYPE DEFAULT 'SA_REP',
    p_mgr employees.manager_id%TYPE DEFAULT 145,
    p_sal employees.salary%TYPE DEFAULT 1000,
    p_comm employees.commission_pct%TYPE DEFAULT 0,
    p_deptid employees.department_id%TYPE DEFAULT 30);

  PROCEDURE add_employee(
    p_first_name employees.first_name%TYPE,
    p_last_name employees.last_name%TYPE,
    p_deptid employees.department_id%TYPE);
```

```

PROCEDURE get_employee(
    p_empid IN employees.employee_id%TYPE,
    p_sal OUT employees.salary%TYPE,
    p_job OUT employees.job_id%TYPE);

FUNCTION get_employee(p_emp_id employees.employee_id%type)
    return employees%rowtype;

FUNCTION get_employee(p_family_name employees.last_name%type)
    return employees%rowtype;

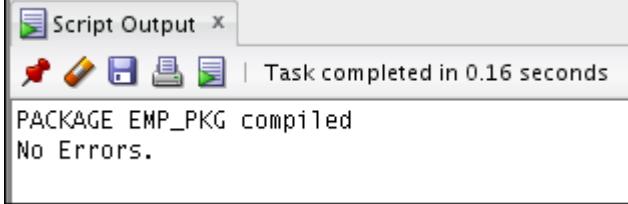
PROCEDURE get_employees(p_dept_id
employees.department_id%type);

PROCEDURE init_departments;

PROCEDURE print_employee(p_rec_emp employees%rowtype);

END emp_pkg;
/
SHOW ERRORS

```



- b. Gehen Sie im Packagebody wie folgt vor:
- 1) Definieren Sie eine private Variable namens `emp_table` auf Basis des in der Spezifikation für Mitarbeiter-Records definierten Typs.
  - 2) Implementieren Sie die Prozedur `get_employees`, um die Daten als Ganzes mit einem einzigen Vorgang in die Tabelle abzurufen.

**Entfernen Sie die Kommentarzeichen, und wählen Sie den Code unter der Aufgabe 1\_b. Um den Packagebody zu erstellen und zu kompilieren, klicken Sie in der SQL Worksheet-Symbolleiste auf das Symbol "Run Script" (F5). Der Code und die Ergebnisse sind unten abgebildet. Der neu hinzugefügte Code ist im Codefeld unten durch Fettschrift hervorgehoben.**

```

CREATE OR REPLACE PACKAGE BODY emp_pkg IS
    TYPE boolean_tab_type IS TABLE OF BOOLEAN
        INDEX BY BINARY_INTEGER;
    valid_departments boolean_tab_type;
    emp_table          emp_tab_type;

```

```

FUNCTION valid_deptid(p_deptid IN
departments.department_id%TYPE) RETURN BOOLEAN IS
    v_dummy PLS_INTEGER;
BEGIN
    RETURN valid_departments.exists(p_deptid);
EXCEPTION
    WHEN NO_DATA_FOUND THEN
        RETURN FALSE;
END valid_deptid;

PROCEDURE add_employee(
    p_first_name employees.first_name%TYPE,
    p_last_name employees.last_name%TYPE,
    p_email employees.email%TYPE,
    p_job employees.job_id%TYPE DEFAULT 'SA_REP',
    p_mgr employees.manager_id%TYPE DEFAULT 145,
    p_sal employees.salary%TYPE DEFAULT 1000,
    p_comm employees.commission_pct%TYPE DEFAULT 0,
    p_deptid employees.department_id%TYPE DEFAULT 30) IS
BEGIN
    IF valid_deptid(p_deptid) THEN

        INSERT INTO employees(employee_id, first_name, last_name,
email,
                job_id, manager_id, hire_date, salary, commission_pct,
department_id)
        VALUES (employees_seq.NEXTVAL, p_first_name, p_last_name,
p_email,
                p_job, p_mgr, TRUNC(SYSDATE), p_sal, p_comm, p_deptid);
    ELSE
        RAISE_APPLICATION_ERROR (-20204, 'Invalid department ID.
Try again.');
    END IF;
END add_employee;

PROCEDURE add_employee(
    p_first_name employees.first_name%TYPE,
    p_last_name employees.last_name%TYPE,
    p_deptid employees.department_id%TYPE) IS
    p_email employees.email%TYPE;

```

```

BEGIN
    p_email := UPPER(SUBSTR(p_first_name, 1,
1)||SUBSTR(p_last_name, 1, 7));
    add_employee(p_first_name, p_last_name, p_email, p_deptid =>
p_deptid);
END;

PROCEDURE get_employee(
    p.empid IN employees.employee_id%TYPE,
    p_sal OUT employees.salary%TYPE,
    p_job OUT employees.job_id%TYPE) IS
BEGIN
    SELECT salary, job_id
    INTO p_sal, p_job
    FROM employees
    WHERE employee_id = p.empid;
END get_employee;

FUNCTION get_employee(p.emp_id employees.employee_id%type)
    return employees%rowtype IS
    rec_emp employees%rowtype;
BEGIN
    SELECT * INTO rec_emp
    FROM employees
    WHERE employee_id = p.emp_id;
    RETURN rec_emp;
END;

FUNCTION get_employee(p.family_name employees.last_name%type)
    return employees%rowtype IS
    rec_emp employees%rowtype;
BEGIN
    SELECT * INTO rec_emp
    FROM employees
    WHERE last_name = p.family_name;
    RETURN rec_emp;
END;

-- New get_employees procedure.

PROCEDURE get_employees(p.dept_id employees.department_id%type)
IS

```

```

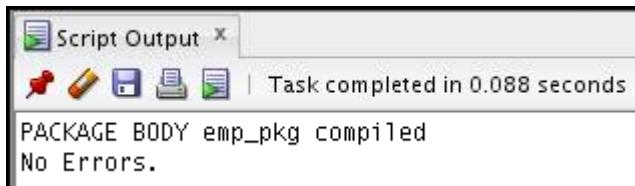
BEGIN
  SELECT * BULK COLLECT INTO emp_table
  FROM EMPLOYEES
  WHERE department_id = p_dept_id;
END;

PROCEDURE init_departments IS
BEGIN
  FOR rec IN (SELECT department_id FROM departments)
  LOOP
    valid_departments(rec.department_id) := TRUE;
  END LOOP;
END;

PROCEDURE print_employee(p_rec_emp employees%rowtype) IS
BEGIN
  DBMS_OUTPUT.PUT_LINE(p_rec_emp.department_id || ' ' ||
                       p_rec_emp.employee_id|| ' ' ||
                       p_rec_emp.first_name|| ' ' ||
                       p_rec_emp.last_name|| ' ' ||
                       p_rec_emp.job_id|| ' ' ||
                       p_rec_emp.salary);
END;

BEGIN
  init_departments;
END emp_pkg;
/
SHOW ERRORS

```



- c. Erstellen Sie in der Spezifikation und im Body eine neue Prozedur namens `show_employees`, die keine Argumente annimmt. Diese Prozedur zeigt den Inhalt der privaten PL/SQL-Tabellenvariablen an (sofern Daten vorhanden sind). Verwenden Sie die in einer der vorherigen Übungen erstellte Prozedur `print_employee`. Um die Ergebnisse anzuzeigen, klicken Sie in SQL Developer in der Registerkarte **DBMS Output** auf das Symbol **Enable DBMS Output**, sofern Sie dies nicht bereits getan haben.

**Entfernen Sie die Kommentarzeichen, und wählen Sie den Code unter der Aufgabe 1\_c. Um das Package mit der neuen Prozedur erneut zu erstellen und zu kompilieren, klicken Sie in der SQL Worksheet-Symbolleiste auf das Symbol "Run Script" (F5). Der Code und die Ergebnisse sind unten abgebildet.**

```
-- Package SPECIFICATION

CREATE OR REPLACE PACKAGE emp_pkg IS
    TYPE emp_tab_type IS TABLE OF employees%ROWTYPE;

    PROCEDURE add_employee(
        p_first_name employees.first_name%TYPE,
        p_last_name employees.last_name%TYPE,
        p_email employees.email%TYPE,
        p_job employees.job_id%TYPE DEFAULT 'SA_REP',
        p_mgr employees.manager_id%TYPE DEFAULT 145,
        p_sal employees.salary%TYPE DEFAULT 1000,
        p_comm employees.commission_pct%TYPE DEFAULT 0,
        p_deptid employees.department_id%TYPE DEFAULT 30);

    PROCEDURE add_employee(
        p_first_name employees.first_name%TYPE,
        p_last_name employees.last_name%TYPE,
        p_deptid employees.department_id%TYPE);

    PROCEDURE get_employee(
        p.empid IN employees.employee_id%TYPE,
        p_sal OUT employees.salary%TYPE,
        p_job OUT employees.job_id%TYPE);

    FUNCTION get_employee(p_emp_id employees.employee_id%type)
        return employees%rowtype;

    FUNCTION get_employee(p_family_name employees.last_name%type)
        return employees%rowtype;
```

```

    PROCEDURE get_employees(p_dept_id
employees.department_id%type);

    PROCEDURE init_departments;

    PROCEDURE print_employee(p_rec_emp employees%rowtype);

PROCEDURE show_employees;

END emp_pkg;
/
SHOW ERRORS

-- Package BODY

CREATE OR REPLACE PACKAGE BODY emp_pkg IS
    TYPE boolean_tab_type IS TABLE OF BOOLEAN
        INDEX BY BINARY_INTEGER;

    valid_departments boolean_tab_type;
    emp_table          emp_tab_type;
    FUNCTION valid_deptid(p_deptid IN
departments.department_id%TYPE)
        RETURN BOOLEAN;

    PROCEDURE add_employee(
        p_first_name employees.first_name%TYPE,
        p_last_name employees.last_name%TYPE,
        p_email      employees.email%TYPE,
        p_job        employees.job_id%TYPE DEFAULT 'SA_REP',
        p_mgr        employees.manager_id%TYPE DEFAULT 145,
        p_sal        employees.salary%TYPE DEFAULT 1000,
        p_comm       employees.commission_pct%TYPE DEFAULT 0,
        p_deptid     employees.department_id%TYPE DEFAULT 30) IS
BEGIN
    IF valid_deptid(p_deptid) THEN
        INSERT INTO employees(employee_id, first_name, last_name,
email,
                job_id, manager_id, hire_date, salary, commission_pct,
department_id)
        VALUES (employees_seq.NEXTVAL, p_first_name, p_last_name,
p_email,
                p_job, p_mgr, TRUNC(SYSDATE), p_sal, p_comm, p_deptid);
    END IF;
END;

```

```

    ELSE
        RAISE_APPLICATION_ERROR (-20204, 'Invalid department ID.
Try again.');
    END IF;
END add_employee;

PROCEDURE add_employee(
    p_first_name employees.first_name%TYPE,
    p_last_name employees.last_name%TYPE,
    p_deptid employees.department_id%TYPE) IS
    p_email employees.email%type;
BEGIN
    p_email := UPPER(SUBSTR(p_first_name, 1,
1)||SUBSTR(p_last_name, 1, 7));
    add_employee(p_first_name, p_last_name, p_email, p_deptid =>
p_deptid);
END;

PROCEDURE get_employee(
    p.empid IN employees.employee_id%TYPE,
    p_sal OUT employees.salary%TYPE,
    p_job OUT employees.job_id%TYPE) IS
BEGIN
    SELECT salary, job_id
    INTO p_sal, p_job
    FROM employees
    WHERE employee_id = p.empid;
END get_employee;

FUNCTION get_employee(p_emp_id employees.employee_id%type)
    return employees%rowtype IS
    rec_emp employees%rowtype;
BEGIN
    SELECT * INTO rec_emp
    FROM employees
    WHERE employee_id = p_emp_id;
    RETURN rec_emp;
END;

FUNCTION get_employee(p_family_name employees.last_name%type)
    return employees%rowtype IS
    rec_emp employees%rowtype;
BEGIN

```

```

SELECT * INTO rec_emp
FROM employees
WHERE last_name = p_family_name;
RETURN rec_emp;
END;

PROCEDURE get_employees(p_dept_id
employees.department_id%type) IS
BEGIN
    SELECT * BULK COLLECT INTO emp_table
    FROM EMPLOYEES
    WHERE department_id = p_dept_id;
END;

PROCEDURE init_departments IS
BEGIN
    FOR rec IN (SELECT department_id FROM departments)
    LOOP
        valid_departments(rec.department_id) := TRUE;
    END LOOP;
END;

PROCEDURE print_employee(p_rec_emp employees%rowtype) IS
BEGIN
    DBMS_OUTPUT.PUT_LINE(p_rec_emp.department_id || ' ' ||
                         p_rec_emp.employee_id|| ' ' ||
                         p_rec_emp.first_name|| ' ' ||
                         p_rec_emp.last_name|| ' ' ||
                         p_rec_emp.job_id|| ' ' ||
                         p_rec_emp.salary);
END;

PROCEDURE show_employees IS
BEGIN
    IF emp_table IS NOT NULL THEN
        DBMS_OUTPUT.PUT_LINE('Employees in Package table');
        FOR i IN 1 .. emp_table.COUNT
        LOOP
            print_employee(emp_table(i));
        END LOOP;
    END IF;
END show_employees;

```

```

FUNCTION valid_deptid(p_deptid IN
departments.department_id%TYPE)
    RETURN BOOLEAN IS
    v_dummy PLS_INTEGER;
BEGIN
    RETURN valid_departments.exists(p_deptid);
EXCEPTION
    WHEN NO_DATA_FOUND THEN
        RETURN FALSE;
END valid_deptid;

BEGIN
    init_departments;
END emp_pkg;

/
SHOW ERRORS

```

```

Script Output X
| Task completed in 0.115 seconds
PACKAGE EMP_PKG compiled
No Errors.
PACKAGE BODY EMP_PKG compiled
No Errors.

```

- d. Aktivieren Sie SERVEROUTPUT. Rufen Sie die Prozedur emp\_pkg.get\_employees für Abteilung 30 auf und dann emp\_pkg.show\_employees. Wiederholen Sie den Vorgang für Abteilung 60.

**Entfernen Sie die Kommentarzeichen, und wählen Sie den Code unter der Aufgabe 1\_d. Um die Packageprozeduren aufzurufen, klicken Sie in der SQL Worksheet-Symbolleiste auf das Symbol "Run Script" (F5). Der Code und die Ergebnisse sind unten abgebildet:**

```

SET SERVEROUTPUT ON

EXECUTE emp_pkg.get_employees(30)
EXECUTE emp_pkg.show_employees

EXECUTE emp_pkg.get_employees(60)
EXECUTE emp_pkg.show_employees

```

```

anonymous block completed
anonymous block completed
Employees in Package table
30 209 Samuel Joplin SA_REP 1000
30 114 Den Raphaely PU_MAN 11000
30 115 Alexander Khoo PU_CLERK 3100
30 116 Shelli Baida PU_CLERK 2900
30 117 Sigal Tobias PU_CLERK 2800
30 118 Guy Himuro PU_CLERK 2600
30 119 Karen Colmenares PU_CLERK 2500

anonymous block completed
anonymous block completed
Employees in Package table
60 103 Alexander Hunold IT_PROG 9000
60 104 Bruce Ernst IT_PROG 6600
60 105 David Austin IT_PROG 4800
60 106 Valli Pataballa IT_PROG 4800
60 107 Diana Lorentz IT_PROG 4200

```

2. Ihr Chef möchte jeden Aufruf der Prozedur `add_employee` im Package protokollieren, bei dem ein neuer Mitarbeiter in die Tabelle `EMPLOYEES` eingefügt wird.
- Laden und führen Sie zuerst den Code unter der Aufgabe 2\_a im Skript `/home/oracle/labs/plpu/solns/sol_08.sql` aus, um eine Logtabelle namens `LOG_NEWEMP` und eine Sequence namens `log_newemp_seq` zu erstellen.
- Entfernen Sie die Kommentarzeichen, und wählen Sie den Code unter der Aufgabe 2\_a. Um das Skript auszuführen, klicken Sie in der SQL Worksheet-Symbolleiste auf das Symbol "Run Script" (F5). Der Code und die Ergebnisse sind unten abgebildet.**

```

CREATE TABLE log_newemp (
    entry_id  NUMBER(6) CONSTRAINT log_newemp_pk PRIMARY KEY,
    user_id    VARCHAR2(30),
    log_time   DATE,
    name       VARCHAR2(60)
);

CREATE SEQUENCE log_newemp_seq;

```

```

table LOG_NEWEMP created.
sequence LOG_NEWEMP_SEQ created.

```

- b. Ändern Sie im Body des Packages `EMP_PKG` die Prozedur `add_employee`, die den tatsächlichen `INSERT`-Vorgang ausführt. Fügen Sie wie folgt eine lokale Prozedur namens `audit_newemp` hinzu:
- 1) Die Prozedur `audit_newemp` muss mithilfe einer autonomen Transaktion einen Log-Record in die Tabelle `LOG_NEWEMP` einfügen.
  - 2) Speichern Sie den Benutzer (`USER`), die aktuelle Uhrzeit und den Namen des neuen Mitarbeiters in der Zeile der Logtabelle.
  - 3) Legen Sie die Spalte `entry_id` mit `log_newemp_seq` fest.

**Hinweis:** Denken Sie daran, in einer Prozedur mit einer autonomen Transaktion einen `COMMIT`-Vorgang auszuführen.

**Entfernen Sie die Kommentarzeichen, und wählen Sie den Code unter der Aufgabe 2\_b. Der neu hinzugefügte Code ist im folgenden Codefeld durch Fettschrift hervorgehoben. Um das Skript auszuführen, klicken Sie in der SQL Worksheet-Symbolleiste auf das Symbol "Run Script" (F5). Der Code und die Ergebnisse werden wie folgt angezeigt:**

```
-- Package SPECIFICATION

CREATE OR REPLACE PACKAGE emp_pkg IS

    TYPE emp_tab_type IS TABLE OF employees%ROWTYPE;

    PROCEDURE add_employee(
        p_first_name employees.first_name%TYPE,
        p_last_name employees.last_name%TYPE,
        p_email employees.email%TYPE,
        p_job employees.job_id%TYPE DEFAULT 'SA_REP',
        p_mgr employees.manager_id%TYPE DEFAULT 145,
        p_sal employees.salary%TYPE DEFAULT 1000,
        p_comm employees.commission_pct%TYPE DEFAULT 0,
        p_deptid employees.department_id%TYPE DEFAULT 30);

    PROCEDURE add_employee(
        p_first_name employees.first_name%TYPE,
        p_last_name employees.last_name%TYPE,
        p_deptid employees.department_id%TYPE);

    PROCEDURE get_employee(
        p.empid IN employees.employee_id%TYPE,
        p_sal OUT employees.salary%TYPE,
        p_job OUT employees.job_id%TYPE);

```

```

FUNCTION get_employee(p_emp_id employees.employee_id%type)
    return employees%rowtype;

FUNCTION get_employee(p_family_name employees.last_name%type)
    return employees%rowtype;

PROCEDURE get_employees(p_dept_id
employees.department_id%type);

PROCEDURE init_departments;

PROCEDURE print_employee(p_rec_emp employees%rowtype);

PROCEDURE show_employees;

END emp_pkg;
/
SHOW ERRORS

-- Package BODY

CREATE OR REPLACE PACKAGE BODY emp_pkg IS
    TYPE boolean_tab_type IS TABLE OF BOOLEAN
        INDEX BY BINARY_INTEGER;

    valid_departments boolean_tab_type;
    emp_table          emp_tab_type;

    FUNCTION valid_deptid(p_deptid IN
departments.department_id%TYPE)
        RETURN BOOLEAN;

    PROCEDURE add_employee(
        p_first_name employees.first_name%TYPE,
        p_last_name employees.last_name%TYPE,
        p_email      employees.email%TYPE,
        p_job        employees.job_id%TYPE DEFAULT 'SA_REP',
        p_mgr        employees.manager_id%TYPE DEFAULT 145,
        p_sal        employees.salary%TYPE DEFAULT 1000,
        p_comm       employees.commission_pct%TYPE DEFAULT 0,
        p_deptid     employees.department_id%TYPE DEFAULT 30) IS

    -- New local procedure

```

```

PROCEDURE audit_newemp IS
  PRAGMA AUTONOMOUS_TRANSACTION;
  user_id VARCHAR2(30) := USER;
BEGIN
  INSERT INTO log_newemp (entry_id, user_id, log_time,
                         name)
  VALUES (log_newemp_seq.NEXTVAL, user_id,
          sysdate,p_first_name||' '||p_last_name);
  COMMIT;
END audit_newemp;

BEGIN
-- add_employee
  IF valid_deptid(p_deptid) THEN
    INSERT INTO employees(employee_id, first_name, last_name,
email,
                           job_id, manager_id, hire_date, salary, commission_pct,
department_id)
    VALUES (employees_seq.NEXTVAL, p_first_name, p_last_name,
p_email,
            p_job, p_mgr, TRUNC(SYSDATE), p_sal, p_comm, p_deptid);
  ELSE
    RAISE_APPLICATION_ERROR (-20204, 'Invalid department ID.
Try again.');
  END IF;
END add_employee;

PROCEDURE add_employee(
  p_first_name employees.first_name%TYPE,
  p_last_name employees.last_name%TYPE,
  p_deptid employees.department_id%TYPE) IS
  p_email employees.email%type;
BEGIN
  p_email := UPPER(SUBSTR(p_first_name, 1,
1)||SUBSTR(p_last_name, 1, 7));
  add_employee(p_first_name, p_last_name, p_email, p_deptid =>
p_deptid);
END;

PROCEDURE get_employee(
  p.empid IN employees.employee_id%TYPE,
  p_sal OUT employees.salary%TYPE,
  p_job OUT employees.job_id%TYPE) IS

```

```

BEGIN
    SELECT salary, job_id
    INTO p_sal, p_job
    FROM employees
    WHERE employee_id = p_empid;
END get_employee;

FUNCTION get_employee(p_emp_id employees.employee_id%type)
    return employees%rowtype IS
    rec_emp employees%rowtype;
BEGIN
    SELECT * INTO rec_emp
    FROM employees
    WHERE employee_id = p_emp_id;
    RETURN rec_emp;
END;

FUNCTION get_employee(p_family_name employees.last_name%type)
    return employees%rowtype IS
    rec_emp employees%rowtype;
BEGIN
    SELECT * INTO rec_emp
    FROM employees
    WHERE last_name = p_family_name;
    RETURN rec_emp;
END;

-- New get_employees procedure.

PROCEDURE get_employees(p_dept_id
employees.department_id%type) IS
BEGIN
    SELECT * BULK COLLECT INTO emp_table
    FROM EMPLOYEES
    WHERE department_id = p_dept_id;
END;

PROCEDURE init_departments IS
BEGIN
    FOR rec IN (SELECT department_id FROM departments)
    LOOP
        valid_departments(rec.department_id) := TRUE;
    END LOOP;

```

```

END;

PROCEDURE print_employee(p_rec_emp employees%rowtype) IS
BEGIN
    DBMS_OUTPUT.PUT_LINE(p_rec_emp.department_id || ' ' ||
                         p_rec_emp.employee_id|| ' ' ||
                         p_rec_emp.first_name|| ' ' ||
                         p_rec_emp.last_name|| ' ' ||
                         p_rec_emp.job_id|| ' ' ||
                         p_rec_emp.salary);
END;

PROCEDURE show_employees IS
BEGIN
    IF emp_table IS NOT NULL THEN
        DBMS_OUTPUT.PUT_LINE('Employees in Package table');
        FOR i IN 1 .. emp_table.COUNT
        LOOP
            print_employee(emp_table(i));
        END LOOP;
    END IF;
END show_employees;

FUNCTION valid_deptid(p_deptid IN
departments.department_id%TYPE)
    RETURN BOOLEAN IS
    v_dummy PLS_INTEGER;
BEGIN
    RETURN valid_departments.exists(p_deptid);
EXCEPTION
    WHEN NO_DATA_FOUND THEN
        RETURN FALSE;
END valid_deptid;

BEGIN
    init_departments;
END emp_pkg;
/
SHOW ERRORS

```

The screenshot shows the 'Script Output' window from an Oracle SQL Worksheet. It displays the following text:

```

Script Output X
| Task completed in 0.163 seconds
PACKAGE EMP_PKG compiled
No Errors.
PACKAGE BODY EMP_PKG compiled
No Errors.

```

- c. Ändern Sie die Prozedur add\_employee, um audit\_emp aufzurufen, bevor der INSERT-Vorgang ausgeführt wird.

**Entfernen Sie die Kommentarzeichen, und wählen Sie den Code unter der Aufgabe 2\_c. Der neu hinzugefügte Code ist im folgenden Codefeld durch Fettschrift hervorgehoben. Um das Skript auszuführen, klicken Sie in der SQL Worksheet-Symbolleiste auf das Symbol "Run Script" (F5). Der Code und die Ergebnisse sind unten abgebildet.**

```

-- Package SPECIFICATION

CREATE OR REPLACE PACKAGE emp_pkg IS

    TYPE emp_tab_type IS TABLE OF employees%ROWTYPE;

    PROCEDURE add_employee(
        p_first_name employees.first_name%TYPE,
        p_last_name employees.last_name%TYPE,
        p_email employees.email%TYPE,
        p_job employees.job_id%TYPE DEFAULT 'SA_REP',
        p_mgr employees.manager_id%TYPE DEFAULT 145,
        p_sal employees.salary%TYPE DEFAULT 1000,
        p_comm employees.commission_pct%TYPE DEFAULT 0,
        p_deptid employees.department_id%TYPE DEFAULT 30);

    PROCEDURE add_employee(
        p_first_name employees.first_name%TYPE,
        p_last_name employees.last_name%TYPE,
        p_deptid employees.department_id%TYPE);

    PROCEDURE get_employee(
        p.empid IN employees.employee_id%TYPE,
        p_sal OUT employees.salary%TYPE,
        p_job OUT employees.job_id%TYPE);

    FUNCTION get_employee(p.emp_id employees.employee_id%type)
        return employees%rowtype;

```

```

FUNCTION get_employee(p_family_name employees.last_name%type)
    return employees%rowtype;

PROCEDURE get_employees(p_dept_id
employees.department_id%type);

PROCEDURE init_departments;

PROCEDURE print_employee(p_rec_emp employees%rowtype);

PROCEDURE show_employees;

END emp_pkg;
/
SHOW ERRORS

-- Package BODY

CREATE OR REPLACE PACKAGE BODY emp_pkg IS
    TYPE boolean_tab_type IS TABLE OF BOOLEAN
        INDEX BY BINARY_INTEGER;

    valid_departments boolean_tab_type;
    emp_table          emp_tab_type;

    FUNCTION valid_deptid(p_deptid IN
departments.department_id%TYPE)
        RETURN BOOLEAN;

    PROCEDURE add_employee(
        p_first_name employees.first_name%TYPE,
        p_last_name employees.last_name%TYPE,
        p_email      employees.email%TYPE,
        p_job        employees.job_id%TYPE DEFAULT 'SA_REP',
        p_mgr        employees.manager_id%TYPE DEFAULT 145,
        p_sal        employees.salary%TYPE DEFAULT 1000,
        p_comm       employees.commission_pct%TYPE DEFAULT 0,
        p_deptid     employees.department_id%TYPE DEFAULT 30) IS

    PROCEDURE audit_newemp IS
        PRAGMA AUTONOMOUS_TRANSACTION;
        user_id VARCHAR2(30) := USER;

```

```

BEGIN
    INSERT INTO log_newemp (entry_id, user_id, log_time, name)
    VALUES (log_newemp_seq.NEXTVAL, user_id,
    sysdate,p_first_name||' '||p_last_name);
    COMMIT;
END audit_newemp;

BEGIN -- add_employee
IF valid_deptid(p_deptid) THEN
    audit_newemp;
    INSERT INTO employees(employee_id, first_name, last_name,
email,
        job_id, manager_id, hire_date, salary, commission_pct,
department_id)
    VALUES (employees_seq.NEXTVAL, p_first_name, p_last_name,
p_email,
        p_job, p_mgr, TRUNC(SYSDATE), p_sal, p_comm, p_deptid);
ELSE
    RAISE_APPLICATION_ERROR (-20204, 'Invalid department ID.
Try again.');
END IF;
END add_employee;

PROCEDURE add_employee(
    p_first_name employees.first_name%TYPE,
    p_last_name employees.last_name%TYPE,
    p_deptid employees.department_id%TYPE) IS
    p_email employees.email%type;
BEGIN
    p_email := UPPER(SUBSTR(p_first_name, 1,
1)||SUBSTR(p_last_name, 1, 7));
    add_employee(p_first_name, p_last_name, p_email, p_deptid =>
p_deptid);
END;

PROCEDURE get_employee(
    p.empid IN employees.employee_id%TYPE,
    p_sal OUT employees.salary%TYPE,
    p_job OUT employees.job_id%TYPE) IS
BEGIN
    SELECT salary, job_id
    INTO p_sal, p_job
    FROM employees
    WHERE employee_id = p.empid;

```

```

END get_employee;

FUNCTION get_employee(p_emp_id employees.employee_id%type)
    return employees%rowtype IS
    rec_emp employees%rowtype;
BEGIN
    SELECT * INTO rec_emp
    FROM employees
    WHERE employee_id = p_emp_id;
    RETURN rec_emp;
END;

FUNCTION get_employee(p_family_name employees.last_name%type)
    return employees%rowtype IS
    rec_emp employees%rowtype;
BEGIN
    SELECT * INTO rec_emp
    FROM employees
    WHERE last_name = p_family_name;
    RETURN rec_emp;
END;

PROCEDURE get_employees(p_dept_id
employees.department_id%type) IS
BEGIN
    SELECT * BULK COLLECT INTO emp_table
    FROM EMPLOYEES
    WHERE department_id = p_dept_id;
END;

PROCEDURE init_departments IS
BEGIN
    FOR rec IN (SELECT department_id FROM departments)
    LOOP
        valid_departments(rec.department_id) := TRUE;
    END LOOP;
END;

PROCEDURE print_employee(p_rec_emp employees%rowtype) IS
BEGIN
    DBMS_OUTPUT.PUT_LINE(p_rec_emp.department_id || ' ' ||
                        p_rec_emp.employee_id|| ' '|| )

```

```

        p_rec_emp.first_name||' ' ||
        p_rec_emp.last_name||' ' ||
        p_rec_emp.job_id||' ' ||
        p_rec_emp.salary);

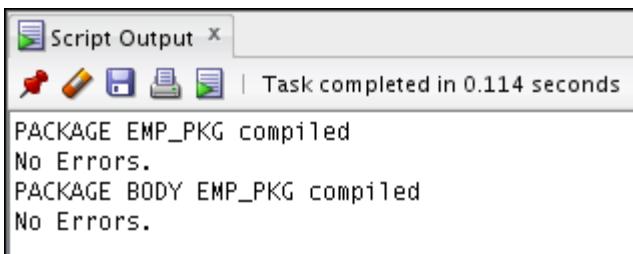
END;

PROCEDURE show_employees IS
BEGIN
  IF emp_table IS NOT NULL THEN
    DBMS_OUTPUT.PUT_LINE('Employees in Package table');
    FOR i IN 1 .. emp_table.COUNT
    LOOP
      print_employee(emp_table(i));
    END LOOP;
  END IF;
END show_employees;

FUNCTION valid_deptid(p_deptid IN
departments.department_id%TYPE)
  RETURN BOOLEAN IS
  v_dummy PLS_INTEGER;
BEGIN
  RETURN valid_departments.exists(p_deptid);
EXCEPTION
  WHEN NO_DATA_FOUND THEN

  RETURN FALSE;
END valid_deptid;
BEGIN
  init_departments;
END emp_pkg;
/
SHOW ERRORS

```



The screenshot shows the 'Script Output' window from Oracle SQL Developer. The window title is 'Script Output'. It contains the following text:

```

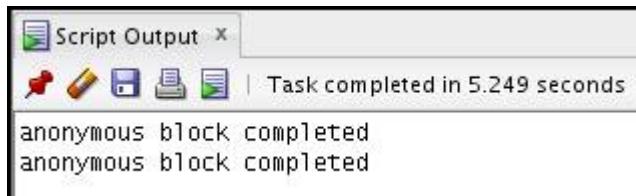
Script Output x
| Task completed in 0.114 seconds
PACKAGE EMP_PKG compiled
No Errors.
PACKAGE BODY EMP_PKG compiled
No Errors.

```

- d. Rufen Sie die Prozedur `add_employee` für die nachstehenden neuen Mitarbeiter auf:  
 Max Smart in Abteilung 20 und Clark Kent in Abteilung 10. Was geschieht?

**Entfernen Sie die Kommentarzeichen, und wählen Sie den Code unter der Aufgabe 2\_d. Um das Skript auszuführen, klicken Sie in der SQL Worksheet-Symbolleiste auf das Symbol "Run Script" (F5). Der Code und die Ergebnisse werden wie folgt angezeigt.**

```
EXECUTE emp_pkg.add_employee('Max', 'Smart', 20)
EXECUTE emp_pkg.add_employee('Clark', 'Kent', 10)
```



**Beide INSERT-Anweisungen werden erfolgreich ausgeführt. Die Logtabelle enthält die beiden im nächsten Schritt abgebildeten Log-Records.**

- e. Fragen Sie die beiden hinzugefügten EMPLOYEES-Records und die Records in der Tabelle `LOG_NEWEMP` ab. Wie viele Log-Records sind vorhanden?

**Entfernen Sie die Kommentarzeichen, und wählen Sie den Code unter der Aufgabe 2\_e. Um das Skript auszuführen, klicken Sie in der SQL Worksheet-Symbolleiste auf das Symbol "Run Script" (F5). Der Code und die Ergebnisse werden wie folgt angezeigt:**

```
select department_id, employee_id, last_name, first_name
from employees
where last_name in ('Kent', 'Smart');

select * from log_newemp;
```

| Script Output                   |             |           |            |
|---------------------------------|-------------|-----------|------------|
| Task completed in 0.011 seconds |             |           |            |
| DEPARTMENT_ID                   | EMPLOYEE_ID | LAST_NAME | FIRST_NAME |
| 10                              | 212         | Kent      | Clark      |
| 20                              | 211         | Smart     | Max        |
| ENTRY_ID                        | USER_ID     | LOG_TIME  | NAME       |
| 1                               | ORA61       | 20-NOV-12 | Max Smart  |
| 2                               | ORA61       | 20-NOV-12 | Clark Kent |

**Es sind zwei Log-Records vorhanden, einer für Smart und ein weiterer für Kent.**

- f. Führen Sie eine ROLLBACK-Anweisung aus, um die nicht festgeschriebenen INSERT-Vorgänge rückgängig zu machen. Verwenden Sie die Abfragen aus Schritt 2e wie folgt:
- 1) Prüfen Sie anhand der ersten Abfrage, ob die Zeilen für die Mitarbeiter Smart und Kent entfernt wurden.
  - 2) Prüfen Sie anhand der zweiten Abfrage die Log-Records in der Tabelle LOG\_NEWEMP. Wie viele Log-Records sind vorhanden? Warum?

ROLLBACK;

The screenshot shows two stacked "Script Output" windows from Oracle SQL Developer. The top window displays the message "rollback complete." after a task completed in 0.006 seconds. The bottom window displays the contents of the LOG\_NEWEMP table, which contains two log records for entries 1 and 2, both made by user ORA61 on 20-NOV-12, with names Max Smart and Clark Kent respectively.

| ENTRY_ID | USER_ID | LOG_TIME  | NAME       |
|----------|---------|-----------|------------|
| 1        | ORA61   | 20-NOV-12 | Max Smart  |
| 2        | ORA61   | 20-NOV-12 | Clark Kent |

**Die beiden Mitarbeiter-Records werden entfernt (zurückgerollt). Die beiden Log-Records bleiben in der Logtabelle, da sie mit einer autonomen Transaktion eingefügt wurden, die vom Rollback in der Haupttransaktion nicht beeinflusst wird.**



# **Übungen zu Lektion 9 – Trigger erstellen**

**Kapitel 9**

# Übungen zu Lektion 9 – Überblick

---

## Lektionsüberblick

In diesen Übungen erstellen Sie Statement Trigger und Row Trigger. Außerdem erstellen Sie Prozeduren, die aus den Triggern aufgerufen werden.

### Hinweis:

1. Führen Sie vor Beginn dieser Übung das Skript  
`/home/oracle/plpu/code_ex/code_ex_scripts/clean_up_scripts/  
cleanup_09.sql` aus.
2. Wenn Sie einen Schritt in einer Übung ausgelassen haben, führen Sie das entsprechende Lösungsskript zu diesem Übungsschritt aus, bevor Sie mit dem nächsten Schritt oder der nächsten Übung fortfahren.

# Übung 1 zu Lektion 9 – Statement Trigger und Row Trigger erstellen

---

## Überblick

In diesen Übungen erstellen Sie Statement Trigger und Row Trigger. Außerdem erstellen Sie Prozeduren, die aus den Triggern aufgerufen werden.

**Hinweis:** Führen Sie vor den folgenden Aufgaben das Skript `cleanup_09.sql` in `/home/oracle/plpu/code_ex/code_ex_scripts/clean_up_scripts/` aus.

## Aufgabe

1. Die Zeilen in der Tabelle `JOBSP` speichern für verschiedene `JOB_ID`-Werte das jeweils zulässige Mindest- und Höchstgehalt. Sie sollen Code eingeben, um sicherzustellen, dass bei Vorgängen vom Typ `INSERT` und `UPDATE` das Gehalt der Mitarbeiter innerhalb der für ihre Tätigkeit zulässigen Spanne liegt.
  - a. Erstellen Sie wie folgt eine Prozedur namens `CHECK_SALARY`:
    - 1) Die Prozedur nimmt zwei Parameter an: einen für die Zeichenfolge der Tätigkeits-ID eines Mitarbeiters und einen für das Gehalt.
    - 2) Die Prozedur bestimmt anhand der Tätigkeits-ID das Mindest- und Höchstgehalt für die angegebene Tätigkeit.
    - 3) Wenn der Gehaltsparameter nicht innerhalb der für die Tätigkeit zulässigen Gehaltsspanne liegt (einschließlich Mindest- und Höchstgehalt), muss er eine Anwendungs-Exception mit der folgenden Meldung auslösen: "Invalid salary <sal>. Salaries for job <jobid> must be between <min> and <max>." Ersetzen Sie die verschiedenen Elemente in der Meldung durch Parameter- und Variablenwerte, die Sie aus Abfragen erhalten haben. Speichern Sie die Datei.
  - b. Erstellen Sie einen Trigger namens `CHECK_SALARY_TRG` für die Tabelle `EMPLOYEES`, der vor einem Vorgang vom Typ `INSERT` oder `UPDATE` für jede Zeile ausgelöst wird:
    - 1) Der Trigger soll die Prozedur `CHECK_SALARY` aufrufen, um die Geschäftslogik auszuführen.
    - 2) Er soll die neue Tätigkeits-ID und das Gehalt an die Prozedurparameter übergeben.
2. Testen Sie den Trigger `CHECK_SALARY_TRG` anhand der folgenden Fälle:
  - a. Fügen Sie mit der Prozedur `EMP_PKG.ADD_EMPLOYEE` die Mitarbeiterin Eleanor Beh in Abteilung 30 ein. Was geschieht und warum?
  - b. Aktualisieren Sie das Gehalt von Mitarbeiter 115 auf 2.000 \$. Ändern Sie in einem gesonderten `UPDATE`-Vorgang die Tätigkeits-ID des Mitarbeiters in `HR.REP`. Was passiert in den einzelnen Fällen?
  - c. Aktualisieren Sie das Gehalt von Mitarbeiter 115 auf 2.800 \$. Was geschieht?
3. Aktualisieren Sie den Trigger `CHECK_SALARY_TRG`, sodass er nur dann ausgelöst wird, wenn die Tätigkeits-ID oder das Gehalt geändert wurde.
  - a. Implementieren Sie die Geschäftsregel mit einer `WHEN`-Klausel, um zu prüfen, ob der Wert für `JOB_ID` oder `SALARY` geändert wurde.

**Hinweis:** Stellen Sie sicher, dass die Bedingung in den Werten `OLD.column_name` auch `NULL` verarbeitet, wenn ein `INSERT`-Vorgang ausgeführt wird. Andernfalls treten bei `INSERT`-Vorgängen Fehler auf.

- b. Testen Sie den Trigger, indem Sie die Prozedur `EMP_PKG.ADD_EMPLOYEE` mit den folgenden Parameterwerten ausführen:
- `p_first_name: 'Eleanor'`
  - `p_last_name: 'Beh'`
  - `p_Email: 'EBEH'`
  - `p_Job: 'IT_PROG'`
  - `p_Sal: 5000`
- c. Aktualisieren Sie Mitarbeiter mit der Tätigkeit `IT_PROG`, indem Sie ihr Gehalt um 2.000 \$ erhöhen. Was geschieht?
- d. Aktualisieren Sie das Gehalt für `Eleanor Beh` auf 9.000 \$.
- Tipp:** Verwenden Sie eine `UPDATE`-Anweisung mit einer Unterabfrage in der Klausel `WHERE`. Was geschieht?
- e. Ändern Sie die Tätigkeit von `Eleanor Beh` in `ST_MAN`, und verwenden Sie dabei eine andere `UPDATE`-Anweisung mit einer Unterabfrage. Was geschieht?
4. Sie sollen verhindern, dass Mitarbeiter während der Geschäftszeiten gelöscht werden.
- a. Erstellen Sie einen Statement Trigger namens `DELETE_EMP_TRG` für die Tabelle `EMPLOYEES`, um zu verhindern, dass Zeilen werktags zwischen 9 und 18 Uhr gelöscht werden.
  - b. Versuchen Sie, Mitarbeiter mit der Tätigkeits-ID `SA REP`, die keiner Abteilung zugeordnet sind, zu löschen.

**Tipp:** Dies ist der Mitarbeiter Grant mit der ID 178.

## Übung 1 zu Lektion 9 – Lösung: Statement Trigger und Row Trigger erstellen

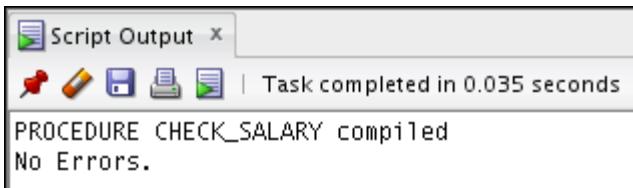
In diesen Übungen erstellen Sie Statement Trigger und Row Trigger. Außerdem erstellen Sie Prozeduren, die aus den Triggern aufgerufen werden.

1. Die Zeilen in der Tabelle JOBS speichern für verschiedene JOB\_ID-Werte das jeweils zulässige Mindest- und Höchstgehalt. Sie sollen Code eingeben, um sicherzustellen, dass bei Vorgängen vom Typ INSERT und UPDATE das Gehalt der Mitarbeiter innerhalb der für ihre Tätigkeit zulässigen Spanne liegt.
  - a. Erstellen Sie wie folgt eine Prozedur namens CHECK\_SALARY:
    - 1) Die Prozedur nimmt zwei Parameter an: einen für die Zeichenfolge der Tätigkeits-ID eines Mitarbeiters und einen für das Gehalt.
    - 2) Die Prozedur verwendet die Tätigkeits-ID, um das Mindest- und Höchstgehalt für die angegebene Tätigkeit zu bestimmen.
    - 3) Wenn der Gehaltsparameter nicht innerhalb der für die Tätigkeit zulässigen Gehaltsspanne liegt (einschließlich Mindest- und Höchstgehalt), muss er eine Anwendungs-Exception mit der folgenden Meldung auslösen: "Invalid salary <sal>. Salaries for job <jobid> must be between <min> and <max>". Ersetzen Sie die verschiedenen Elemente in der Meldung durch Parameter- und Variablenwerte, die Sie aus Abfragen erhalten haben. Speichern Sie die Datei.

**Öffnen Sie das Skript sol\_09.sql im Verzeichnis**

/home/oracle/labs/plpu/soln. Entfernen Sie die Kommentarzeichen, und wählen Sie den Code unter der Aufgabe 1\_a. Um das Skript auszuführen, klicken Sie in der SQL Worksheet-Symbolleiste auf das Symbol "Run Script" (F5). Der Code und die Ergebnisse sind unten abgebildet.

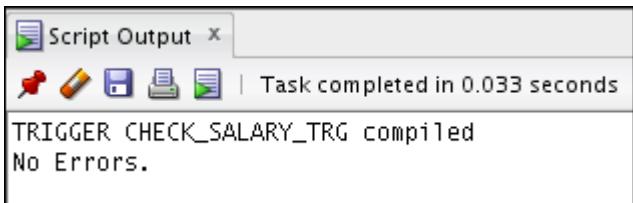
```
CREATE OR REPLACE PROCEDURE check_salary (p_the_job VARCHAR2,
p_the_salary NUMBER) IS
  v_minsal jobs.min_salary%type;
  v_maxsal jobs.max_salary%type;
BEGIN
  SELECT min_salary, max_salary INTO v_minsal, v_maxsal
  FROM jobs
  WHERE job_id = UPPER(p_the_job);
  IF p_the_salary NOT BETWEEN v_minsal AND v_maxsal THEN
    RAISE_APPLICATION_ERROR(-20100,
      'Invalid salary $' || p_the_salary || '. ' ||
      'Salaries for job ' || p_the_job ||
      ' must be between $'|| v_minsal || ' and $' || v_maxsal);
  END IF;
END;
/
SHOW ERRORS
```



- b. Erstellen Sie in der Tabelle EMPLOYEES einen Trigger namens CHECK\_SALARY\_TRG, der vor einem INSERT- oder UPDATE-Vorgang in einer Zeile ausgelöst wird.
- 1) Der Trigger soll die Prozedur CHECK\_SALARY aufrufen, um die Geschäftslogik auszuführen.
  - 2) Er soll die neue Tätigkeits-ID und das Gehalt an die Prozedurparameter übergeben.

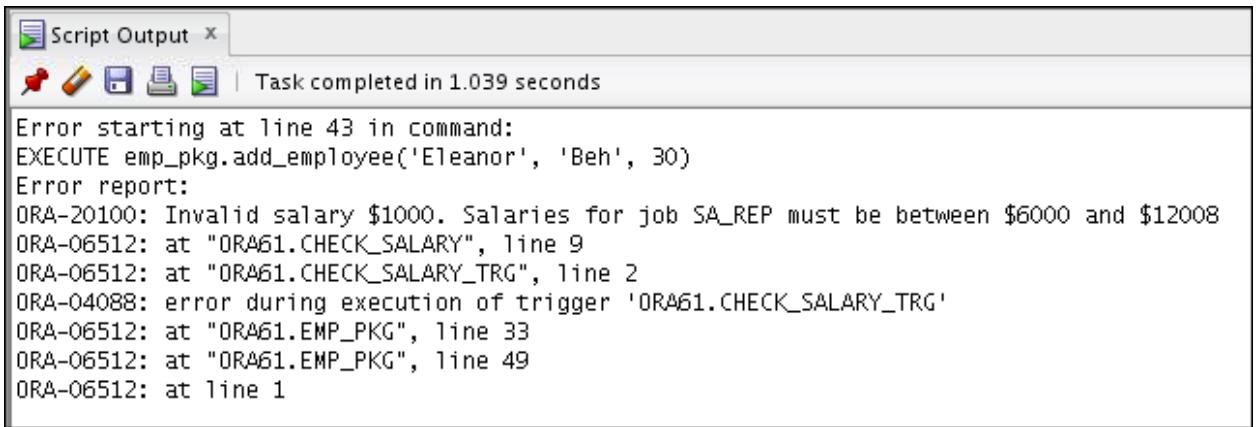
**Entfernen Sie die Kommentarzeichen, und wählen Sie den Code unter der Aufgabe 1\_b. Um das Skript auszuführen, klicken Sie in der SQL Worksheet-Symbolleiste auf das Symbol "Run Script" (F5). Der Code und die Ergebnisse sind unten abgebildet.**

```
CREATE OR REPLACE TRIGGER check_salary_trg
BEFORE INSERT OR UPDATE OF job_id, salary
ON employees
FOR EACH ROW
BEGIN
    check_salary(:new.job_id, :new.salary);
END;
/
SHOW ERRORS
```



2. Testen Sie den Trigger CHECK\_SALARY\_TRG anhand der folgenden Fälle:
- a. Fügen Sie mit der Prozedur EMP\_PKG.ADD\_EMPLOYEE die Mitarbeiterin Eleanor Beh in Abteilung 30 ein. Was geschieht und warum?
- Entfernen Sie die Kommentarzeichen, und wählen Sie den Code unter der Aufgabe 2\_a. Um das Skript auszuführen, klicken Sie in der SQL Worksheet-Symbolleiste auf das Symbol "Run Script" (F5). Der Code und die Ergebnisse sind unten abgebildet.**

```
EXECUTE emp_pkg.add_employee('Eleanor', 'Beh', 30)
```



The screenshot shows the 'Script Output' window from Oracle SQL Worksheet. It displays the following error message:

```

Script Output X
| Task completed in 1.039 seconds
Error starting at line 43 in command:
EXECUTE emp_pkg.add_employee('Eleanor', 'Beh', 30)
Error report:
ORA-20100: Invalid salary $1000. Salaries for job SA_REP must be between $6000 and $12008
ORA-06512: at "ORA61.CHECK_SALARY", line 9
ORA-06512: at "ORA61.CHECK_SALARY_TRG", line 2
ORA-04088: error during execution of trigger 'ORA61.CHECK_SALARY_TRG'
ORA-06512: at "ORA61.EMP_PKG", line 33
ORA-06512: at "ORA61.EMP_PKG", line 49
ORA-06512: at line 1

```

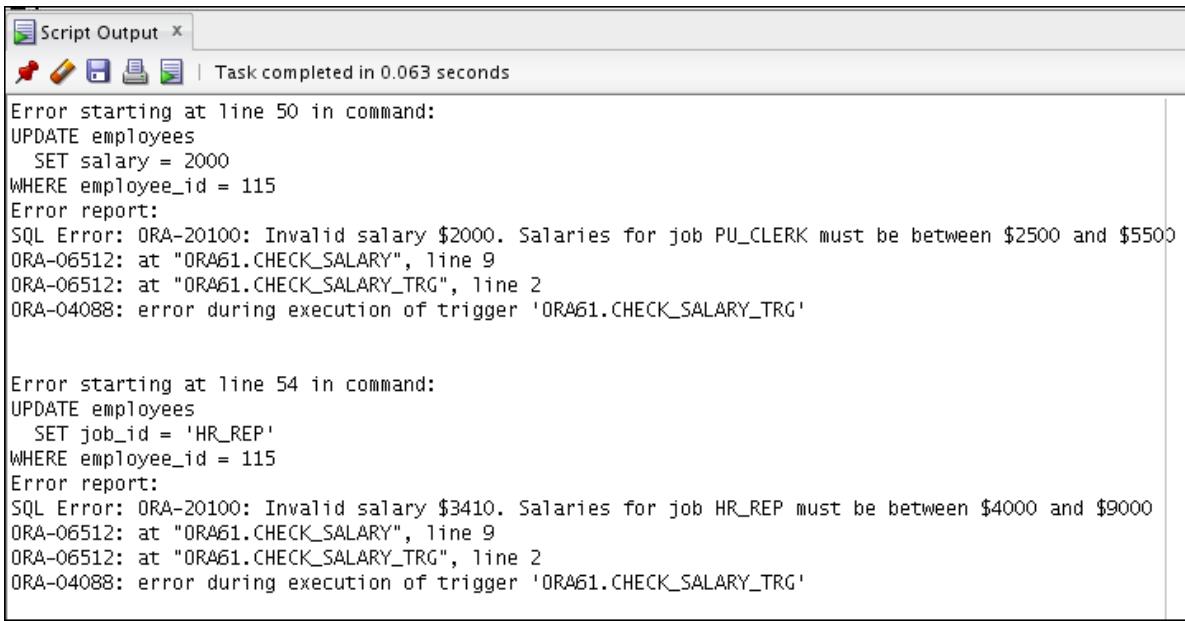
**Der Trigger löst eine Exception aus, weil die Prozedur `EMP_PKG.ADD_EMPLOYEE` eine überladene Version von sich selbst aufruft, bei der das Standardgehalt 1.000 \$ und die Standardtätigkeits-ID `SA_REP` verwendet werden. Das Mindestgehalt in der Tabelle `JOBs` für die Tätigkeits-ID `SA_REP` ist jedoch mit 6.000 \$ angegeben.**

- Aktualisieren Sie das Gehalt von Mitarbeiter 115 auf 2.000 \$. Ändern Sie in einem gesonderten UPDATE-Vorgang die Tätigkeits-ID des Mitarbeiters in `HR_REP`. Was passiert in den einzelnen Fällen?

**Entfernen Sie die Kommentarzeichen, und wählen Sie den Code unter der Aufgabe 2\_b. Um das Skript auszuführen, klicken Sie in der SQL Worksheet-Symbolleiste auf das Symbol "Run Script" (F5). Der Code und die Ergebnisse sind unten abgebildet.**

```
UPDATE employees
  SET salary = 2000
 WHERE employee_id = 115;
```

```
UPDATE employees
  SET job_id = 'HR_REP'
 WHERE employee_id = 115;
```



```

Script Output x
| Task completed in 0.063 seconds
Error starting at line 50 in command:
UPDATE employees
  SET salary = 2000
 WHERE employee_id = 115
Error report:
SQL Error: ORA-20100: Invalid salary $2000. Salaries for job PU_CLERK must be between $2500 and $5500
ORA-06512: at "ORA61.CHECK_SALARY", line 9
ORA-06512: at "ORA61.CHECK_SALARY_TRG", line 2
ORA-04088: error during execution of trigger 'ORA61.CHECK_SALARY_TRG'

Error starting at line 54 in command:
UPDATE employees
  SET job_id = 'HR REP'
 WHERE employee_id = 115
Error report:
SQL Error: ORA-20100: Invalid salary $3410. Salaries for job HR REP must be between $4000 and $9000
ORA-06512: at "ORA61.CHECK_SALARY", line 9
ORA-06512: at "ORA61.CHECK_SALARY_TRG", line 2
ORA-04088: error during execution of trigger 'ORA61.CHECK_SALARY_TRG'

```

**Die erste UPDATE-Anweisung kann das Gehalt nicht auf 2.000 \$ einstellen. Die Triggerregel CHECK\_SALARY verhindert den UPDATE-Vorgang, da das neue Gehalt für den Mitarbeiter 115 unter dem für die Tätigkeits-ID PU\_CLERK zulässigen Mindestgehalt liegt.**

**Der zweite UPDATE-Vorgang kann die Tätigkeit des Mitarbeiters nicht ändern, da das aktuelle Gehalt von 3.100 \$ des Mitarbeiters unter dem Mindestgehalt für die neue Tätigkeits-ID HR REP liegt.**

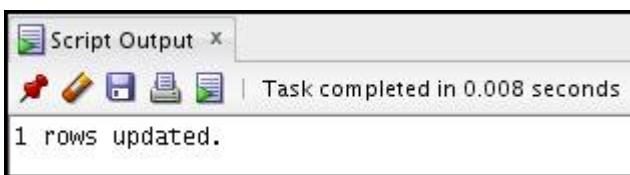
- Aktualisieren Sie das Gehalt von Mitarbeiter 115 auf 2.800 \$. Was geschieht?

**Entfernen Sie die Kommentarzeichen, und wählen Sie den Code unter der Aufgabe 2\_c. Um das Skript auszuführen, klicken Sie in der SQL Worksheet-Symbolleiste auf das Symbol "Run Script" (F5). Der Code und die Ergebnisse sind unten abgebildet.**

```

UPDATE employees
  SET salary = 2800
 WHERE employee_id = 115;

```



```

Script Output x
| Task completed in 0.008 seconds
1 rows updated.

```

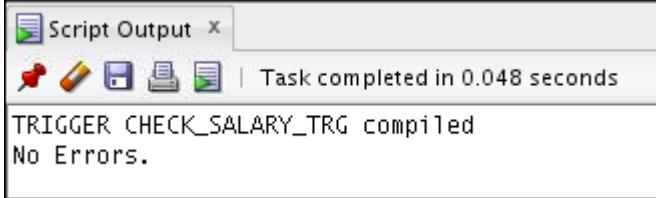
**Der UPDATE-Vorgang ist erfolgreich, da das neue Gehalt innerhalb der zulässigen Spanne für die aktuelle Tätigkeits-ID liegt.**

3. Aktualisieren Sie den Trigger `CHECK_SALARY_TRG`, sodass er nur dann ausgelöst wird, wenn die Tätigkeits-ID oder das Gehalt geändert wurde.
  - a. Implementieren Sie die Geschäftsregel mit einer `WHEN`-Klausel, um zu prüfen, ob der Wert für `JOB_ID` oder `SALARY` geändert wurde.

**Hinweis:** Stellen Sie sicher, dass die Bedingung in den Werten `OLD.column_name` auch `NULL` verarbeitet, wenn ein `INSERT`-Vorgang ausgeführt wird. Andernfalls treten bei `INSERT`-Vorgängen Fehler auf.

**Entfernen Sie die Kommentarzeichen, und wählen Sie den Code unter der Aufgabe 3\_a. Um das Skript auszuführen, klicken Sie in der SQL Worksheet-Symbolleiste auf das Symbol "Run Script" (F5). Der Code und die Ergebnisse sind unten abgebildet.**

```
CREATE OR REPLACE TRIGGER check_salary_trg
BEFORE INSERT OR UPDATE OF job_id, salary
ON employees FOR EACH ROW
WHEN (new.job_id <> NVL(old.job_id,'?') OR
      new.salary <> NVL(old.salary,0))
BEGIN
  check_salary(:new.job_id, :new.salary);
END;
/
SHOW ERRORS
```

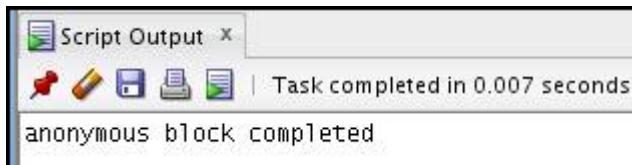


- b. Testen Sie den Trigger, indem Sie die Prozedur `EMP_PKG.ADD_EMPLOYEE` mit den folgenden Parameterwerten ausführen:

- `p_first_name: 'Eleanor'`
- `p_last name: 'Beh'`
- `p_Email: 'EBEH'`
- `p_Job: 'IT_PROG'`
- `p_Sal: 5000`

**Entfernen Sie die Kommentarzeichen, und wählen Sie den Code unter der Aufgabe 3\_b. Um das Skript auszuführen, klicken Sie in der SQL Worksheet-Symbolleiste auf das Symbol "Run Script" (F5). Der Code und die Ergebnisse sind unten abgebildet.**

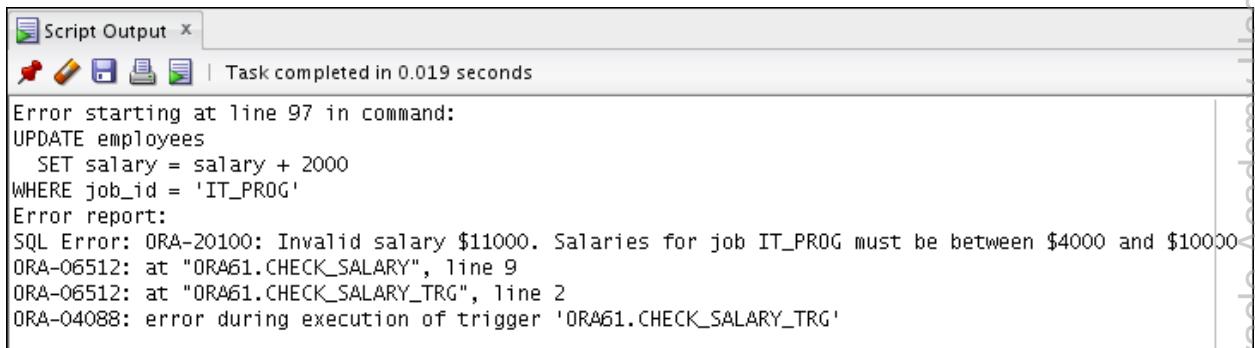
```
BEGIN
  emp_pkg.add_employee('Eleanor', 'Beh', 'EBEH',
                        p_job => 'IT_PROG', p_sal => 5000);
END;
/
```



- c. Aktualisieren Sie Mitarbeiter mit der Tätigkeit IT\_PROG, indem Sie ihr Gehalt um 2.000 \$ erhöhen. Was geschieht?

**Entfernen Sie die Kommentarzeichen, und wählen Sie den Code unter der Aufgabe 3\_c. Um das Skript auszuführen, klicken Sie in der SQL Worksheet-Symbolleiste auf das Symbol "Run Script" (F5). Der Code und die Ergebnisse sind unten abgebildet.**

```
UPDATE employees
   SET salary = salary + 2000
 WHERE job_id = 'IT_PROG';
```



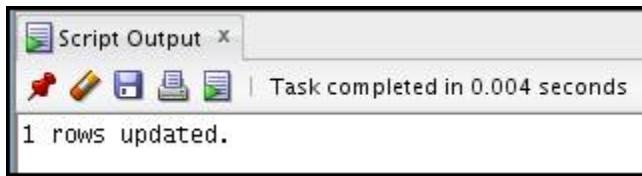
**Das Gehalt eines Mitarbeiters für die angegebene Tätigkeits-ID überschreitet das Höchstgehalt für diese Tätigkeits-ID. Es werden keine Gehälter von Mitarbeitern mit der Tätigkeits-ID IT\_PROG aktualisiert.**

- d. Aktualisieren Sie das Gehalt für Eleanor Beh auf 9.000 \$.

**Tipp:** Verwenden Sie eine UPDATE-Anweisung mit einer Unterabfrage in der Klausel WHERE. Was geschieht?

**Entfernen Sie die Kommentarzeichen, und wählen Sie den Code unter der Aufgabe 3\_d. Um das Skript auszuführen, klicken Sie in der SQL Worksheet-Symbolleiste auf das Symbol "Run Script" (F5). Der Code und die Ergebnisse sind unten abgebildet.**

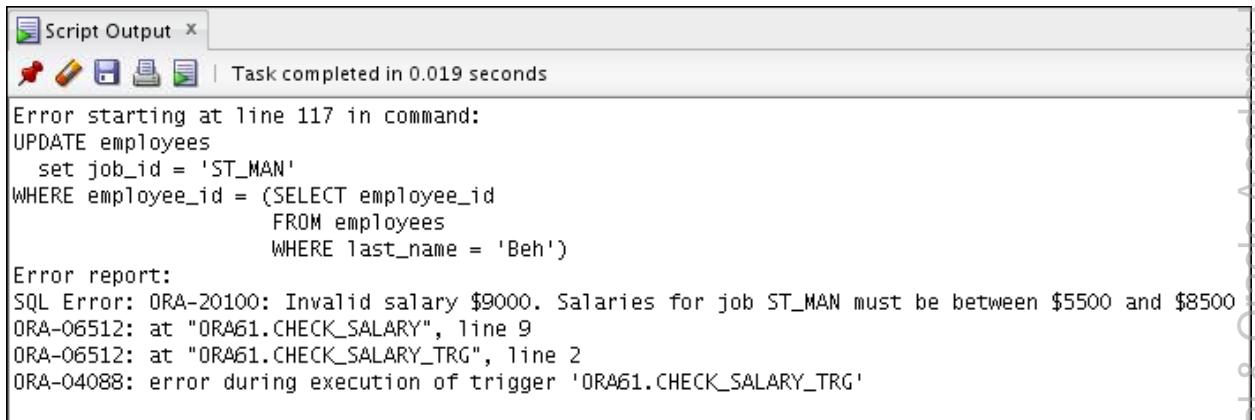
```
UPDATE employees
   SET salary = 9000
 WHERE employee_id = (SELECT employee_id
                        FROM employees
                       WHERE last_name = 'Beh');
```



- e. Ändern Sie die Tätigkeit von Eleanor Beh in ST\_MAN, und verwenden Sie dabei eine andere UPDATE-Anweisung mit einer Unterabfrage. Was geschieht?

**Entfernen Sie die Kommentarzeichen, und wählen Sie den Code unter der Aufgabe 3\_e. Um das Skript auszuführen, klicken Sie in der SQL Worksheet-Symbolleiste auf das Symbol "Run Script" (F5). Der Code und die Ergebnisse sind unten abgebildet.**

```
UPDATE employees
  set job_id = 'ST_MAN'
WHERE employee_id = (SELECT employee_id
                      FROM employees
                     WHERE last_name = 'Beh');
```



**Das Höchstgehalt der neuen Tätigkeits-ID liegt unter dem aktuellen Gehalt der Mitarbeiterin. Deshalb wird der UPDATE-Vorgang nicht ausgeführt.**

4. Sie sollen verhindern, dass Mitarbeiter während der Geschäftszeiten gelöscht werden.
- Erstellen Sie einen Statement Trigger namens DELETE\_EMP\_TRG für die Tabelle EMPLOYEES, um zu verhindern, dass Zeilen werktags zwischen 9 und 18 Uhr gelöscht werden.

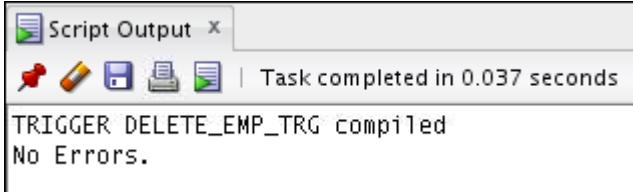
**Entfernen Sie die Kommentarzeichen, und wählen Sie den Code unter der Aufgabe 4\_a. Um das Skript auszuführen, klicken Sie in der SQL Worksheet-Symbolleiste auf das Symbol "Run Script" (F5). Der Code und die Ergebnisse sind unten abgebildet.**

```
CREATE OR REPLACE TRIGGER delete_emp_trg
BEFORE DELETE ON employees
DECLARE
  the_day VARCHAR2(3) := TO_CHAR(SYSDATE, 'DY');
  the_hour PLS_INTEGER := TO_NUMBER(TO_CHAR(SYSDATE, 'HH24'));
BEGIN
```

```

    IF (the_hour BETWEEN 9 AND 18) AND (the_day NOT IN
    ('SAT','SUN')) THEN
        RAISE_APPLICATION_ERROR(-20150,
            'Employee records cannot be deleted during the business
            hours of 9AM and 6PM');
    END IF;
END;
/
SHOW ERRORS

```



- b. Versuchen Sie, Mitarbeiter mit der Tätigkeits-ID SA\_REP, die keiner Abteilung zugeordnet sind, zu löschen.

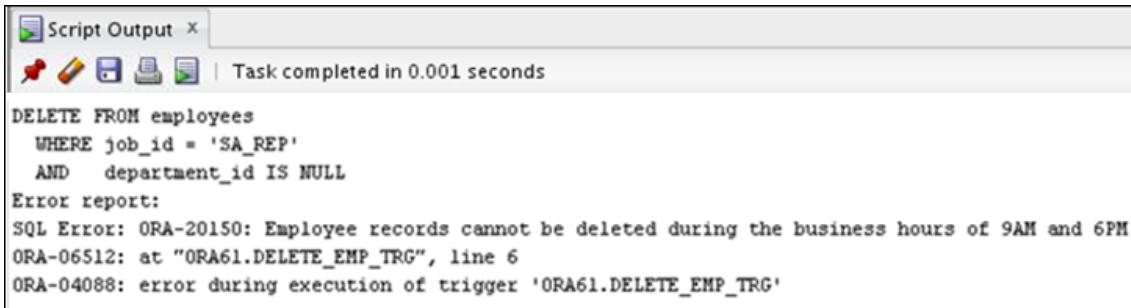
**Tipp:** Dies ist der Mitarbeiter Grant mit der ID 178.

**Entfernen Sie die Kommentarzeichen, und wählen Sie den Code unter der Aufgabe 4\_b. Um das Skript auszuführen, klicken Sie in der SQL Worksheet-Symbolleiste auf das Symbol "Run Script" (F5). Der Code und die Ergebnisse sind unten abgebildet.**

```

DELETE FROM employees
WHERE job_id = 'SA_REP'
    AND department_id IS NULL;

```



**Hinweis:** Ob Sie die Löschtätigkeiten ausführen können, hängt von der aktuellen Uhrzeit auf Ihrem Hostrechner in der Kursumgebung ab. Beispiel: Im Screenshot oben war der Löschtätigkeiten nicht erfolgreich, da er außerhalb der zulässigen Geschäftsstunden ausgeführt wurde (basierend auf der Uhrzeit des Hostrechners).

# **Übungen zu Lektion 10 – Komplexe, DDL- und Datenbankereignis-Trigger erstellen**

**Kapitel 10**

# Übungen zu Lektion 10 – Überblick

---

## Lektionsüberblick

In dieser Übung implementieren Sie eine einfache Geschäftsregel, um die Datenintegrität der Gehälter von Mitarbeitern hinsichtlich der gültigen Gehaltsspanne für ihre Tätigkeit sicherzustellen. Sie erstellen für diese Regel einen Trigger. Während dieses Prozesses wirken sich die neuen Trigger kaskadierend auf Trigger aus, die in den Übungen zur vorherigen Lektion erstellt wurden. Die kaskadierende Auswirkung führt zu einer Exception für die sich verändernde Tabelle JOBS. Anschließend erstellen Sie ein PL/SQL-Package und zusätzliche Trigger, um das Problem der sich verändernden Tabelle zu beheben.

### Hinweis:

1. Führen Sie vor Beginn dieser Übung das Skript  
`/home/oracle/plpu/code_ex/code_ex_scripts/clean_up_scripts/  
cleanup_10.sql` aus.
2. Wenn Sie einen Schritt in einer Übung ausgelassen haben, führen Sie das entsprechende Lösungsskript zu diesem Übungsschritt aus, bevor Sie mit dem nächsten Schritt oder der nächsten Übung fortfahren.

# Übung 1 zu Lektion 10 – Datenintegritätsregeln und Exceptions für sich verändernde Tabellen verwalten

---

## Überblick

In dieser Übung implementieren Sie eine einfache Geschäftsregel, um die Datenintegrität der Gehälter von Mitarbeitern hinsichtlich der gültigen Gehaltsspanne für ihre Tätigkeit sicherzustellen. Sie erstellen für diese Regel einen Trigger. Während dieses Prozesses wirken sich die neuen Trigger kaskadierend auf Trigger aus, die in den Übungen zur vorherigen Lektion erstellt wurden. Die kaskadierende Auswirkung führt zu einer Exception für die sich verändernde Tabelle JOBS. Anschließend erstellen Sie ein PL/SQL-Package und zusätzliche Trigger, um das Problem der sich verändernden Tabelle zu beheben.

**Hinweis:** Führen Sie vor den folgenden Aufgaben das Skript `cleanup_10.sql` in `/home/oracle/plpu/code_ex/code_ex_scripts/clean_up_scripts/` aus.

## Aufgabe

1. Mitarbeiter erhalten eine automatische Gehaltserhöhung, wenn das Mindestgehalt für eine Tätigkeit auf einen höheren Betrag als ihr derzeitiges Gehalt erhöht wird. Implementieren Sie diese Anforderung mithilfe einer Packageprozedur, die von einem Trigger für die Tabelle JOBS aufgerufen wird. Wenn Sie das Mindestgehalt in der Tabelle JOBS erhöhen und das Mitarbeitergehalt aktualisieren, versucht der Trigger `CHECK_SALARY`, die veränderbare Tabelle JOBS zu lesen. Sie erhalten daraufhin eine Exception für die sich verändernde Tabelle, die Sie beheben, indem Sie ein neues Package und zusätzliche Trigger erstellen.
  - a. Aktualisieren Sie wie folgt das Package `EMP_PKG` (das Sie zuletzt in den Übungen zu Lektion 9 aktualisiert haben):
    - 1) Fügen Sie eine Prozedur namens `SET_SALARY` hinzu, die das Mitarbeitergehalt aktualisiert.
    - 2) Die Prozedur `SET_SALARY` nimmt die folgenden zwei Parameter an: Die Tätigkeits-ID für diese Gehälter, die eventuell aktualisiert werden müssen, und das neue Mindestgehalt für die Tätigkeits-ID
  - b. Erstellen Sie einen Row Trigger namens `UPD_MINSALARY_TRG` für die Tabelle JOBS, der die Prozedur `EMP_PKG.SET_SALARY` aufruft, wenn das Mindestgehalt in der Tabelle JOBS für eine angegebene Tätigkeits-ID aktualisiert wird.
  - c. Erstellen Sie eine Abfrage, um die Personalnummer, den Nachnamen, die Tätigkeits-ID, das aktuelle Gehalt und das Mindestgehalt für Mitarbeiter anzuzeigen, die Programmierer mit der Tätigkeits-ID 'IT\_PROG' sind. Aktualisieren Sie anschließend das Mindestgehalt in der Tabelle JOBS durch eine Erhöhung um 1.000 \$. Was geschieht?
2. Um das Problem der sich verändernden Tabelle zu beheben, erstellen Sie ein `JOBS_PKG`-Package, um im Speicher eine Kopie der Zeilen aus der Tabelle JOBS beizubehalten. Ändern Sie anschließend die Prozedur `CHECK_SALARY`. Verwenden Sie anstelle einer Abfrage für eine sich verändernde Tabelle die Packagedaten, um die Exception zu umgehen. Sie müssen jedoch einen Statement Trigger vom Typ `BEFORE INSERT OR UPDATE` für die Tabelle `EMPLOYEES` erstellen, um den Status des Packages `JOBS_PKG` zu initialisieren, bevor der Row Trigger `CHECK_SALARY` ausgelöst wird.

- a. Erstellen Sie ein neues Package namens JOBS\_PKG mit der folgenden Spezifikation:

```

PROCEDURE initialize;
FUNCTION get_minsalary(p_jobid VARCHAR2) RETURN NUMBER;
FUNCTION get_maxsalary(p_jobid VARCHAR2) RETURN NUMBER;
PROCEDURE set_minsalary(p_jobid VARCHAR2,min_salary
NUMBER);
PROCEDURE set_maxsalary(p_jobid VARCHAR2,max_salary
NUMBER);

```

- b. Implementieren Sie wie folgt den Body von JOBS\_PKG:

- 1) Deklarieren Sie eine private PL/SQL-INDEX BY-Tabelle namens jobs\_tab\_type, die durch einen Zeichenfolgetyp indiziert wird, der auf JOBS.JOB\_ID%TYPE basiert.
  - 2) Deklarieren Sie eine private Variable namens jobstab auf der Basis von jobs\_tab\_type.
  - 3) Die Prozedur INITIALIZE liest die Zeilen in der Tabelle JOBS mithilfe einer Cursorschleife und nutzt den Wert JOB\_ID für den Index jobstab, der der entsprechenden Zeile zugewiesen ist.
  - 4) Die Funktion GET\_MINSALARY nutzt einen p\_jobid-Parameter als Index für jobstab und gibt min\_salary für dieses Element zurück.
  - 5) Die Funktion GET\_MAXSALARY nutzt einen p\_jobid-Parameter als Index für jobstab und gibt max\_salary für dieses Element zurück.
  - 6) Die Prozedur SET\_MINSALARY nutzt ihren p\_jobid-Parameter als Index für jobstab, um das Feld min\_salary seines Elements auf den Wert im Parameter min\_salary einzustellen.
  - 7) Die Prozedur SET\_MAXSALARY nutzt ihren p\_jobid-Parameter als Index für jobstab, um das Feld max\_salary seines Elements auf den Wert im Parameter max\_salary einzustellen.
- c. Kopieren Sie die Prozedur CHECK\_SALARY aus der 9. Übung, Aufgabe 1a, und ändern Sie den Code, indem Sie die Abfrage für die Tabelle JOBS durch Anweisungen ersetzen, um die lokalen Variablen minsal und maxsal mit Werten aus den JOBS\_PKG-Daten durch Aufrufen der entsprechenden GET\_\*SALARY-Funktionen einzustellen. Dieser Schritt sollte die Trigger-Exception für sich verändernde Tabellen beseitigen.
- d. Implementieren Sie einen BEFORE INSERT OR UPDATE Statement Trigger namens INIT\_JOBPKG\_TRG, der die Syntax CALL verwendet, um die Prozedur JOBS\_PKG.INITIALIZE aufzurufen. Auf diese Weise stellen Sie sicher, dass der aktuelle Packagestatus vorliegt, bevor die DML-Vorgänge ausgeführt werden.
- e. Testen Sie die Codeänderungen, indem Sie die Abfrage ausführen, um die Mitarbeiter anzuzeigen, die Programmierer sind. Setzen Sie dann eine UPDATE-Anweisung ab, um das Mindestgehalt der Tätigkeits-ID IT\_PROG in der Tabelle JOBS um 1.000 zu erhöhen. Lassen Sie darauf eine Abfrage für Mitarbeiter mit der Tätigkeits-ID IT\_PROG folgen, um die sich daraus ergebenden Änderungen zu prüfen. Bei welchen Mitarbeitern wurde das Gehalt auf das Mindestgehalt für ihre Tätigkeit eingestellt?

3. Da die Prozedur `CHECK_SALARY` durch `CHECK_SALARY_TRG` ausgelöst wird, bevor ein Mitarbeiter hinzugefügt oder aktualisiert wird, müssen Sie prüfen, ob dies weiterhin wie gewohnt funktioniert.
  - a. Testen Sie dies, indem Sie mithilfe von `EMP_PKG.ADD_EMPLOYEE` einen neuen Mitarbeiter mit den folgenden Parametern hinzufügen: ('Steve', 'Morse', 'SMORSE' und `sal => 6500`). Was geschieht?
  - b. So beheben Sie das Problem beim Hinzufügen oder Aktualisieren eines Mitarbeiters:
    - 1) Erstellen Sie einen `BEFORE INSERT OR UPDATE` Statement Trigger namens `EMPLOYEE_INITJOBS_TRG` für die Tabelle `EMPLOYEES`, der die Prozedur `JOBSPKG.INITIALIZE` aufruft.
    - 2) Verwenden Sie im Triggerbody die Syntax `CALL`.
  - c. Testen Sie den Trigger, indem Sie erneut den Mitarbeiter Steve Morse hinzufügen. Prüfen Sie, ob der Record in die Tabelle `EMPLOYEES` eingefügt wurde, indem Sie die Personalnummer, den Vor- und Nachnamen, das Gehalt, die Tätigkeits-ID und die Abteilungsnummer anzeigen.

## Übung 1 zu Lektion 10 – Lösung: Datenintegritätsregeln und Exceptions für sich verändernde Tabellen verwalten

In dieser Übung implementieren Sie eine einfache Geschäftsregel, um die Datenintegrität der Gehälter von Mitarbeitern hinsichtlich der gültigen Gehaltsspanne für ihre Tätigkeit sicherzustellen. Sie erstellen für diese Regel einen Trigger. Während dieses Prozesses wirken sich die neuen Trigger kaskadierend auf Trigger aus, die in den Übungen zur vorherigen Lektion erstellt wurden. Die kaskadierende Auswirkung führt zu einer Exception für die sich verändernde Tabelle JOBS. Anschließend erstellen Sie ein PL/SQL-Package und zusätzliche Trigger, um das Problem der sich verändernden Tabelle zu beheben.

1. Mitarbeiter erhalten eine automatische Gehaltserhöhung, wenn das Mindestgehalt für eine Tätigkeit auf einen höheren Betrag als ihr derzeitiges Gehalt erhöht wird. Implementieren Sie diese Anforderung mithilfe einer Packageprozedur, die von einem Trigger für die Tabelle JOBS aufgerufen wird. Wenn Sie das Mindestgehalt in der Tabelle JOBS erhöhen und das Mitarbeitergehalt aktualisieren, versucht der Trigger CHECK\_SALARY, die veränderbare Tabelle JOBS zu lesen. Sie erhalten daraufhin eine Exception für die sich verändernde Tabelle, die Sie beheben, indem Sie ein neues Package und zusätzliche Trigger erstellen.
  - a. Aktualisieren Sie wie folgt das Package EMP\_PKG (das Sie zuletzt in den Übungen zu Lektion 9 aktualisiert haben):
    - 1) Fügen Sie eine Prozedur namens SET\_SALARY hinzu, die das Mitarbeitergehalt aktualisiert.
    - 2) Die Prozedur SET\_SALARY nimmt die folgenden zwei Parameter an: Die Tätigkeits-ID für diese Gehälter, die eventuell aktualisiert werden müssen, und das neue Mindestgehalt für die Tätigkeits-ID

**Öffnen Sie das Skript sol\_10.sql im Verzeichnis**

/home/oracle/labs/plpu/soln. Entfernen Sie die Kommentarzeichen, und wählen Sie den Code unter der Aufgabe 1\_a. Um das Skript auszuführen, klicken Sie in der SQL Worksheet-Symbolleiste auf das Symbol "Run Script" (F5). Der Code und die Ergebnisse werden wie folgt angezeigt. Der neu hinzugefügte Code ist im folgenden Codefeld durch Fettschrift hervorgehoben.

```
-- Package SPECIFICATION

CREATE OR REPLACE PACKAGE emp_pkg IS

  TYPE emp_tab_type IS TABLE OF employees%ROWTYPE;

  PROCEDURE add_employee(
    p_first_name employees.first_name%TYPE,
    p_last_name employees.last_name%TYPE,
    p_email employees.email%TYPE,
    p_job employees.job_id%TYPE DEFAULT 'SA_REP',
```

```

    p_mgr employees.manager_id%TYPE DEFAULT 145,
    p_sal employees.salary%TYPE DEFAULT 1000,
    p_commm employees.commission_pct%TYPE DEFAULT 0,
    p_deptid employees.department_id%TYPE DEFAULT 30);

PROCEDURE add_employee(
    p_first_name employees.first_name%TYPE,
    p_last_name employees.last_name%TYPE,
    p_deptid employees.department_id%TYPE);

PROCEDURE get_employee(
    p.empid IN employees.employee_id%TYPE,
    p_sal OUT employees.salary%TYPE,
    p_job OUT employees.job_id%TYPE);

FUNCTION get_employee(p.emp_id employees.employee_id%type)
    return employees%rowtype;

FUNCTION get_employee(p.family_name employees.last_name%type)
    return employees%rowtype;

PROCEDURE get_employees(p_dept_id
employees.department_id%type);

PROCEDURE init_departments;

PROCEDURE print_employee(p_rec_emp employees%rowtype);

PROCEDURE show_employees;

-- New set_salary procedure

PROCEDURE set_salary(p_jobid VARCHAR2, p_min_salary NUMBER);

END emp_pkg;
/
SHOW ERRORS

-- Package BODY

CREATE OR REPLACE PACKAGE BODY emp_pkg IS
    TYPE boolean_tab_type IS TABLE OF BOOLEAN
        INDEX BY BINARY_INTEGER;

```

```

valid_departments boolean_tab_type;
emp_table          emp_tab_type;

FUNCTION valid_deptid(p_deptid IN
departments.department_id%TYPE)
    RETURN BOOLEAN;

PROCEDURE add_employee(
    p_first_name employees.first_name%TYPE,
    p_last_name employees.last_name%TYPE,
    p_email     employees.email%TYPE,
    p_job       employees.job_id%TYPE DEFAULT 'SA_REP',
    p_mgr       employees.manager_id%TYPE DEFAULT 145,
    p_sal       employees.salary%TYPE DEFAULT 1000,
    p_comm      employees.commission_pct%TYPE DEFAULT 0,
    p_deptid   employees.department_id%TYPE DEFAULT 30) IS

PROCEDURE audit_newemp IS
    PRAGMA AUTONOMOUS_TRANSACTION;
    user_id VARCHAR2(30) := USER;
BEGIN
    INSERT INTO log_newemp (entry_id, user_id, log_time, name)
        VALUES (log_newemp_seq.NEXTVAL, user_id,
sysdate,p_first_name||' '||p_last_name);
    COMMIT;
END audit_newemp;

BEGIN -- add_employee
    IF valid_deptid(p_deptid) THEN
        audit_newemp;
        INSERT INTO employees(employee_id, first_name, last_name,
email,
            job_id, manager_id, hire_date, salary, commission_pct,
department_id)
            VALUES (employees_seq.NEXTVAL, p_first_name, p_last_name,
p_email,
                p_job, p_mgr, TRUNC(SYSDATE), p_sal, p_comm, p_deptid);
    ELSE
        RAISE_APPLICATION_ERROR (-20204, 'Invalid department ID.
Try again.');
    END IF;
END add_employee;

```

```

PROCEDURE add_employee(
    p_first_name employees.first_name%TYPE,
    p_last_name employees.last_name%TYPE,
    p_deptid employees.department_id%TYPE) IS
    p_email employees.email%type;
BEGIN
    p_email := UPPER(SUBSTR(p_first_name, 1,
1) || SUBSTR(p_last_name, 1, 7));
    add_employee(p_first_name, p_last_name, p_email, p_deptid =>
p_deptid);
END;

PROCEDURE get_employee(
    p.empid IN employees.employee_id%TYPE,
    p_sal OUT employees.salary%TYPE,
    p_job OUT employees.job_id%TYPE) IS
BEGIN
    SELECT salary, job_id
    INTO p_sal, p_job
    FROM employees
    WHERE employee_id = p.empid;
END get_employee;

FUNCTION get_employee(p_emp_id employees.employee_id%type)
    return employees%rowtype IS
    rec_emp employees%rowtype;
BEGIN
    SELECT * INTO rec_emp
    FROM employees
    WHERE employee_id = p_emp_id;
    RETURN rec_emp;
END;

FUNCTION get_employee(p_family_name employees.last_name%type)
    return employees%rowtype IS
    rec_emp employees%rowtype;
BEGIN
    SELECT * INTO rec_emp
    FROM employees
    WHERE last_name = p_family_name;
    RETURN rec_emp;
END;

```

```

PROCEDURE get_employees(p_dept_id
employees.department_id%type) IS
BEGIN
    SELECT * BULK COLLECT INTO emp_table
    FROM EMPLOYEES
    WHERE department_id = p_dept_id;
END;

PROCEDURE init_departments IS
BEGIN
    FOR rec IN (SELECT department_id FROM departments)
    LOOP
        valid_departments(rec.department_id) := TRUE;
    END LOOP;
END;

PROCEDURE print_employee(p_rec_emp employees%rowtype) IS
BEGIN
    DBMS_OUTPUT.PUT_LINE(p_rec_emp.department_id || ' ' ||
                         p_rec_emp.employee_id|| ' ' ||
                         p_rec_emp.first_name|| ' ' ||
                         p_rec_emp.last_name|| ' ' ||
                         p_rec_emp.job_id|| ' ' ||
                         p_rec_emp.salary);
END;

PROCEDURE show_employees IS
BEGIN
    IF emp_table IS NOT NULL THEN
        DBMS_OUTPUT.PUT_LINE('Employees in Package table');
        FOR i IN 1 .. emp_table.COUNT
        LOOP
            print_employee(emp_table(i));
        END LOOP;
    END IF;
END show_employees;

FUNCTION valid_deptid(p_deptid IN
departments.department_id%TYPE)
    RETURN BOOLEAN IS
    v_dummy PLS_INTEGER;
BEGIN
    RETURN valid_departments.exists(p_deptid);

```

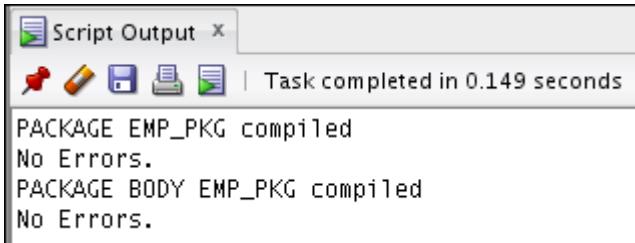
```

EXCEPTION
    WHEN NO_DATA_FOUND THEN
        RETURN FALSE;
END valid_deptid;

-- New set_salary procedure
PROCEDURE set_salary(p_jobid VARCHAR2, p_min_salary NUMBER) IS
    CURSOR cur_emp IS
        SELECT employee_id
        FROM employees
        WHERE job_id = p_jobid AND salary < p_min_salary;
BEGIN
    FOR rec_emp IN cur_emp
    LOOP
        UPDATE employees
            SET salary = p_min_salary
            WHERE employee_id = rec_emp.employee_id;
    END LOOP;
END set_salary;

BEGIN
    init_departments;
END emp_pkg;
/
SHOW ERRORS

```



- b. Erstellen Sie einen Row Trigger namens UPD\_MINSALARY\_TRG für die Tabelle JOBS, der die Prozedur EMP\_PKG.SET\_SALARY aufruft, wenn das Mindestgehalt in der Tabelle JOBS für eine angegebene Tätigkeits-ID aktualisiert wird.

**Entfernen Sie die Kommentarzeichen, und wählen Sie den Code unter der Aufgabe 1\_b. Um das Skript auszuführen, klicken Sie in der SQL Worksheet-Symbolleiste auf das Symbol "Run Script" (F5). Der Code und die Ergebnisse sind unten abgebildet.**

```

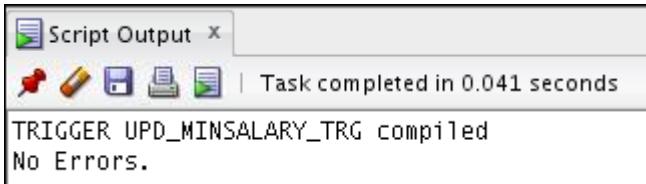
CREATE OR REPLACE TRIGGER upd_minsalary_trg
AFTER UPDATE OF min_salary ON JOBS
FOR EACH ROW
BEGIN

```

```

emp_pkg.set_salary(:new.job_id, :new.min_salary);
END;
/
SHOW ERRORS

```



- c. Erstellen Sie eine Abfrage, um die Personalnummer, den Nachnamen, die Tätigkeits-ID, das aktuelle Gehalt und das Mindestgehalt für Mitarbeiter anzuzeigen, die Programmierer mit der Tätigkeits-ID 'IT\_PROG' sind. Aktualisieren Sie anschließend das Mindestgehalt in der Tabelle JOBS durch eine Erhöhung um 1.000 \$. Was geschieht?

**Entfernen Sie die Kommentarzeichen, und wählen Sie den Code unter der Aufgabe 1\_c. Um das Skript auszuführen, klicken Sie in der SQL Worksheet-Symbolleiste auf das Symbol "Run Script" (F5). Der Code und die Ergebnisse sind unten abgebildet.**

```

SELECT employee_id, last_name, salary
FROM employees
WHERE job_id = 'IT_PROG';

UPDATE jobs
SET min_salary = min_salary + 1000
WHERE job_id = 'IT_PROG';

```

Script Output X | Task completed in 0.044 seconds

| EMPLOYEE_ID | LAST_NAME | SALARY |
|-------------|-----------|--------|
| 103         | Hunold    | 9000   |
| 104         | Ernst     | 7260   |
| 105         | Austin    | 4800   |
| 106         | Pataballa | 4800   |
| 107         | Lorentz   | 4200   |
| 214         | Beh       | 9000   |

6 rows selected

Error starting at line 235 in command:

```
UPDATE jobs
  SET min_salary = min_salary + 1000
 WHERE job_id = 'IT_PROG'
Error report:
SQL Error: ORA-04091: table ORA61.JOBS is mutating, trigger/function may not see it
ORA-06512: at "ORA61.CHECK_SALARY", line 5
ORA-06512: at "ORA61.SALARY_CHECK", line 1
ORA-04088: error during execution of trigger 'ORA61.SALARY_CHECK'
ORA-06512: at "ORA61.EMP_PKG", line 139
ORA-06512: at "ORA61.UPD_MINSALARY_TRG", line 2
ORA-04088: error during execution of trigger 'ORA61.UPD_MINSALARY_TRG'
04091. 00000 - "table %s.%s is mutating, trigger/function may not see it"
*Cause: A trigger (or a user defined plsql function that is referenced in
this statement) attempted to look at (or modify) a table that was
in the middle of being modified by the statement which fired it.
*Action: Rewrite the trigger (or function) so it does not read that table.
```

Die Aktualisierung der Spalte `min_salary` für die Tätigkeit 'IT\_PROG' wird nicht erfolgreich ausgeführt, weil der Trigger `UPD_MINSALARY_TRG` für die Tabelle `JOBS` versucht, die Mitarbeitergehälter durch Aufruf der Prozedur `EMP_PKG.SET_SALARY` zu aktualisieren. Die Prozedur `SET_SALARY` löst den Trigger `CHECK_SALARY_TRG` aus (eine kaskadierende Auswirkung). `CHECK_SALARY_TRG` ruft die Prozedur `CHECK_SALARY` auf, die versucht, die Daten der Tabelle `JOBS` zu lesen. Während die Prozedur `CHECK_SALARY` die Tabelle `JOBS` liest, erkennt sie die Exception für die sich verändernde Tabelle.

- Um das Problem der sich verändernden Tabelle zu beheben, erstellen Sie ein `JOBS_PKG`-Package, um im Speicher eine Kopie der Zeilen aus der Tabelle `JOBS` beizubehalten. Ändern Sie anschließend die Prozedur `CHECK_SALARY`. Verwenden Sie anstelle einer Abfrage für eine sich verändernde Tabelle die Packagedaten, um die Exception zu umgehen. Sie müssen jedoch einen Statement Trigger vom Typ `BEFORE INSERT OR UPDATE` für die Tabelle `EMPLOYEES` erstellen, um den Status des Packages `JOBS_PKG` zu initialisieren, bevor der Row Trigger `CHECK_SALARY` ausgelöst wird.

- a. Erstellen Sie ein neues Package namens JOBS\_PKG mit der folgenden Spezifikation:

```

PROCEDURE initialize;
FUNCTION get_minsalary(p_jobid VARCHAR2) RETURN NUMBER;
FUNCTION get_maxsalary(p_jobid VARCHAR2) RETURN NUMBER;
PROCEDURE set_minsalary(p_jobid VARCHAR2, min_salary
NUMBER);
PROCEDURE set_maxsalary(p_jobid VARCHAR2, max_salary
NUMBER);

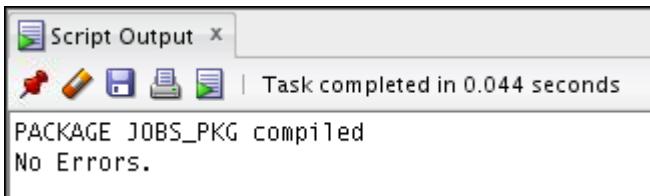
```

**Entfernen Sie die Kommentarzeichen, und wählen Sie den Code unter der Aufgabe2\_a, oder kopieren Sie den folgenden Code, und fügen Sie ihn in den SQL Worksheet-Bereich ein. Um das Skript auszuführen, klicken Sie in der SQL Worksheet-Symbolleiste auf das Symbol "Run Script" (F5). Der Code und die Ergebnisse sind unten abgebildet.**

```

CREATE OR REPLACE PACKAGE jobs_pkg IS
  PROCEDURE initialize;
  FUNCTION get_minsalary(p_jobid VARCHAR2) RETURN NUMBER;
  FUNCTION get_maxsalary(p_jobid VARCHAR2) RETURN NUMBER;
  PROCEDURE set_minsalary(p_jobid VARCHAR2, p_min_salary
NUMBER);
  PROCEDURE set_maxsalary(p_jobid VARCHAR2, p_max_salary
NUMBER);
END jobs_pkg;
/
SHOW ERRORS

```



- b. Implementieren Sie wie folgt den Body von JOBS\_PKG:

- 1) Deklarieren Sie eine private PL/SQL-INDEX BY-Tabelle namens jobs\_tab\_type, die durch einen Zeichenfolgetyp indiziert wird, der auf JOBS.JOB\_ID%TYPE basiert.
- 2) Deklarieren Sie eine private Variable namens jobstab auf der Basis von jobs\_tab\_type.
- 3) Die Prozedur INITIALIZE liest die Zeilen in der Tabelle JOBS mithilfe einer Cursorschleife und nutzt den Wert JOB\_ID für den Index jobstab, der der entsprechenden Zeile zugewiesen ist.
- 4) Die Funktion GET\_MINSALARY nutzt einen p\_jobid-Parameter als Index für jobstab und gibt min\_salary für dieses Element zurück.
- 5) Die Funktion GET\_MAXSALARY nutzt einen p\_jobid-Parameter als Index für jobstab und gibt max\_salary für dieses Element zurück.

- 6) Die Prozedur SET\_MINSALARY nutzt ihren p\_jobid-Parameter als Index für jobstab, um das Feld min\_salary des Elements auf den Wert im Parameter min\_salary einzustellen.
- 7) Die Prozedur SET\_MAXSALARY nutzt ihren p\_jobid-Parameter als Index für jobstab, um das Feld max\_salary des Elements auf den Wert im Parameter max\_salary einzustellen.

**Entfernen Sie die Kommentarzeichen, und wählen Sie den Code unter der Aufgabe 2\_b. Um das Skript auszuführen, klicken Sie in der SQL Worksheet-Symbolleiste auf das Symbol "Run Script" (F5). Der Code und die Ergebnisse sind unten abgebildet. Um den Packagebody zu kompilieren, klicken Sie im Object Navigator-Baum mit der rechten Maustaste auf den Namen des Packages oder des Packagebodys. Wählen Sie anschließend "Compile".**

```

CREATE OR REPLACE PACKAGE BODY jobs_pkg IS
  TYPE jobs_tab_type IS TABLE OF jobs%rowtype
    INDEX BY jobs.job_id%type;
  jobstab jobs_tab_type;

  PROCEDURE initialize IS
  BEGIN
    FOR rec_job IN (SELECT * FROM jobs)
    LOOP
      jobstab(rec_job.job_id) := rec_job;
    END LOOP;
  END initialize;

  FUNCTION get_minsalary(p_jobid VARCHAR2) RETURN NUMBER IS
  BEGIN
    RETURN jobstab(p_jobid).min_salary;
  END get_minsalary;

  FUNCTION get_maxsalary(p_jobid VARCHAR2) RETURN NUMBER IS
  BEGIN
    RETURN jobstab(p_jobid).max_salary;
  END get_maxsalary;

  PROCEDURE set_minsalary(p_jobid VARCHAR2, p_min_salary NUMBER)
  IS
  BEGIN
    jobstab(p_jobid).max_salary := p_min_salary;
  END set_minsalary;

  PROCEDURE set_maxsalary(p_jobid VARCHAR2, p_max_salary NUMBER)
  IS

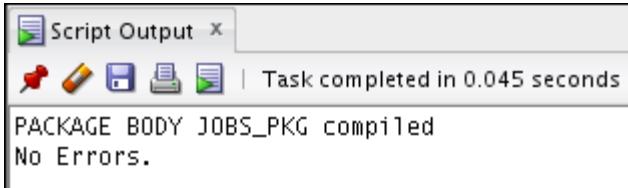
```

```

BEGIN
    jobstab(p_jobid).max_salary := p_max_salary;
END set_maxsalary;

END jobs_pkg;
/
SHOW ERRORS

```



- c. Kopieren Sie die Prozedur CHECK\_SALARY aus der 9. Übung, Aufgabe 1a, und ändern Sie den Code, indem Sie die Abfrage für die Tabelle JOBS durch Anweisungen ersetzen, um die lokalen Variablen minsal und maxsal mit Werten aus den JOBS\_PKG-Daten durch Aufrufen der entsprechenden GET\_\*SALARY-Funktionen einzustellen. Dieser Schritt sollte die Trigger-Exception für sich verändernde Tabellen beseitigen.

**Entfernen Sie die Kommentarzeichen, und wählen Sie den Code unter der Aufgabe 2\_c. Um das Skript auszuführen, klicken Sie in der SQL Worksheet-Symbolleiste auf das Symbol "Run Script" (F5). Der Code und die Ergebnisse sind unten abgebildet.**

```

CREATE OR REPLACE PROCEDURE check_salary (p_the_job VARCHAR2,
p_the_salary NUMBER) IS
    v_minsal jobs.min_salary%type;
    v_maxsal jobs.max_salary%type;
BEGIN

    -- Commented out to avoid mutating trigger exception on the
    -- JOBS table
    --SELECT min_salary, max_salary INTO v_minsal, v_maxsal
    --FROM jobs
    --WHERE job_id = UPPER(p_the_job);

    v_minsal := jobs_pkg.get_minsalary(UPPER(p_the_job));
    v_maxsal := jobs_pkg.get_maxsalary(UPPER(p_the_job));
    IF p_the_salary NOT BETWEEN v_minsal AND v_maxsal THEN
        RAISE_APPLICATION_ERROR(-20100,
            'Invalid salary $'||p_the_salary||'. ||
            'Salaries for job '|| p_the_job ||
            ' must be between $'|| v_minsal ||' and $' || v_maxsal);
    END IF;
END;
/

```

SHOW ERRORS

```
Script Output x
| Task completed in 0.045 seconds
PROCEDURE CHECK_SALARY compiled
No Errors.
```

- d. Implementieren Sie einen BEFORE INSERT OR UPDATE Statement Trigger namens INIT\_JOBPKG\_TRG, der die Syntax CALL verwendet, um die Prozedur JOBS\_PKG.INITIALIZE aufzurufen. Auf diese Weise stellen Sie sicher, dass der aktuelle Packagestatus vorliegt, bevor die DML-Vorgänge ausgeführt werden.

**Entfernen Sie die Kommentarzeichen, und wählen Sie den Code unter der Aufgabe 2\_d. Um das Skript auszuführen, klicken Sie in der SQL Worksheet-Symbolleiste auf das Symbol "Run Script" (F5). Der Code und die Ergebnisse sind unten abgebildet.**

```
CREATE OR REPLACE TRIGGER init_jobpkg_trg
BEFORE INSERT OR UPDATE ON jobs
CALL jobs_pkg.initialize
/
SHOW ERRORS
```

```
Script Output x
| Task completed in 0.042 seconds
TRIGGER INIT_JOBPKG_TRG compiled
No Errors.
```

- e. Testen Sie die Codeänderungen, indem Sie die Abfrage ausführen, um die Mitarbeiter anzuzeigen, die Programmierer sind. Setzen Sie dann eine UPDATE-Anweisung ab, um das Mindestgehalt der Tätigkeits-ID IT\_PROG in der Tabelle JOBS um 1.000 zu erhöhen. Lassen Sie darauf eine Abfrage für Mitarbeiter mit der Tätigkeits-ID IT\_PROG folgen, um die sich daraus ergebenden Änderungen zu prüfen. Bei welchen Mitarbeitern wurde das Gehalt auf das Mindestgehalt für ihre Tätigkeit eingestellt?

**Entfernen Sie die Kommentarzeichen, und wählen Sie den Code unter der Aufgabe 2\_e. Um das Skript auszuführen, klicken Sie in der SQL Worksheet-Symbolleiste auf das Symbol "Run Script" (F5). Der Code und die Ergebnisse sind unten abgebildet.**

```
SELECT employee_id, last_name, salary
FROM employees
WHERE job_id = 'IT_PROG';
```

```
UPDATE jobs
SET min_salary = min_salary + 1000
WHERE job_id = 'IT_PROG';
```

```
SELECT employee_id, last_name, salary
```

```
FROM employees
WHERE job_id = 'IT_PROG';

Script Output X
Task completed in 0.066 seconds

EMPLOYEE_ID LAST_NAME          SALARY
-----
103 Hunold           9000
104 Ernst            7260
105 Austin           4800
106 Pataballa        4800
107 Lorentz          4200
214 Beh              9000

6 rows selected

1 rows updated.

EMPLOYEE_ID LAST_NAME          SALARY
-----
103 Hunold           9000
104 Ernst            7260
105 Austin           5000
106 Pataballa        5000
107 Lorentz          5000
214 Beh              9000

6 rows selected
```

**Bei den Mitarbeitern mit den Nachnamen Austin, Pataballa und Lorentz wurde das Gehalt aktualisiert. Während dieses Prozesses ist keine Exception aufgetreten, und Sie haben eine Lösung für die Trigger-Exception für sich verändernde Tabellen implementiert.**

3. Da die Prozedur CHECK\_SALARY durch CHECK\_SALARY\_TRG ausgelöst wird, bevor ein Mitarbeiter hinzugefügt oder aktualisiert wird, müssen Sie prüfen, ob dies weiterhin wie gewohnt funktioniert.
  - a. Testen Sie dies, indem Sie mithilfe von EMP\_PKG.ADD\_EMPLOYEE einen neuen Mitarbeiter mit den folgenden Parametern hinzufügen: ('Steve', 'Morse', 'SMORSE' und sal => 6500). Was geschieht?

**Entfernen Sie die Kommentarzeichen, und wählen Sie den Code unter der Aufgabe 3\_a. Um das Skript auszuführen, klicken Sie in der SQL Worksheet-Symbolleiste auf das Symbol "Run Script" (F5). Der Code und die Ergebnisse sind unten abgebildet.**

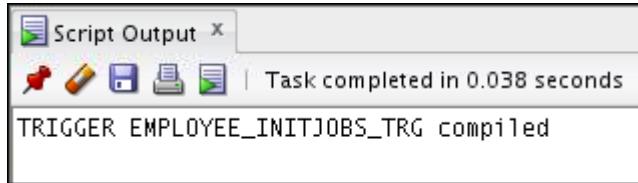
```
EXECUTE emp_pkg.add_employee('Steve', 'Morse', 'SMORSE', p_sal
=> 6500)
```

```
Script Output X
Task completed in 1.013 seconds

anonymous block completed
```

- b. So beheben Sie das Problem beim Hinzufügen oder Aktualisieren eines Mitarbeiters:
- 1) Erstellen Sie einen BEFORE INSERT OR UPDATE Statement Trigger namens EMPLOYEE\_INITJOBS\_TRG für die Tabelle EMPLOYEES, der die Prozedur JOBS\_PKG.INITIALIZE aufruft.
  - 2) Verwenden Sie im Triggerbody die Syntax CALL.
- Entfernen Sie die Kommentarzeichen, und wählen Sie den Code unter der Aufgabe 3\_b. Um das Skript auszuführen, klicken Sie in der SQL Worksheet-Symbolleiste auf das Symbol "Run Script" (F5). Der Code und die Ergebnisse sind unten abgebildet.**

```
CREATE TRIGGER employee_initjobs_trg
BEFORE INSERT OR UPDATE OF job_id, salary ON employees
CALL jobs_pkg.initialize
/
```



- c. Testen Sie den Trigger, indem Sie erneut den Mitarbeiter Steve Morse hinzufügen. Prüfen Sie, ob der Record in die Tabelle EMPLOYEES eingefügt wurde, indem Sie die Personalnummer, den Vor- und Nachnamen, das Gehalt, die Tätigkeits-ID und die Abteilungsnummer anzeigen.

```
EXECUTE emp_pkg.add_employee('Steve', 'Morse', 'SMORSE', p_sal =>
6500)
/
SELECT employee_id, first_name, last_name, salary, job_id,
department_id
FROM employees
WHERE last_name = 'Morse';
```

**Entfernen Sie die Kommentarzeichen, und wählen Sie den Code unter der Aufgabe 3\_c. Um das Skript auszuführen, klicken Sie in der SQL Worksheet-Symbolleiste auf das Symbol "Run Script" (F5). Der Code und die Ergebnisse sind unten abgebildet.**

Script Output | Task completed in 0.004 seconds

```
Error starting at line 384 in command:
EXECUTE emp_pkg.add_employee('Steve', 'Morse', 'SMORSE', p_sal => 6500)
Error report:
ORA-00001: unique constraint (ORA61.EMP_EMAIL_UK) violated
ORA-06512: at "ORA61.EMP_PKG", line 33
ORA-06512: at line 1
00001. 00000 - "unique constraint (%s.%s) violated"
*Cause: An UPDATE or INSERT statement attempted to insert a duplicate key.
        For Trusted Oracle configured in DBMS MAC mode, you may see
        this message if a duplicate entry exists at a different level.
*Action: Either remove the unique restriction or do not insert the key.
log_execution: Employee Inserted
```

| EMPLOYEE_ID | FIRST_NAME | LAST_NAME | SALARY | JOB_ID | DEPARTMENT_ID |
|-------------|------------|-----------|--------|--------|---------------|
| 215         | Steve      | Morse     | 6500   | SA_REP | 30            |

# **Übungen zu Lektion 11 – PL/SQL-Compiler**

## **Kapitel 11**

# Übungen zu Lektion 11 – Überblick

---

## Lektionsüberblick

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

### Hinweis:

1. Führen Sie vor Beginn dieser Übung das Skript  
`/home/oracle/plpu/code_ex/code_ex_scripts/clean_up_scripts/  
cleanup_11.sql` aus.
2. Wenn Sie einen Schritt in einer Übung ausgelassen haben, führen Sie das entsprechende Lösungsskript zu diesem Übungsschritt aus, bevor Sie mit dem nächsten Schritt oder der nächsten Übung fortfahren.

# Übung 1 zu Lektion 11 – PL/SQL-Compilerparameter und -warnungen verwenden

---

## Überblick

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

**Hinweis:** Führen Sie vor den folgenden Aufgaben das Skript `cleanup_11.sql` in `/home/oracle/plpu/code_ex/code_ex_scripts/clean_up_scripts/` aus.

## Aufgabe

1. Zeigen Sie folgende Informationen zu Compilerinitialisierungsparametern mit der Data Dictionary View `USER_PLSQL_OBJECT_SETTINGS` an. Beachten Sie die Einstellungen für das Objekt `ADD_JOB_HISTORY`.
 

**Hinweis:** Verwenden Sie das Symbol **Execute Statement** (F9), um die Ergebnisse in der Registerkarte **Results** anzuzeigen.

  - a. Objektname
  - b. Objekttyp
  - c. Kompilierungsmodus des Objekts
  - d. Optimierungsstufe für die Kompilierung
2. Ändern Sie den Parameter `PLSQL_CODE_TYPE`, um die native Kompilierung für Ihre Session zu aktivieren, und kompilieren Sie `ADD_DEPARTMENT`.
  - a. Führen Sie den Befehl `ALTER SESSION` aus, um die native Kompilierung für die Session zu aktivieren.
  - b. Kompilieren Sie die Prozedur `ADD_DEPARTMENT`.
  - c. Führen Sie den Code unter der 1. Aufgabe im Skript `sol_11` erneut aus. Beachten Sie den Parameter `PLSQL_CODE_TYPE`.
  - d. Wechseln Sie wie folgt in den interpretierten Kompilierungsmodus:
3. Wählen Sie **Tools > Preferences > Database > PL/SQL Compiler Options**, um alle Kategorien von Compilerwarnungen zu deaktivieren.
4. Bearbeiten und prüfen Sie das Skript `lab_11_04.sql`, und führen Sie es anschließend aus, um die Prozedur `UNREACHABLE_CODE` zu erstellen. Klicken Sie auf das Symbol **Run Script** (F5), um die Prozedur zu erstellen. Verwenden Sie zur Kompilierung der Prozedur den Prozedurnamen aus dem Object Navigation-Baum.
5. Welche Compilerwarnungen werden gegebenenfalls in der Registerkarte **Compiler – Log** angezeigt?
6. Aktivieren Sie im Fenster **Preferences** alle Compilerwarnmeldungen für diese Session.
7. Rekomplizieren Sie die Prozedur `UNREACHABLE_CODE` mithilfe des Object Navigator-Baumes. Welche Compilerwarnmeldungen werden gegebenenfalls angezeigt?

8. Zeigen Sie mit der Data Dictionary View `USER_ERRORS` wie folgt Details zu den Compilerwarnmeldungen an:
9. Erstellen Sie ein Skript namens `warning_msgs`, das die Kategorien für folgende Compiler-Warnmeldungnummern mithilfe der Packages `EXECUTE DBMS_OUTPUT` und `DBMS_WARNING` identifiziert: 5050, 6075 und 7100. Aktivieren Sie `SERVEROUTPUT`, bevor Sie das Skript ausführen.

## Übung 1 zu Lektion 11 – Lösung: PL/SQL-Compilerparameter und -warnungen verwenden

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

- Zeigen Sie folgende Informationen zu Compilerinitialisierungsparametern mit der Data Dictionary View `USER_PLSQL_OBJECT_SETTINGS` an. Beachten Sie die Einstellungen für das Objekt `ADD_JOB_HISTORY`.

**Hinweis:** Verwenden Sie das Symbol **Execute Statement** (F9), um die Ergebnisse in der Registerkarte **Results** anzuzeigen.

- Objektname
- Objekttyp
- Kompilierungsmodus des Objekts
- Optimierungsstufe für die Kompilierung

**Entfernen Sie die Kommentarzeichen, und wählen Sie den Code unter der 1. Aufgabe. Um die Abfrage auszuführen, klicken Sie in der SQL Worksheet-Symbolleiste auf das Symbol "Run Script" (F9). Der Code und Beispieldaten sind unten abgebildet.**

```
SELECT name, type, plsql_code_type as code_type,
       plsql_optimize_level as opt_lvl
  FROM user_plsql_object_settings;
```

| NAME                  | TYPE      | CODE_TYPE   | OPT_LVL |
|-----------------------|-----------|-------------|---------|
| 1 ADD_COL             | PROCEDURE | INTERPRETED | 0       |
| 2 ADD_DEPARTMENT      | PROCEDURE | INTERPRETED | 0       |
| 3 ADD_DEPARTMENT_NOEX | PROCEDURE | INTERPRETED | 0       |

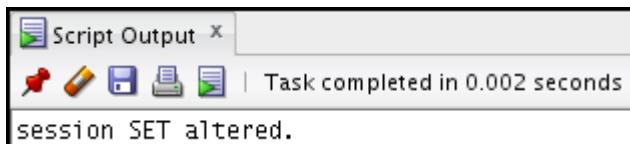
...

**Hinweis:** In diesem Schritt liegt der Fokus auf der Prozedur `ADD_DEPARTMENT`. Eventuelle Abweichungen im Screenshot können ignoriert werden.

- Ändern Sie den Parameter `PLSQL_CODE_TYPE`, um die native Kompilierung für Ihre Session zu aktivieren, und kompilieren Sie `ADD_DEPARTMENT`.
  - Führen Sie den Befehl `ALTER SESSION` aus, um die native Kompilierung für die Session zu aktivieren.

**Entfernen Sie die Kommentarzeichen, und wählen Sie den Code unter der Aufgabe 2\_a. Um die Abfrage auszuführen, klicken Sie in der SQL Worksheet-Symbolleiste auf das Symbol "Run Script" (F5). Der Code und die Ergebnisse sind unten abgebildet.**

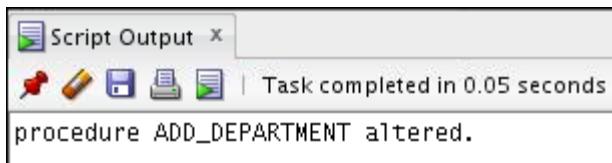
```
ALTER SESSION SET PLSQL_CODE_TYPE = 'NATIVE';
```



- b. Kompilieren Sie die Prozedur ADD\_DEPARTMENT.

**Entfernen Sie die Kommentarzeichen, und wählen Sie den Code unter der Aufgabe 2\_b. Um die Abfrage auszuführen, klicken Sie in der SQL Worksheet-Symbolleiste auf das Symbol "Run Script" (F5). Der Code und die Ergebnisse sind unten abgebildet.**

```
ALTER PROCEDURE add_department COMPILE;
```



- c. Führen Sie den Code unter der 1. Aufgabe im Skript sol\_11.sql erneut aus, indem Sie im SQL Worksheet auf das Symbol **Execute Statement** (F9) klicken. Beachten Sie den Parameter PLSQL\_CODE\_TYPE.

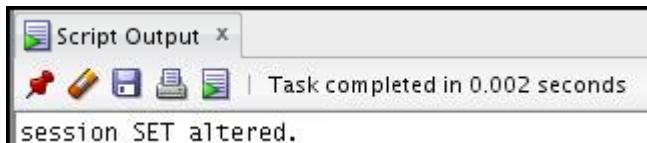
```
SELECT name, type, plsql_code_type as code_type,
plsql_optimize_level as opt_lvl
FROM user_plsql_object_settings;
```

| NAME                  | TYPE      | CODE_TYPE   | OPT_LVL |
|-----------------------|-----------|-------------|---------|
| 1 ADD_COL             | PROCEDURE | INTERPRETED | 0       |
| 2 ADD_DEPARTMENT      | PROCEDURE | NATIVE      | 1       |
| 3 ADD_DEPARTMENT_NOEX | PROCEDURE | INTERPRETED | 0       |

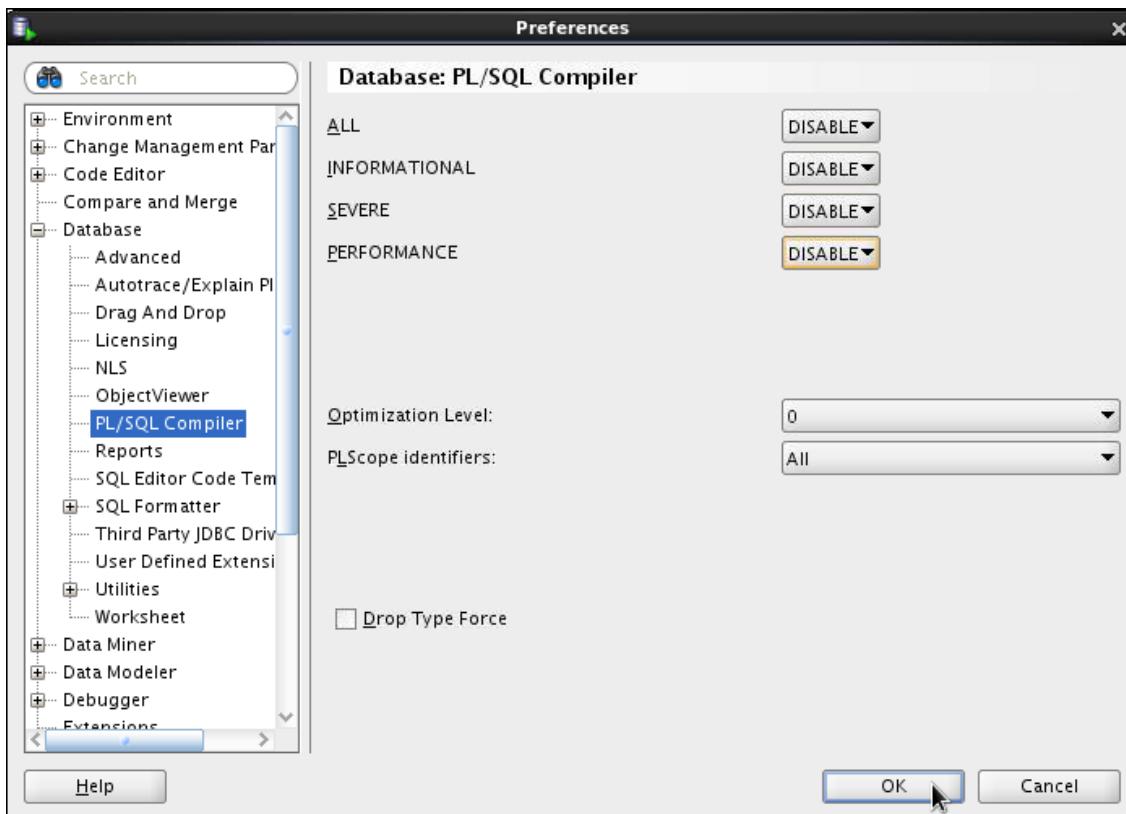
...

- d. Wechseln Sie wie folgt in den interpretierten Kompilierungsmodus:

```
ALTER SESSION SET PLSQL_CODE_TYPE = 'INTERPRETED';
```



3. Wählen Sie **Tools > Preferences > Database > PL/SQL Compiler Options**, um alle Kategorien von Compilerwarnungen zu deaktivieren.

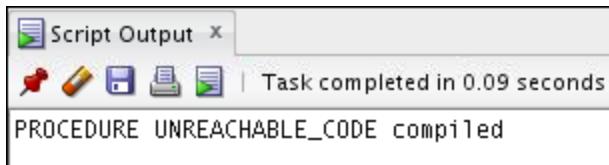


**Wählen Sie DISABLE für alle vier PL/SQL-Compilerwarnkategorien, und klicken Sie dann auf "OK".**

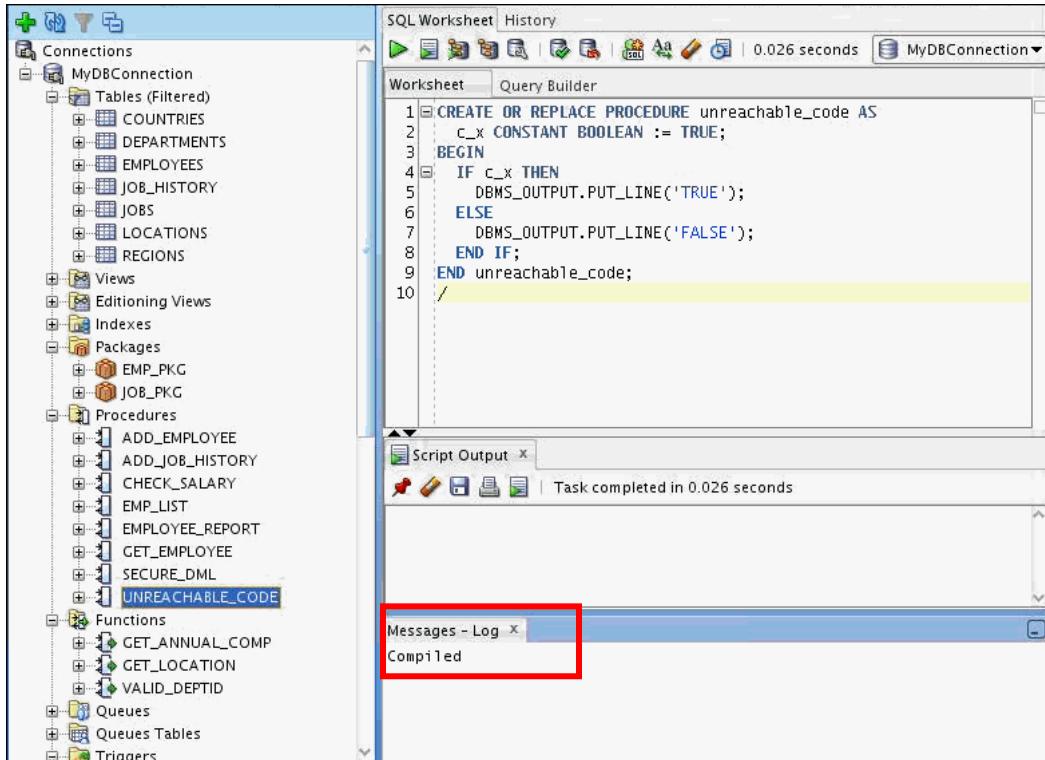
4. Bearbeiten und prüfen Sie das Skript `lab_11_04.sql`, und führen Sie es anschließend aus, um die Prozedur `UNREACHABLE_CODE` zu erstellen. Klicken Sie auf das Symbol **Run Script** (F5), um die Prozedur zu erstellen und zu komplizieren.

**Entfernen Sie die Kommentarzeichen, und wählen Sie den Code unter der 4. Aufgabe. Um die Abfrage auszuführen, klicken Sie in der SQL Worksheet-Symboleiste auf das Symbol "Run Script" (F5). Der Code und die Ergebnisse sind unten abgebildet.**

```
CREATE OR REPLACE PROCEDURE unreachable_code AS
  c_x CONSTANT BOOLEAN := TRUE;
BEGIN
  IF c_x THEN
    DBMS_OUTPUT.PUT_LINE('TRUE');
  ELSE
    DBMS_OUTPUT.PUT_LINE('FALSE');
  END IF;
END unreachable_code;
/
```



**Um Compilerwarnungsfehler anzuzeigen, klicken Sie im Object Navigation-Baum im Knoten "Procedures" mit der rechten Maustaste auf den Namen der Prozedur und klicken dann auf "Compile".**

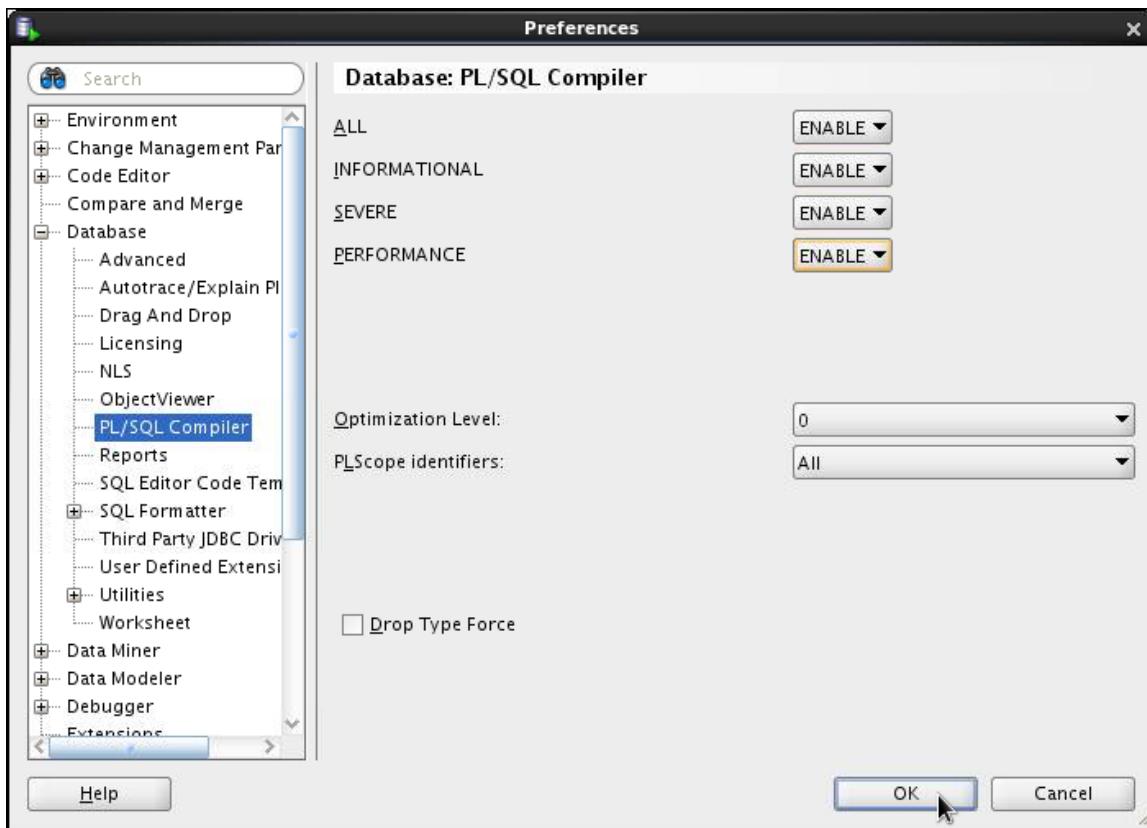


### Hinweis

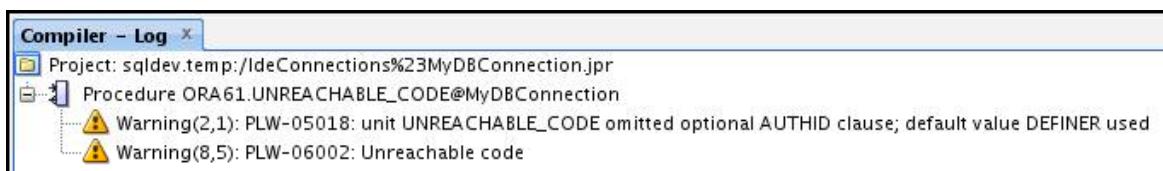
- Wenn die Prozedur nicht im Object Navigation-Baum angezeigt wird, klicken Sie in der Registerkarte **Connections** auf das Symbol **Refresh**.
  - Vergewissern Sie sich, dass die Registerkarte **Messages – Log** angezeigt wird. (Wählen Sie in der Menüleiste **View > Log**.)
5. Welche Compilerwarnungen werden gegebenenfalls in der Registerkarte **Compiler – Log** angezeigt?

**Beachten Sie, dass Sie in der Registerkarte "Messages – Log" die Meldung "Compiled" erhalten. Sie erhalten keine Warnmeldungen, weil Sie die Compilerwarnungen im 3. Schritt deaktiviert haben.**

6. Aktivieren Sie im Fenster **Preferences** alle Compilerwarnmeldungen für diese Session.  
**Wählen Sie ENABLE für alle vier PL/SQL-Compilerwarnungen, und klicken Sie dann auf "OK".**



7. Rekompilieren Sie die Prozedur UNREACHABLE\_CODE mithilfe des Object Navigator-Baumes. Welche Compilerwarnmeldungen werden gegebenenfalls angezeigt?  
**Klicken Sie im Object Navigator-Baum mit der rechten Maustaste auf den Namen der Prozedur, und wählen Sie "Compile". Beachten Sie die in der Registerkarte "Compiler – Log" angezeigten Meldungen.**



**Hinweis:** Die folgenden Warnungen sind in bestimmten Versionen von SQL Developer normal. Sie erhalten die Warnungen, weil Ihre Version von SQL Developer noch den in der Oracle 11g-Datenbank verworfenen Parameter PLSQL\_DEBUG verwendet.

```
Warning (1) :PLW-06015:parameter PLSQL_DEBUG is deprecated ; use
PLSQL_OPTIMIZE_LEVEL=1
```

```
Warning (1) :PLW-06013:deprecated parameter PLSQL_DEBUG forces
PLSQL_OPTIMIZE_LEVEL<=1
```

8. Zeigen Sie mit der Data Dictionary View `USER_ERRORS` wie folgt Details zu den Compilerwarnmeldungen an:

```
DESCRIBE user_errors
```

The screenshot shows the 'Script Output' tab of the Oracle SQL Worksheet. The command `DESCRIBE user_errors` has been run, and the results are displayed in a table format. The table has columns for Name, Null, and Type. The data rows are:

| Name           | Null     | Type           |
|----------------|----------|----------------|
| NAME           | NOT NULL | VARCHAR2(128)  |
| TYPE           |          | VARCHAR2(12)   |
| SEQUENCE       | NOT NULL | NUMBER         |
| LINE           | NOT NULL | NUMBER         |
| POSITION       | NOT NULL | NUMBER         |
| TEXT           | NOT NULL | VARCHAR2(4000) |
| ATTRIBUTE      |          | VARCHAR2(9)    |
| MESSAGE_NUMBER |          | NUMBER         |

```
SELECT *
FROM user_errors;
```

The screenshot shows the 'Query Result' tab of the Oracle SQL Worksheet. The command `SELECT * FROM user_errors` has been run, and the results are displayed in a table format. The table has columns for NAME, TYPE, SEQUENCE, LINE, POSITION, TEXT, ATTRIBUTE, and MESSAGE\_NUMBER. The data rows are:

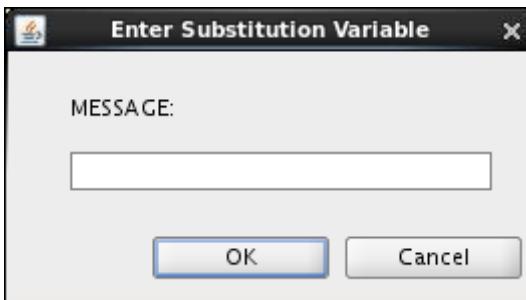
| NAME                         | TYPE | SEQUENCE | LINE | POSITION                                             | TEXT    | ATTRIBUTE | MESSAGE_NUMBER |
|------------------------------|------|----------|------|------------------------------------------------------|---------|-----------|----------------|
| 1 UNREACHABLE_CODE PROCEDURE | 1    | 1        | 1    | PLW-05018: unit UNREACHABLE_CODE omitted optional... | WARNING | 5018      |                |
| 2 UNREACHABLE_CODE PROCEDURE | 2    | 7        | 5    | PLW-06002: Unreachable code                          | WARNING | 6002      |                |

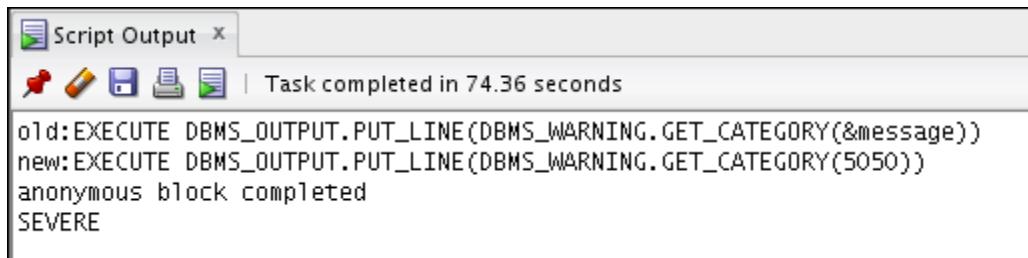
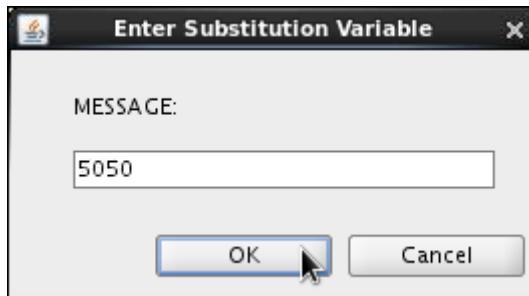
**Hinweis:** Die Ausgabe wurde in der Registerkarte **Results** angezeigt, weil die Anweisung `SELECT` mit der Taste F9 ausgeführt wurde. Die tatsächlichen Ergebnisse der Anweisung `SELECT` hängen von der Anzahl der Fehler in Ihrer Session ab.

9. Erstellen Sie ein Skript namens `warning_msgs`, das die Kategorien für folgende Compiler-Warnmeldungsnummern mithilfe der Packages `EXECUTE DBMS_OUTPUT` und `DBMS_WARNING` identifiziert: 5050, 6075 und 7100. Aktivieren Sie `SERVEROUTPUT`, bevor Sie das Skript ausführen.

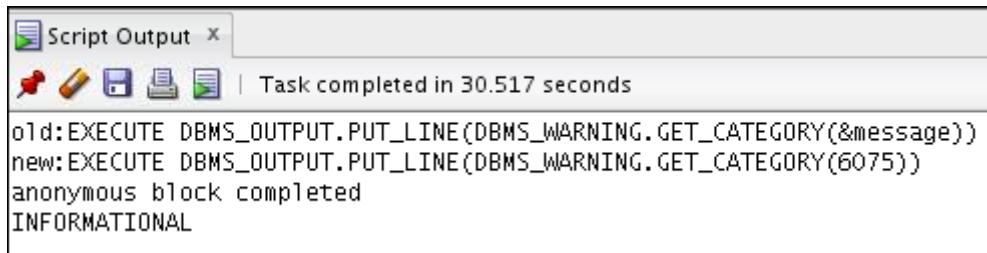
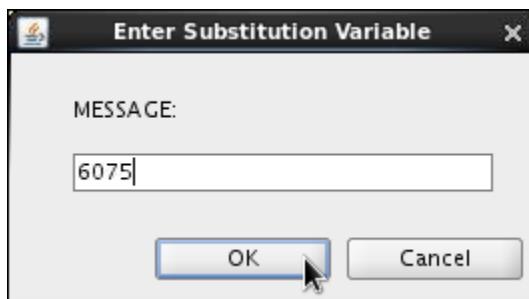
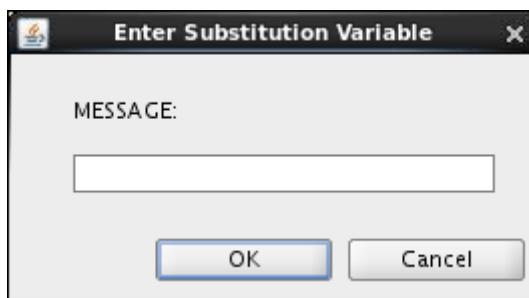
**Entfernen Sie die Kommentarzeichen, und wählen Sie den Code unter der 9. Aufgabe. Um die Abfrage auszuführen, klicken Sie in der SQL Worksheet-Symboleiste auf das Symbol "Run Script" (F5). Der Code und die Ergebnisse sind unten abgebildet.**

```
EXECUTE
DBMS_OUTPUT.PUT_LINE(DBMS_WARNING.GET_CATEGORY(&message));
```

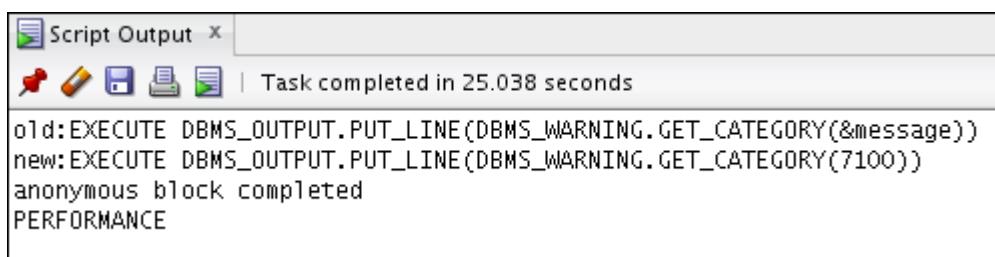
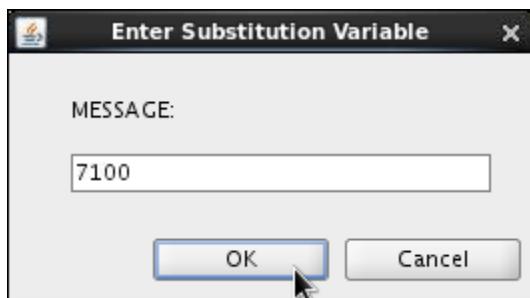
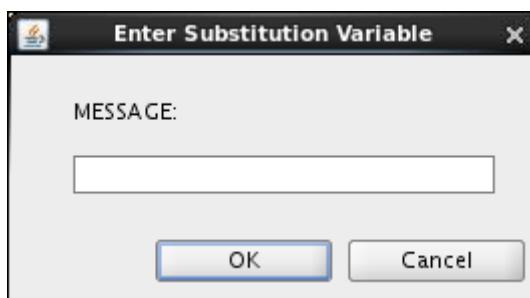




```
EXECUTE
DBMS_OUTPUT.PUT_LINE(DBMS_WARNING.GET_CATEGORY(&message));
```



```
EXECUTE  
DBMS_OUTPUT.PUT_LINE(DBMS_WARNING.GET_CATEGORY(&message)) ;
```



# **Übungen zu Lektion 12 – Abhängigkeiten verwalten**

## **Kapitel 12**

# Übungen zu Lektion 12 – Überblick

---

## Lektionsüberblick

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

### Hinweis:

1. Führen Sie das Skript  
`/home/oracle/plpu/code_ex/code_ex_scripts/clean_up_scripts/  
cleanup_12.sql` aus, bevor Sie mit der Übung beginnen.
2. Wenn Sie einen Schritt in einer Übung ausgelassen haben, führen Sie das entsprechende Lösungsskript zu diesem Übungsschritt aus, bevor Sie mit dem nächsten Schritt oder der nächsten Übung fortfahren.

# Übung 1 zu Lektion 12 – Abhängigkeiten im Schema verwalten

---

## Überblick

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

**Hinweis:** Führen Sie zunächst das Skript cleanup\_12.sql unter /home/oracle/plpu/code\_ex/code\_ex\_scripts/clean\_up\_scripts/ aus.

## Aufgabe

1. Erstellen Sie eine Baumstruktur mit allen Abhängigkeiten der Prozedur add\_employee und der Funktion valid\_deptid.

**Hinweis:** Erstellen Sie zunächst die Prozedur add\_employee und die Funktion valid\_deptid, wie in den Übungen zur 3. Lektion ("Funktionen erstellen und Unterprogramme debuggen") beschrieben.

- a. Laden Sie das Skript utldtree.sql aus dem Ordner /home/oracle/labs/plpu/labs, und führen Sie es aus.
- b. Führen Sie die Prozedur deptree\_fill für die Prozedur add\_employee aus.
- c. Fragen Sie die View IDEPTREE ab, um die Ergebnisse anzuzeigen.
- d. Führen Sie die Prozedur deptree\_fill für die Funktion valid\_deptid aus.
- e. Fragen Sie die View IDEPTREE ab, um die Ergebnisse anzuzeigen.

### Führen Sie die folgende Übung durch, falls Sie noch Zeit haben:

2. Ändern Sie ungültige Objekte dynamisch wieder in gültige Objekte.
  - a. Erstellen Sie eine Kopie Ihrer Tabelle EMPLOYEES, und nennen Sie sie EMPS.
  - b. Ändern Sie die Tabelle EMPLOYEES, indem Sie die Spalte TOTSAL mit dem Datentyp NUMBER(9, 2) hinzufügen.
  - c. Erstellen und speichern Sie eine Abfrage, um Name, Typ und Status aller ungültigen Objekte anzuzeigen.
  - d. Fügen Sie in das Package compile\_pkg (erstellt in den Übungen zur 7. Lektion, "Dynamisches SQL") eine Prozedur recompile ein, die alle ungültigen Prozeduren, Funktionen und Packages in Ihrem Schema rekompiliert. Verwenden Sie natives dynamisches SQL, um den ungültigen Objekttyp zu ändern und zu kompilieren.
  - e. Führen Sie die Prozedur compile\_pkg.recompile aus.
  - f. Um den Wert der Spalte STATUS zu prüfen, führen Sie die Skriptdatei aus, die Sie in Schritt 2c erstellt haben. Sind noch immer Objekte mit dem Status INVALID vorhanden?

## Übung 1 zu Lektion 12 – Lösung: Abhängigkeiten im Schema verwalten

---

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

1. Erstellen Sie eine Baumstruktur mit allen Abhängigkeiten der Prozedur add\_employee und der Funktion valid\_deptid.

**Hinweis:** add\_employee und valid\_deptid wurden in den Übungen zur 3. Lektion ("Funktionen erstellen und Unterprogramme debuggen") erstellt. Um die Prozedur add\_employee und die Funktion valid\_deptid zu erstellen, führen Sie folgenden Code aus:

```

CREATE OR REPLACE PROCEDURE add_employee(
    p_first_name employees.first_name%TYPE,
    p_last_name  employees.last_name%TYPE,
    p_email      employees.email%TYPE,
    p_job        employees.job_id%TYPE          DEFAULT 'SA_REP',
    p_mgr        employees.manager_id%TYPE       DEFAULT 145,
    p_sal        employees.salary%TYPE          DEFAULT 1000,
    p_comm       employees.commission_pct%TYPE  DEFAULT 0,
    p_deptid     employees.department_id%TYPE   DEFAULT 30) IS
BEGIN
    IF valid_deptid(p_deptid) THEN
        INSERT INTO employees(employee_id, first_name, last_name,
                           email,
                           job_id, manager_id, hire_date, salary, commission_pct,
                           department_id)
        VALUES (employees_seq.NEXTVAL, p_first_name, p_last_name,
                p_email,
                p_job, p_mgr, TRUNC(SYSDATE), p_sal, p_comm, p_deptid);
    ELSE
        RAISE_APPLICATION_ERROR (-20204, 'Invalid department ID. Try again.');
    END IF;
END add_employee;
/
CREATE OR REPLACE FUNCTION valid_deptid(
    p_deptid IN departments.department_id%TYPE)

```

---

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

```

    RETURN BOOLEAN IS
        v_dummy    PLS_INTEGER;

BEGIN
    SELECT 1
    INTO   v_dummy
    FROM   departments
    WHERE  department_id = p_deptid;
    RETURN TRUE;
EXCEPTION
    WHEN NO_DATA_FOUND THEN
        RETURN FALSE;
END valid_deptid;
/

```

- a. Laden Sie das Skript `utldtree.sql` aus dem Ordner  
`/home/oracle/labs/plpu/labs`, und führen Sie es aus.

**Öffnen Sie das Skript `/home/oracle/labs/plpu/solns/utldtree.sql`. Um das Skript auszuführen, klicken Sie in der SQL Worksheet-Symbolleiste auf das Symbol "Run Script" (F5). Der Code und die Ergebnisse sind unten abgebildet.**

```

Rem
Rem $Header: utldtree.sql,v 3.2 2012/11/21 16:24:44 RKOOI Stab $
Rem
Rem Copyright (c) 1991 by Oracle Corporation
Rem NAME
Rem deptree.sql - Show objects recursively dependent on
Rem given object
Rem DESCRIPTION
Rem This procedure, view and temp table will allow you to see
Rem all objects that are (recursively) dependent on the given
Rem object.
Rem Note: you will only see objects for which you have
Rem permission.
Rem Examples:
Rem execute deptree_fill('procedure', 'scott', 'billing');
Rem select * from deptree order by seq#;
Rem
Rem execute deptree_fill('table', 'scott', 'emp');
Rem select * from deptree order by seq#;
Rem

```

```

Rem   execute deptree_fill('package body', 'scott',
Rem   'accts_payable');
Rem   select * from deptree order by seq#;
Rem
Rem   A prettier way to display this information than
Rem   select * from deptree order by seq#;
Rem is
Rem   select * from ideptree;
Rem   This shows the dependency relationship via indenting.
Rem   Notice that no order by clause is needed with ideptree.
Rem   RETURNS
Rem
Rem   NOTES
Rem   Run this script once for each schema that needs this
Rem   utility.
Rem   MODIFIED      (MM/DD/YY)
Rem   rkooi          10/26/92 - owner -> schema for SQL2
Rem   glumpkin       10/20/92 - Renamed from DEPTREE.SQL
Rem   rkooi          09/02/92 - change ORU errors
Rem   rkooi          06/10/92 - add rae errors
Rem   rkooi          01/13/92 - update for sys vs. regular user
Rem   rkooi          01/10/92 - fix ideptree
Rem   rkooi          01/10/92 - Better formatting, add ideptree
view
Rem   rkooi          12/02/91 - deal with cursors
Rem   rkooi          10/19/91 - Creation

DROP SEQUENCE deptree_seq
/
CREATE SEQUENCE deptree_seq cache 200
-- cache 200 to make sequence faster

/
DROP TABLE deptree_temptab
/
CREATE TABLE deptree_temptab
(
    object_id          number,
    referenced_object_id number,
    nest_level         number,
    seq#               number
)

```

```

/
CREATE OR REPLACE PROCEDURE deptree_fill (type char, schema
char, name char) IS
    obj_id number;
BEGIN
    DELETE FROM deptree_temptab;
    COMMIT;
    SELECT object_id INTO obj_id FROM all_objects
        WHERE owner = upper(deptree_fill.schema)

    AND    object_name  = upper(deptree_fill.name)
        AND    object_type = upper(deptree_fill.type);
    INSERT INTO deptree_temptab
        VALUES(obj_id, 0, 0, 0);
    INSERT INTO deptree_temptab
        SELECT object_id, referenced_object_id,
            level, deptree_seq.nextval
        FROM public_dependency
        CONNECT BY PRIOR object_id = referenced_object_id
        START WITH referenced_object_id = deptree_fill.obj_id;
EXCEPTION
    WHEN no_data_found then
        raise_application_error(-20000, 'ORU-10013: ' ||
            type || ' ' || schema || '.' || name || ' was not
            found.');
    END;
/

DROP VIEW deptree
/

SET ECHO ON

REM This view will succeed if current user is sys. This view
REM shows which shared cursors depend on the given object. If
REM the current user is not sys, then this view get an error
REM either about lack of privileges or about the non-existence
REM of REM table x$kglxss.

SET ECHO OFF
CREATE VIEW sys.deptree
    (nested_level, type, schema, name, seq#)
AS

```

```

      SELECT d.nest_level, o.object_type, o.owner, o.object_name,
d.seq#
      FROM deptree temptab d, dba_objects o
      WHERE d.object_id = o.object_id (+)
UNION ALL
      SELECT d.nest_level+1, 'CURSOR', '<shared>',
' "' || c.kglnaobj || ''', d.seq#+.5
      FROM deptree temptab d, x$kgldp k, x$kglob g, obj$ o, user$ u,
x$kglob c,
          x$kglx s a
      WHERE d.object_id = o.obj#
      AND    o.name = g.kglnaobj
      AND    o.owner# = u.user#
      AND    u.name = g.kglnaown
      AND    g.kglhdadr = k.kglrfhdl
      AND    k.kglhdadr = a.kglhdadr -- make sure it is not a
transitive
      AND    k.kgldepno = a.kglxsdep -- reference, but a direct
one
      AND    k.kglhdadr = c.kglhdadr
      AND    c.kglhdnsp = 0 -- a cursor
/
SET ECHO ON

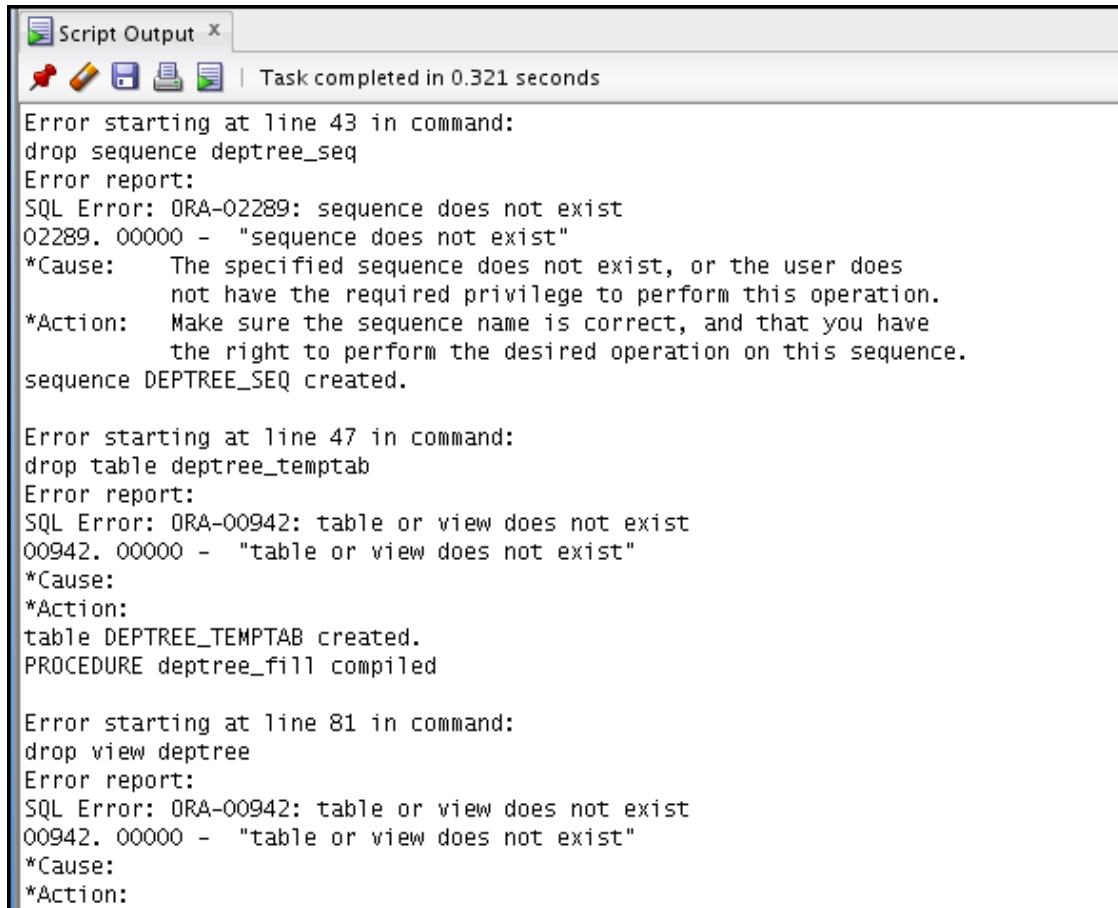
REM This view will succeed if current user is not sys. This view
REM does *not* show which shared cursors depend on the given
REM object.
REM If the current user is sys then this view will get an error
REM indicating that the view already exists (since prior view
REM create will have succeeded).

SET ECHO OFF
CREATE VIEW deptree
  (nested_level, type, schema, name, seq#)
AS
  select d.nest_level, o.object_type, o.owner, o.object_name,
d.seq#
  FROM deptree temptab d, all_objects o
  WHERE d.object_id = o.object_id (+)
/
DROP VIEW ideptree
/
CREATE VIEW ideptree (dependencies)

```

```
AS
    SELECT lpad(' ', 3*(max(nested_level))) || max(nvl(type, '<no
permission>'))
        || ' ' || schema || decode(type, NULL, '', '.') || name
   FROM deptree
 GROUP BY seq# /* So user can omit sort-by when selecting from
ideptree */
/

```



The screenshot shows the 'Script Output' window in Oracle SQL Developer. The window title is 'Script Output'. It displays the output of a script that attempts to drop sequences, tables, and views that do not exist. The output includes error messages, cause, and action details for each command.

```
Script Output | Task completed in 0.321 seconds
Error starting at line 43 in command:
drop sequence deptree_seq
Error report:
SQL Error: ORA-02289: sequence does not exist
02289. 00000 - "sequence does not exist"
*Cause: The specified sequence does not exist, or the user does
not have the required privilege to perform this operation.
*Action: Make sure the sequence name is correct, and that you have
the right to perform the desired operation on this sequence.
sequence DEPTREE_SEQ created.

Error starting at line 47 in command:
drop table deptree temptab
Error report:
SQL Error: ORA-00942: table or view does not exist
00942. 00000 - "table or view does not exist"
*Cause:
*Action:
table DEPTREE_TEMPTAB created.
PROCEDURE deptree_fill compiled

Error starting at line 81 in command:
drop view deptree
Error report:
SQL Error: ORA-00942: table or view does not exist
00942. 00000 - "table or view does not exist"
*Cause:
*Action:
```

```

> REM This view will succeed if current user is sys. This view shows
> REM which shared cursors depend on the given object. If the current
> REM user is not sys, then this view get an error either about lack
> REM of privileges or about the non-existence of table x$kg1xs.

Error starting at line 92 in command:
create view sys.deptree
  (nested_level, type, schema, name, seq#)
as
  select d.nest_level, o.object_type, o.owner, o.object_name, d.seq#
    from deptree temptab d, dba_objects o
   where d.object_id = o.object_id (+)
union all
  select d.nest_level+1, 'CURSOR', '<shared>', ''||c.kglnaobj||'', d.seq#+.5
    from deptree temptab d, x$kgldp k, x$kglob g, obj$ o, user$ u, x$kglob c,
         x$kg1xs a
   where d.object_id = o.obj#
     and o.name = g.kglnaobj
     and o.owner# = u.user#
     and u.name = g.kglnaown
     and g.kglhdadr = k.kglrfhd1
     and k.kglhdadr = a.kglhdadr /* make sure it is not a transitive */
     and k.kgldepno = a.kglxdep /* reference, but a direct one */
     and k.kglhdadr = c.kglhdadr
     and c.kglhdnsp = 0 /* a cursor */
Error at Command Line:96 Column:7
Error report:
SQL Error: ORA-00942: table or view does not exist
00942. 00000 - "table or view does not exist"

```

```

00942. 00000 - "table or view does not exist"
*Cause:
>Action:
> REM This view will succeed if current user is not sys. This view
> REM does *not* show which shared cursors depend on the given object.
> REM If the current user is sys then this view will get an error
> REM indicating that the view already exists (since prior view create
> REM will have succeeded).
view DEPTREE created.
Error starting at line 130 in command:
drop view ideptree
Error report:
SQL Error: ORA-00942: table or view does not exist
00942. 00000 - "table or view does not exist"
*Cause:
>Action:
view IDEPTREE created.
sequence DEPTREE_SEQ dropped.
sequence DEPTREE_SEQ created.
table DEPTREE_TEMPTAB dropped.
table DEPTREE_TEMPTAB created.
PROCEDURE deptree_fill compiled
view DEPTREE dropped.

```

```
> REM This view will succeed if current user is sys. This view shows
> REM which shared cursors depend on the given object. If the current
> REM user is not sys, then this view get an error either about lack
> REM of privileges or about the non-existence of table x$kg1xs.

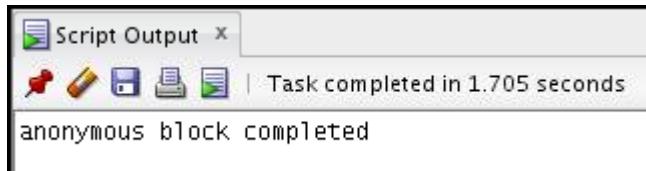
Error starting at line 92 in command:
create view sys.deptree
  (nested_level, type, schema, name, seq#)
as
  select d.nest_level, o.object_type, o.owner, o.object_name, d.seq#
  from deptree temptab d, dba_objects o
  where d.object_id = o.object_id (+)
union all
  select d.nest_level+1, 'CURSOR', '<shared>', ''||c.kglnaobj||'', d.seq#+.5
  from deptree temptab d, x$kgldp k, x$kglob g, obj$ o, user$ u, x$kglob c,
       x$kg1xs a
  where d.object_id = o.obj#
  and   o.name = g.kglnaobj
  and   o.owner# = u.user#
  and   u.name = g.kglnaown
  and   g.kglhdadr = k.kglrfhdl
  and   k.kglhdadr = a.kglhdadr /* make sure it is not a transitive */
  and   k.kgldepno = a.kg1xsdep /* reference, but a direct one */
  and   k.kglhdadr = c.kg1hdadr
  and   c.kglhdnsp = 0 /* a cursor */
```

```
Error at Command Line:96 Column:7
Error report:
SQL Error: ORA-00942: table or view does not exist
00942. 00000 -  "table or view does not exist"
*Cause:
*Action:
> REM This view will succeed if current user is not sys. This view
> REM does *not* show which shared cursors depend on the given object.
> REM If the current user is sys then this view will get an error
> REM indicating that the view already exists (since prior view create
> REM will have succeeded).
view DEPTREE created.
view IDEPTREE dropped.
view IDEPTREE created.
```

- b. Führen Sie die Prozedur deptree\_fill für die Prozedur add\_employee aus.

**Öffnen Sie das Skript /home/oracle/labs/plpu/solns/sol\_12.sql.  
Entfernen Sie die Kommentarzeichen, und wählen Sie den Code unter der Aufgabe 1\_b. Um das Skript auszuführen, klicken Sie in der SQL Worksheet-Symbolleiste auf das Symbol "Run Script" (F5). Der Code und die Ergebnisse sind unten abgebildet.**

```
EXECUTE deptree_fill('PROCEDURE', USER, 'add_employee')
```



- c. Fragen Sie die View IDEPTREE ab, um die Ergebnisse anzuzeigen.

**Entfernen Sie die Kommentarzeichen, und wählen Sie den Code unter der Aufgabe 1\_c. Um das Skript auszuführen, klicken Sie in der SQL Worksheet-Symbolleiste auf das Symbol "Run Script" (F5). Der Code und die Ergebnisse sind unten abgebildet.**

```
SELECT * FROM IDEPTREE;
```

```
Script Output
Run Edit Save Print Paste | Task completed in 0.201 seconds
DEPENDENCIES
-----
PROCEDURE ORA61.ADD_EMPLOYEE
```

- d. Führen Sie die Prozedur deptree\_fill für die Funktion valid\_deptid aus.

**Entfernen Sie die Kommentarzeichen, und wählen Sie den Code unter der Aufgabe 1\_d. Um das Skript auszuführen, klicken Sie in der SQL Worksheet-Symbolleiste auf das Symbol "Run Script" (F5). Der Code und die Ergebnisse sind unten abgebildet.**

```
EXECUTE deptree_fill('FUNCTION', USER, 'valid_deptid')
```

```
Script Output
Run Edit Save Print Paste | Task completed in 0.945 seconds
anonymous block completed
```

- e. Fragen Sie die View IDEPTREE ab, um die Ergebnisse anzuzeigen.

**Entfernen Sie die Kommentarzeichen, und wählen Sie den Code unter der Aufgabe 1\_e. Um das Skript auszuführen, klicken Sie in der SQL Worksheet-Symbolleiste auf das Symbol "Execute Statement" (F9). Der Code und die Ergebnisse sind unten abgebildet.**

```
SELECT * FROM IDEPTREE;
```

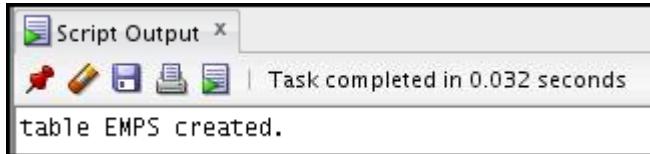
```
Script Output
Run Edit Save Print Paste | Task completed in 0.003 seconds
DEPENDENCIES
-----
PROCEDURE ORA61.ADD_EMPLOYEE
FUNCTION ORA61.VALID_DEPTID
```

**Führen Sie die folgende Übung durch, falls Sie noch Zeit haben:**

2. Ändern Sie ungültige Objekte dynamisch wieder in gültige Objekte.
  - a. Erstellen Sie eine Kopie Ihrer Tabelle EMPLOYEES, und nennen Sie sie EMPS.

**Entfernen Sie die Kommentarzeichen, und wählen Sie den Code unter der Aufgabe 2\_a. Um das Skript auszuführen, klicken Sie in der SQL Worksheet-Symbolleiste auf das Symbol "Run Script" (F5). Der Code und die Ergebnisse sind unten abgebildet.**

```
CREATE TABLE emps AS
SELECT * FROM employees;
```

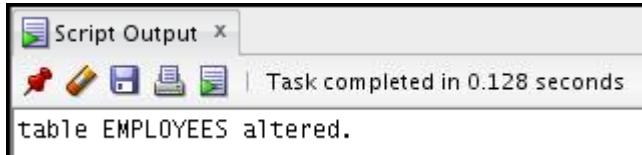


**Hinweis:** Ignorieren Sie bei der Ausführung der Anweisung CREATE eventuell angezeigte Fehlermeldungen.

- b. Ändern Sie die Tabelle EMPLOYEES, indem Sie die Spalte TOTSAL mit dem Datentyp NUMBER (9, 2) hinzufügen.

**Entfernen Sie die Kommentarzeichen, und wählen Sie den Code unter der Aufgabe 2\_b. Um das Skript auszuführen, klicken Sie in der SQL Worksheet-Symbolleiste auf das Symbol "Run Script" (F5). Der Code und die Ergebnisse sind unten abgebildet.**

```
ALTER TABLE employees
ADD (totalsal NUMBER(9,2));
```



- c. Erstellen und speichern Sie eine Abfrage, um Name, Typ und Status aller ungültigen Objekte anzuzeigen.

**Entfernen Sie die Kommentarzeichen, und wählen Sie den Code unter der Aufgabe 2\_c. Um das Skript auszuführen, klicken Sie in der SQL Worksheet-Symbolleiste auf das Symbol "Execute Statement" (F9). Der Code und die Ergebnisse sind unten abgebildet.**

```
SELECT object_name, object_type, status
FROM USER_OBJECTS
WHERE status = 'INVALID';
```

|   | OBJECT_NAME        | OBJECT_TYPE  | STATUS  |
|---|--------------------|--------------|---------|
| 1 | COMPILE_PKG        | PACKAGE BODY | INVALID |
| 2 | DELETE_EMP_TRG     | TRIGGER      | INVALID |
| 3 | CHECK_SALARY_TRG   | TRIGGER      | INVALID |
| 4 | EMP_PKG            | PACKAGE BODY | INVALID |
| 5 | EMP_PKG            | PACKAGE      | INVALID |
| 6 | UPDATE_JOB_HISTORY | TRIGGER      | INVALID |

**Hinweis:** Ignorieren Sie die Abweichungen im Screenshot.

- d. Fügen Sie in das Package `compile_pkg` (erstellt in den Übungen zur 7. Lektion, "Dynamisches SQL") eine Prozedur `recompile` ein, die alle ungültigen Prozeduren, Funktionen und Packages in Ihrem Schema rekompiliert. Verwenden Sie natives dynamisches SQL, um den ungültigen Objekttyp zu ändern und zu kompilieren.

**Entfernen Sie die Kommentarzeichen, und wählen Sie den Code unter der Aufgabe 2\_d. Um das Skript auszuführen, klicken Sie in der SQL Worksheet-Symbolleiste auf das Symbol "Run Script" (F5). Der Code und die Ergebnisse sind unten abgebildet. Der neu hinzugefügte Code ist im folgenden Codefeld durch Fettschrift hervorgehoben.**

```

CREATE OR REPLACE PACKAGE compile_pkg IS
    PROCEDURE make(p_name VARCHAR2);
    PROCEDURE recompile;
END compile_pkg;
/


SHOW ERRORS


CREATE OR REPLACE PACKAGE BODY compile_pkg_body IS

    PROCEDURE p_execute(stmt VARCHAR2) IS
    BEGIN
        DBMS_OUTPUT.PUT_LINE(stmt);
        EXECUTE IMMEDIATE stmt;
    END;

    FUNCTION get_type(p_name VARCHAR2) RETURN VARCHAR2 IS
        proc_type VARCHAR2(30) := NULL;
    BEGIN
        -- The ROWNUM = 1 is added to the condition
        -- to ensure only one row is returned if the
        -- name represents a PACKAGE, which may also
        -- have a PACKAGE BODY. In this case, we can
        -- only compile the complete package, but not
        -- the specification or body as separate

```

```

-- components.
SELECT object_type INTO proc_type
FROM user_objects
WHERE object_name = UPPER(p_name)
AND ROWNUM = 1;
RETURN proc_type;
EXCEPTION
WHEN NO_DATA_FOUND THEN
RETURN NULL;
END;

PROCEDURE make(p_name VARCHAR2) IS
stmt      VARCHAR2(100);
proc_type  VARCHAR2(30) := get_type(p_name);
BEGIN
IF proc_type IS NOT NULL THEN
stmt := 'ALTER || proc_type || ''|| p_name ||' COMPILE';

p_execute(stmt);
ELSE
RAISE_APPLICATION_ERROR(-20001,
'Subprogram '''|| p_name ||''' does not exist');
END IF;
END make;

PROCEDURE recompile IS
stmt VARCHAR2(200);
obj_name user_objects.object_name%type;
obj_type user_objects.object_type%type;
BEGIN
FOR objrec IN (SELECT object_name, object_type
                FROM user_objects
               WHERE status = 'INVALID'
                 AND object_type <> 'PACKAGE BODY')
LOOP
stmt := 'ALTER || objrec.object_type || ''||'
       objrec.object_name ||' COMPILE';
p_execute(stmt);
END LOOP;
END recompile;

END compile_pkg;

```

/

SHOW ERRORS

```

Script Output x
Task completed in 0.13 seconds
PACKAGE_COMPILE_PKG compiled
No Errors.
PACKAGE_BODY_COMPILE_PKG compiled
No Errors.

```

- e. Führen Sie die Prozedur `compile_pkg.recompile` aus.

**Entfernen Sie die Kommentarzeichen, und wählen Sie den Code unter der Aufgabe 2\_d. Um das Skript auszuführen, klicken Sie in der SQL Worksheet-Symbolleiste auf das Symbol "Run Script" (F5). Der Code und die Ergebnisse sind unten abgebildet.**

```
EXECUTE compile_pkg.recompile
```

```

Script Output x
Task completed in 0.372 seconds
anonymous block completed

```

**Hinweis:** Ignorieren Sie eventuell angezeigte Fehlermeldungen. Die Prozedur wäre kompiliert worden.

- f. Um den Wert der Spalte STATUS zu prüfen, führen Sie die Skriptdatei aus, die Sie in Schritt 2\_c erstellt haben. Sind noch immer Objekte mit dem Status INVALID vorhanden?

**Entfernen Sie die Kommentarzeichen, und wählen Sie den Code unter der Aufgabe 2\_c. Um das Skript auszuführen, klicken Sie in der SQL Worksheet-Symbolleiste auf das Symbol "Execute Statement" (F9). Der Code und die Ergebnisse sind unten abgebildet.**

```
SELECT object_name, object_type, status
FROM USER_OBJECTS
WHERE status = 'INVALID';
```

| OBJECT_NAME | OBJECT_TYPE | STATUS |
|-------------|-------------|--------|
|             |             |        |

**Hinweis:** Vergleichen Sie diese Ausgabe mit der Ausgabe aus Schritt 2(c). Wie Sie sehen, sind alle Objekte aus dem vorherigen Screenshot jetzt gültig.

# **Zusätzliche Übungen 1**

## **Kapitel 13**

# Zusätzliche Übungen 1 – Überblick

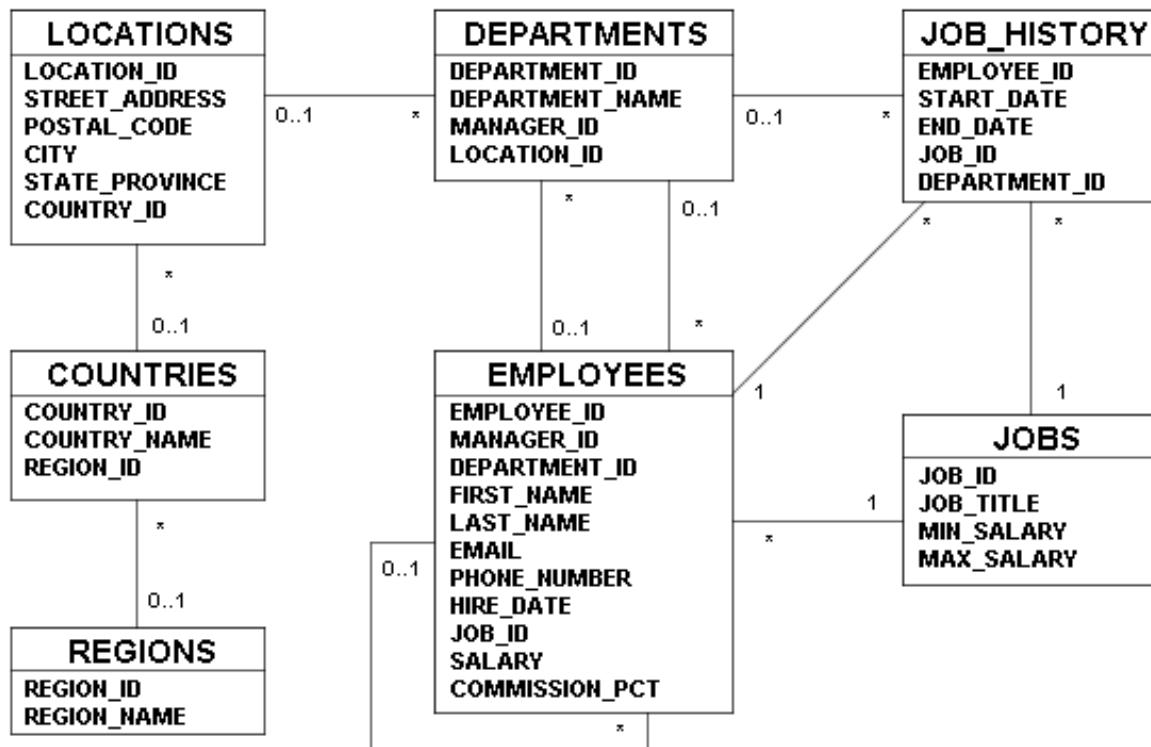
## Lektionsüberblick

Die zusätzlichen Übungen sind eine Ergänzung zum Kurs *Oracle Database: Develop PL/SQL Program Units*. Sie wenden in den Übungen die im Kurs beschriebenen Konzepte an. Die zusätzlichen Übungen sind in zwei Lektionen gegliedert.

Lektion 1 enthält ergänzende Übungen zum Erstellen von Stored Procedures, Funktionen, Packages und Triggern und zur Verwendung der von Oracle bereitgestellten Packages mit SQL Developer oder SQL\*Plus als Entwicklungsumgebung. Zu den in diesem Übungsteil verwendeten Tabellen gehören EMPLOYEES, JOBS, JOB\_HISTORY und DEPARTMENTS.

Zu Beginn jeder Übung finden Sie ein Entity Relationship-Diagramm. Jedes Entity Relationship-Diagramm zeigt die Tabellen-Entitys und ihre Beziehungen. Detailliertere Definitionen der Tabellen und der darin enthaltenen Daten enthält der Anhang "Zusätzliche Übungen: Tabellenbeschreibungen und -daten".

## Schema Human Resources (HR) – Entity Relationship-Diagramm



# Übung 1 zu Lektion 1 – Neue Datenbankverbindung für SQL Developer erstellen

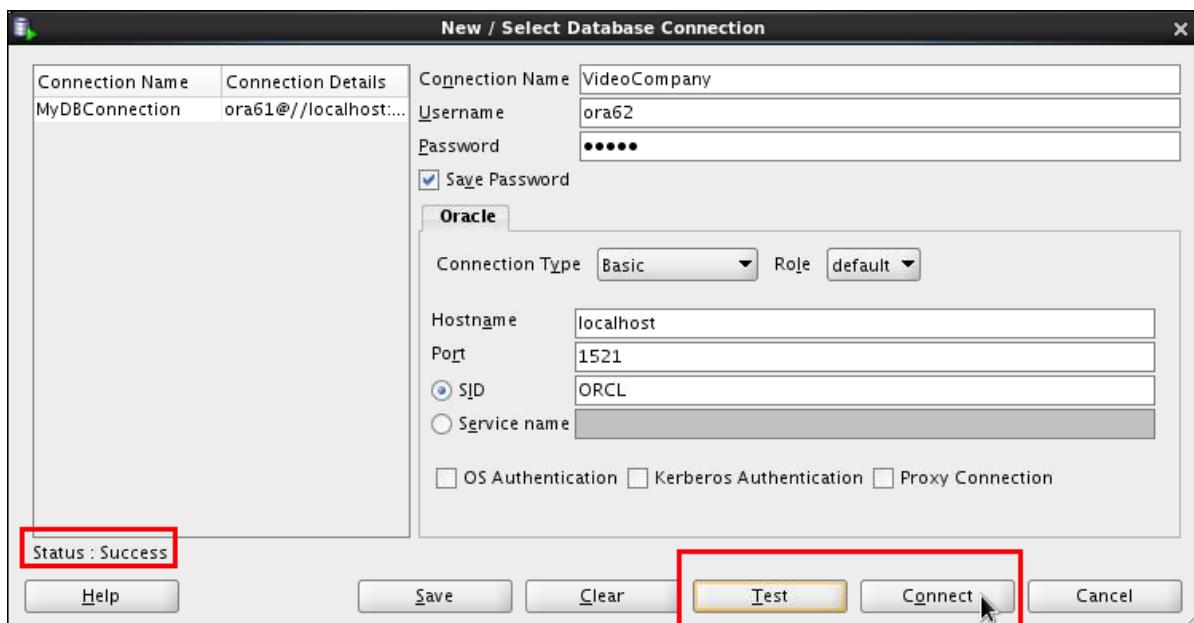
## Überblick

In dieser Übung starten Sie SQL Developer mit Ihren Verbindungsinformationen und erstellen eine neue Datenbankverbindung.

Starten Sie SQL Developer mit dem Benutzernamen und dem Kennwort, die Ihnen vom Dozenten vorgegeben wurden, beispielsweise ora62.

## Aufgabe

1. Starten Sie SQL Developer mit dem Benutzernamen und dem Kennwort, die Ihnen vom Dozenten vorgegeben wurden, beispielsweise ora62.
2. Melden Sie sich mit folgenden Daten bei der Datenbank an:
  - a. **Connection Name:** VideoCompany
  - b. **Username:** ora62
  - c. **Password:** ora62
  - d. Aktivieren Sie das Kontrollkästchen **Save Password**.
  - e. **Hostname:** Geben Sie den Hostnamen für Ihren PC oder alternativ localhost ein.
  - f. **Port:** 1521
  - g. **SID:** ORCL
3. Testen Sie die neue Verbindung. Wenn als Status **Success** angezeigt wird, melden Sie sich über diese neue Verbindung bei der Datenbank an:
  - a. Klicken Sie im Fenster **New/Select Database Connection** auf die Schaltfläche **Test**. Wenn als Status **Success** angezeigt wird, klicken Sie auf die Schaltfläche **Connect**.



## Übung 1 zu Lektion 1 – Lösung: Neue Datenbankverbindung für SQL Developer erstellen

In dieser Übung starten Sie SQL Developer mit Ihren Verbindungsinformationen und erstellen eine neue Datenbankverbindung.

1. Starten Sie SQL Developer mit dem Benutzernamen und dem Kennwort, die Ihnen vom Dozenten vorgegeben wurden, beispielsweise ora62.

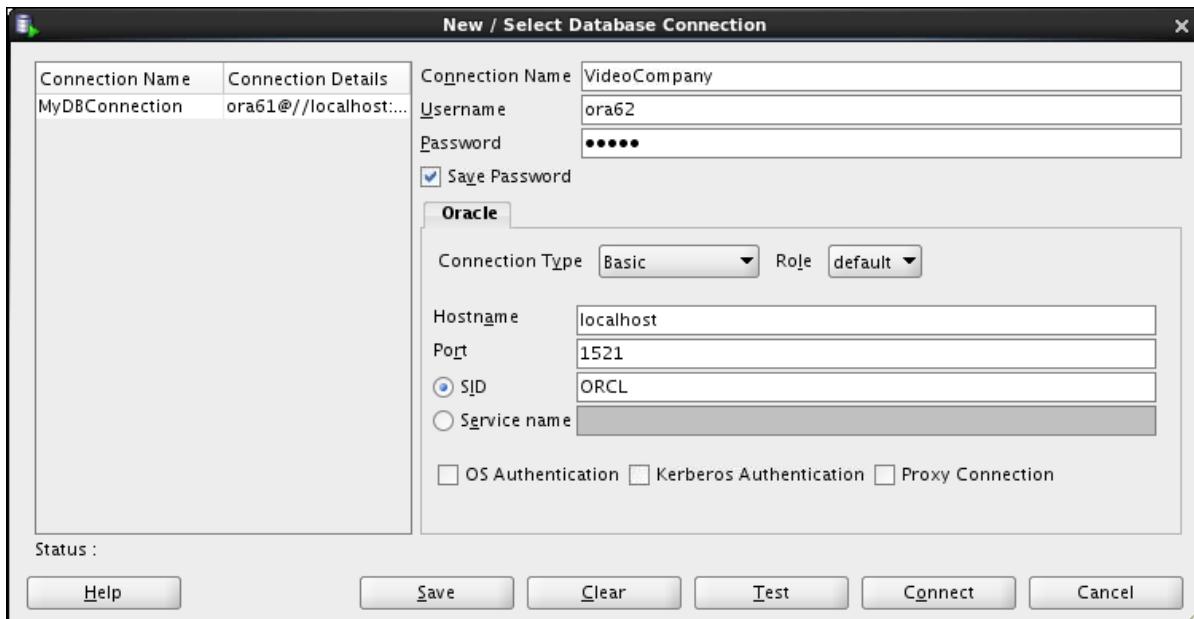
**Klicken Sie auf Ihrem Desktop auf das Symbol "SQL Developer".**



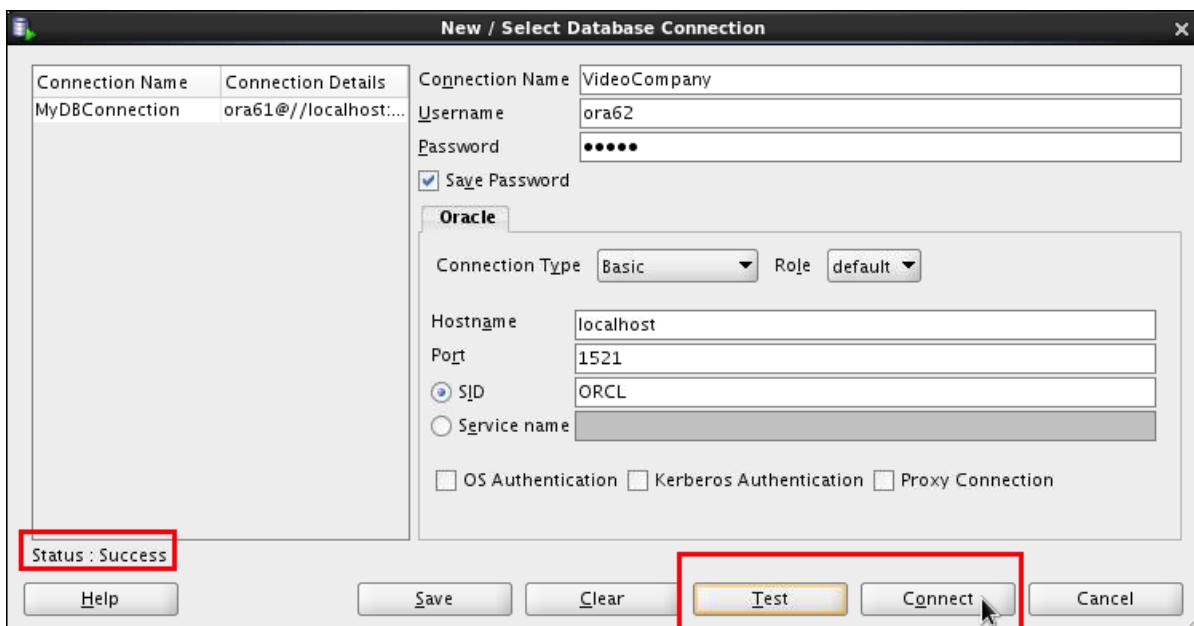
2. Melden Sie sich mit folgenden Daten bei der Datenbank an:
  - a. **Connection Name:** VideoCompany
  - b. **Username:** ora62
  - c. **Password:** ora62
  - d. **Hostname:** Geben Sie den Hostnamen für Ihren PC oder alternativ localhost ein.
  - e. **Port:** 1521
  - f. **SID:** ORCL

**Klicken Sie in der Registerkarte "Connections" mit der rechten Maustaste auf das Symbol "Connections", und wählen Sie im Kontextmenü die Option "New Connection". Das Fenster "New/Select Database Connection" wird angezeigt. Erstellen Sie mithilfe der oben aufgeführten Informationen die neue Datenbankverbindung.**

**Hinweis:** Um die Eigenschaften der neu erstellten Verbindung anzuzeigen, klicken Sie mit der rechten Maustaste auf den Namen der Verbindung und wählen im Kontextmenü die Option **Properties**. Ersetzen Sie die Werte bei **Username**, **Password**, **Hostname** und **Service name** durch die Werte, die Sie von Ihrem Dozenten erhalten haben. Nachstehend finden Sie ein Beispiel für eine neu erstellte Datenbankverbindung für den Teilnehmer ora62:



3. Testen Sie die neue Verbindung. Wenn als Status **Success** angezeigt wird, melden Sie sich über diese neue Verbindung bei der Datenbank an:
  - a. Klicken Sie im Fenster **New>Select Database Connection** auf die Schaltfläche **Test**. Wenn als Status **Success** angezeigt wird, klicken Sie auf die Schaltfläche **Connect**.



## Übung 2 zu Lektion 1 – Neue Tätigkeit zur Tabelle JOBS hinzufügen

### Überblick

In dieser Übung erstellen Sie ein Unterprogramm, um eine neue Tätigkeit zur Tabelle JOBS hinzuzufügen.

### Aufgaben

1. Erstellen Sie eine Stored Procedure namens NEW\_JOB, um einen neuen Auftrag in die Tabelle JOBS einzufügen. Die Prozedur soll drei Parameter annehmen: Die ersten beiden Parameter geben eine Tätigkeits-ID und eine Tätigkeitskennung an. Der dritte Parameter legt das Mindestgehalt fest. Legen Sie das Doppelte des für die Tätigkeits-ID angegebenen Mindestgehalts als Höchstgehalt für die neue Tätigkeit fest.
2. Aktivieren Sie SERVEROUTPUT, und rufen Sie dann die Prozedur auf, um einen neuen Job mit der Tätigkeits-ID SY\_ANAL, der Tätigkeitskennung System Analyst und einem Mindestgehalt von 6000 hinzuzufügen.
3. Prüfen Sie, ob eine Zeile hinzugefügt wurde, und notieren Sie sich die neue Tätigkeits-ID für die nächste Übung. Schreiben Sie die Änderungen fest.

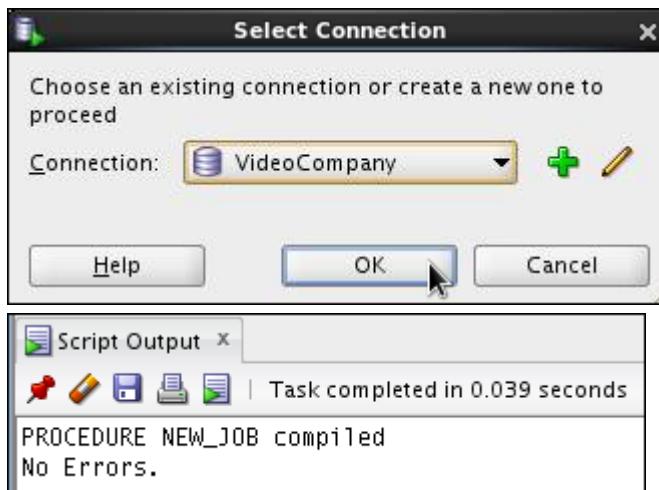
## Übung 2 zu Lektion 1 – Lösung: Neue Tätigkeit zur Tabelle JOBS hinzufügen

In dieser Übung erstellen Sie ein Unterprogramm, um eine neue Tätigkeit zur Tabelle JOBS hinzuzufügen.

1. Erstellen Sie eine Stored Procedure namens NEW\_JOB, um einen neuen Auftrag in die Tabelle JOBS einzufügen. Die Prozedur soll drei Parameter annehmen: Die ersten beiden Parameter geben eine Tätigkeits-ID und eine Tätigkeitskennung an. Der dritte Parameter legt das Mindestgehalt fest. Legen Sie das Doppelte des für die Tätigkeits-ID angegebenen Mindestgehalts als Höchstgehalt für die neue Tätigkeit fest.

**Öffnen Sie das Skript /home/oracle/labs/plpu/solns/sol\_ap1.sql Entfernen Sie die Kommentarzeichen, und wählen Sie den Code in Aufgabe 1 der zusätzlichen Übung 2 zu Lektion 1. Klicken Sie in der SQL Worksheet-Symbolleiste auf das Symbol "Run Script" (F5), um die Prozedur zu erstellen und zu kompilieren. Stellen Sie sicher, dass Sie die neue videoCompany-Verbindung gewählt haben. Code, Aufforderung zur Wahl einer Verbindung und Ergebnisse werden wie folgt angezeigt:**

```
CREATE OR REPLACE PROCEDURE new_job(
    p_jobid    IN jobs.job_id%TYPE,
    p_title    IN jobs.job_title%TYPE,
    v_minsal   IN jobs.min_salary%TYPE) IS
    v_maxsal   jobs.max_salary%TYPE := 2 * v_minsal;
BEGIN
    INSERT INTO jobs(job_id, job_title, min_salary, max_salary)
    VALUES (p_jobid, p_title, v_minsal, v_maxsal);
    DBMS_OUTPUT.PUT_LINE ('New row added to JOBS table:');
    DBMS_OUTPUT.PUT_LINE (p_jobid || ' ' || p_title || ' ' ||
                          v_minsal || ' ' || v_maxsal);
END new_job;
/
SHOW ERRORS
```



2. Aktivieren Sie SERVEROUTPUT, und rufen Sie dann die Prozedur auf, um einen neuen Job mit der Tätigkeits-ID SY\_ANAL, der Tätigkeitskennung System Analyst und einem Mindestgehalt von 6000 hinzuzufügen.

**Entfernen Sie die Kommentarzeichen, und wählen Sie den Code in der Aufgabe 2 der zusätzlichen Übung 2 zu Lektion 1. Wählen Sie bei entsprechender Aufforderung die neue Verbindung VideoCompany. Der Code und die Ergebnisse werden wie folgt angezeigt:**

```
SET SERVEROUTPUT ON
```

```
EXECUTE new_job ('SY_ANAL', 'System Analyst', 6000)
```

```
Script Output X
Task completed in 1.065 seconds
anonymous block completed
New row added to JOBS table:
SY_ANAL System Analyst 6000 12000
```

3. Prüfen Sie, ob eine Zeile hinzugefügt wurde, und notieren Sie sich die neue Tätigkeits-ID für die nächste Übung. Schreiben Sie die Änderungen fest.

**Entfernen Sie die Kommentarzeichen, und wählen Sie den Code in der Aufgabe 3 der zusätzlichen Übung 2 zu Lektion 1. Der Code und die Ergebnisse werden wie folgt angezeigt:**

```
SELECT *
FROM   jobs
WHERE  job_id = 'SY_ANAL';
COMMIT;
```

| JOB_ID  | JOB_TITLE      | MIN_SALARY | MAX_SALARY |
|---------|----------------|------------|------------|
| SY_ANAL | System Analyst | 6000       | 12000      |

```
Script Output X
Task completed in 0.047 seconds
JOB_ID      JOB_TITLE          MIN_SALARY  MAX_SALARY
-----      -----          -----
SY_ANAL      System Analyst      6000       12000
committed.
```

## Übung 3 zu Lektion 1 – Neue Rolle zur Tabelle JOB\_HISTORY hinzufügen

---

### Überblick

In dieser zusätzlichen Übung fügen Sie für einen vorhandenen Mitarbeiter eine neue Rolle in die Tabelle JOB\_HISTORY ein.

### Aufgaben

1. Erstellen Sie die Stored Procedure ADD\_JOB\_HIST, um eine neue Zeile in die Tabelle JOB\_HISTORY einzufügen. Diese Zeile soll einen Mitarbeiter enthalten, der die Tätigkeit wechselt und für den jetzt die in Übung 2 erstellte neue Tätigkeits-ID (SY\_ANAL) gilt.
  - a. Die Prozedur sollte zwei Parameter liefern, einen für die ID des Mitarbeiters, der die Tätigkeit wechselt, und die zweite für die neue Tätigkeits-ID.
  - b. Lesen Sie die Personalnummer aus der Tabelle EMPLOYEES, und fügen Sie sie in die Tabelle JOB\_HISTORY ein.
  - c. Legen Sie das Einstellungsdatum dieses Mitarbeiters als Anfangsdatum und das aktuelle Datum als Enddatum für diese Zeile in der Tabelle JOB\_HISTORY fest.
  - d. Ändern Sie das Einstellungsdatum des Mitarbeiters in der Tabelle EMPLOYEES in das Datum des aktuellen Tages.
  - e. Aktualisieren Sie die Tätigkeits-ID für diesen Mitarbeiter mit der übergebenen Tätigkeits-ID (SY\_ANAL). Legen Sie als Gehalt das Mindestgehalt für diese Tätigkeits-ID + 500 fest.  
**Hinweis:** Sehen Sie eine Exception-Behandlung für den Fall vor, dass versucht wird, einen nicht vorhandenen Mitarbeiter einzufügen.
2. Deaktivieren Sie alle Trigger für die Tabellen EMPLOYEES, JOBS und JOB\_HISTORY, bevor Sie die Prozedur ADD\_JOB\_HIST aufrufen.
3. Aktivieren Sie SERVEROUTPUT, und führen Sie dann die Prozedur mit der Personalnummer 106 und der Tätigkeits-ID SY\_ANAL als Parameter aus.
4. Führen Sie eine Abfrage in den Tabellen JOB\_HISTORY und EMPLOYEES durch, um Ihre Änderungen für den Mitarbeiter 106 anzuzeigen, und schreiben Sie die Änderungen anschließend fest.
5. Aktivieren Sie die Trigger für die Tabellen EMPLOYEES, JOBS und JOB\_HISTORY erneut.

## Übung 3 zu Lektion 1 – Lösung: Neue Rolle zur Tabelle JOB\_HISTORY hinzufügen

---

In dieser zusätzlichen Übung fügen Sie für einen vorhandenen Mitarbeiter eine neue Rolle in die Tabelle JOB\_HISTORY ein.

1. Erstellen Sie die Stored Procedure ADD\_JOB\_HIST, um eine neue Zeile in die Tabelle JOB\_HISTORY einzufügen. Diese Zeile soll einen Mitarbeiter enthalten, der die Tätigkeit wechselt und für den jetzt die in Übung 2 erstellte neue Tätigkeits-ID (SY\_ANAL) gilt.
  - a. Die Prozedur sollte zwei Parameter liefern, einen für die ID des Mitarbeiters, der die Tätigkeit wechselt, und die zweite für die neue Tätigkeits-ID.
  - b. Lesen Sie die Personalnummer aus der Tabelle EMPLOYEES, und fügen Sie sie in die Tabelle JOB\_HISTORY ein.
  - c. Legen Sie das Einstellungsdatum dieses Mitarbeiters als Anfangsdatum und das aktuelle Datum als Enddatum für diese Zeile in der Tabelle JOB\_HISTORY fest.
  - d. Ändern Sie das Einstellungsdatum des Mitarbeiters in der Tabelle EMPLOYEES in das Datum des aktuellen Tages.
  - e. Aktualisieren Sie die Tätigkeits-ID für diesen Mitarbeiter mit der übergebenen Tätigkeits-ID (SY\_ANAL). Legen Sie als Gehalt das Mindestgehalt für diese Tätigkeits-ID + 500 fest.

**Hinweis:** Sehen Sie eine Exception-Behandlung für den Fall vor, dass versucht wird, einen nicht vorhandenen Mitarbeiter einzufügen.

**Entfernen Sie die Kommentarzeichen, und wählen Sie den Code in Aufgabe 1 der zusätzlichen Übung 3 von Lektion 1. Der Code und die Ergebnisse werden wie folgt angezeigt:**

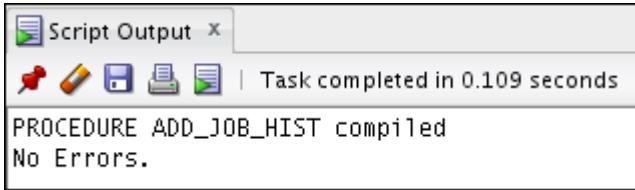
```
CREATE OR REPLACE PROCEDURE add_job_hist(
  p_emp_id      IN employees.employee_id%TYPE,
  p_new_jobid   IN jobs.job_id%TYPE) IS
BEGIN
  INSERT INTO job_history
    SELECT employee_id, hire_date, SYSDATE, job_id,
           department_id
      FROM   employees
     WHERE  employee_id = p_emp_id;
  UPDATE employees
    SET   hire_date = SYSDATE,
          job_id = p_new_jobid,
          salary = (SELECT min_salary + 500
                      FROM   jobs
                     WHERE  job_id = p_new_jobid)
   WHERE employee_id = p_emp_id;
  DBMS_OUTPUT.PUT_LINE ('Added employee ' || p_emp_id ||
```

```

        ' details to the JOB_HISTORY table');
DBMS_OUTPUT.PUT_LINE ('Updated current job of employee ' ||
                      p_emp_id|| ' to '|| p_new_jobid);

EXCEPTION
    WHEN NO_DATA_FOUND THEN
        RAISE_APPLICATION_ERROR (-20001, 'Employee does not
exist!');
END add_job_hist;
/
SHOW ERRORS

```



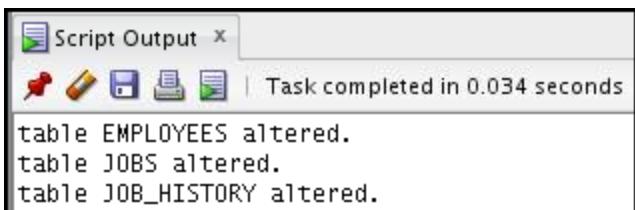
- Deaktivieren Sie alle Trigger für die Tabellen EMPLOYEES, JOBS und JOB\_HISTORY, bevor Sie die Prozedur ADD\_JOB\_HIST aufrufen.

**Entfernen Sie die Kommentarzeichen, und wählen Sie den Code in Aufgabe 2 der zusätzlichen Übung 3 von Lektion 1. Der Code und die Ergebnisse werden wie folgt angezeigt:**

```

ALTER TABLE employees DISABLE ALL TRIGGERS;
ALTER TABLE jobs DISABLE ALL TRIGGERS;
ALTER TABLE job_history DISABLE ALL TRIGGERS;

```



- Aktivieren Sie SERVEROUTPUT, und führen Sie dann die Prozedur mit der Personalnummer 106 und der Tätigkeits-ID SY\_ANAL als Parameter aus.

**Entfernen Sie die Kommentarzeichen, und wählen Sie den Code in Aufgabe 3 der zusätzlichen Übung 3 von Lektion 1. Der Code und die Ergebnisse werden wie folgt angezeigt:**

```

SET SERVEROUTPUT ON

EXECUTE add_job_hist(106, 'SY_ANAL')

```

Script Output | Task completed in 0.101 seconds

```
anonymous block completed
Added employee 106 details to the JOB_HISTORY table
Updated current job of employee 106 to SY_ANAL
```

4. Führen Sie eine Abfrage in den Tabellen JOB\_HISTORY und EMPLOYEES durch, um Ihre Änderungen für den Mitarbeiter 106 anzuzeigen, und schreiben Sie die Änderungen anschließend fest.

**Entfernen Sie die Kommentarzeichen, und wählen Sie den Code in Aufgabe 4 der zusätzlichen Übung 3 von Lektion 1. Der Code und die Ergebnisse werden wie folgt angezeigt:**

```
SELECT * FROM job_history
WHERE employee_id = 106;

SELECT job_id, salary FROM employees
WHERE employee_id = 106;

COMMIT;
```

Script Output | Task completed in 0.038 seconds

| EMPLOYEE_ID | START_DATE | END_DATE  | JOB_ID  | DEPARTMENT_ID |
|-------------|------------|-----------|---------|---------------|
| 106         | 05-FEB-06  | 21-NOV-12 | IT_PROG | 60            |

| JOB_ID  | SALARY |
|---------|--------|
| SY_ANAL | 6500   |

committed.

5. Aktivieren Sie die Trigger für die Tabellen EMPLOYEES, JOBS und JOB\_HISTORY erneut.

**Entfernen Sie die Kommentarzeichen, und wählen Sie den Code in Aufgabe 5 der zusätzlichen Übung 3 von Lektion 1. Der Code und die Ergebnisse werden wie folgt angezeigt:**

```
ALTER TABLE employees ENABLE ALL TRIGGERS;
ALTER TABLE jobs ENABLE ALL TRIGGERS;
ALTER TABLE job_history ENABLE ALL TRIGGERS;
```

Script Output | Task completed in 0.082 seconds

```
table EMPLOYEES altered.
table JOBS altered.
table JOB_HISTORY altered.
```

## Übung 4 zu Lektion 1 – Mindest- und Höchstgehalt für eine Tätigkeit aktualisieren

---

### Überblick

In dieser zusätzlichen Übung erstellen Sie ein Programm, um das Mindest- und das Höchstgehalt für eine Tätigkeit in der Tabelle JOBS zu aktualisieren.

### Aufgaben

1. Erstellen Sie die Stored Procedure UPD\_JOBSAL, um das Mindest- und das Höchstgehalt für eine bestimmte Tätigkeits-ID in der Tabelle JOBS zu aktualisieren. Die Prozedur soll drei Parameter zur Verfügung stellen: die Tätigkeits-ID, ein neues Mindestgehalt und ein neues Höchstgehalt. Programmieren Sie die notwendige Exception-Behandlung für den Fall, dass eine für die Tabelle JOBS ungültige Tätigkeits-ID angegeben wird. Lösen Sie eine Exception aus, wenn das angegebene Höchstgehalt unter dem Mindestgehalt liegt. Stellen Sie außerdem eine Meldung bereit, die angezeigt wird, wenn die Zeile in der Tabelle JOBS gesperrt ist.

**Hinweis:** Die Fehlernummer für gesperrte/belegte Ressourcen ist -54.

2. Aktivieren Sie SERVEROUTPUT, und führen Sie die Prozedur UPD\_JOBSAL mithilfe der Tätigkeits-ID SY\_ANAL, einem Mindestgehalt von 7000 und einem Höchstgehalt von 140 aus.

**Hinweis:** Hierbei sollte eine Exception-Meldung generiert werden.

3. Deaktivieren Sie Trigger für die Tabellen EMPLOYEES und JOBS.
4. Führen Sie die Prozedur UPD\_JOBSAL mit der Tätigkeits-ID SY\_ANAL, einem Mindestgehalt von 7000 und einem Höchstgehalt von 14000 aus.
5. Fragen Sie die Tabelle JOBS ab, um die Änderungen anzuzeigen. Schreiben Sie die Änderungen anschließend fest.
6. Aktivieren Sie die Trigger für die Tabellen EMPLOYEES und JOBS.

## Übung 4 zu Lektion 1 – Lösung: Mindest- und Höchstgehalt für eine Tätigkeit aktualisieren

---

In dieser zusätzlichen Übung erstellen Sie ein Programm, um das Mindest- und das Höchstgehalt für eine Tätigkeit in der Tabelle JOBS zu aktualisieren.

1. Erstellen Sie die Stored Procedure UPD\_JOBSAL, um das Mindest- und das Höchstgehalt für eine bestimmte Tätigkeits-ID in der Tabelle JOBS zu aktualisieren. Die Prozedur soll drei Parameter zur Verfügung stellen: die Tätigkeits-ID, ein neues Mindestgehalt und ein neues Höchstgehalt. Programmieren Sie die notwendige Exception-Behandlung für den Fall, dass eine für die Tabelle JOBS ungültige Tätigkeits-ID angegeben wird. Lösen Sie eine Exception aus, wenn das angegebene Höchstgehalt unter dem Mindestgehalt liegt. Stellen Sie außerdem eine Meldung bereit, die angezeigt wird, wenn die Zeile in der Tabelle JOBS gesperrt ist.

**Hinweis:** Die Fehlernummer für gesperrte/belegte Ressourcen ist -54.

**Entfernen Sie die Kommentarzeichen, und wählen Sie den Code in Aufgabe 1 der zusätzlichen Übung 4 zu Lektion 1. Der Code und die Ergebnisse werden wie folgt angezeigt:**

```

CREATE OR REPLACE PROCEDURE upd_jobsal (
    p_jobid      IN jobs.job_id%type,
    p_new_minsal  IN jobs.min_salary%type,
    p_new_maxsal  IN jobs.max_salary%type) IS
    v_dummy          PLS_INTEGER;
    e_resource_busy  EXCEPTION;
    e_sal_error      EXCEPTION;
    PRAGMA           EXCEPTION_INIT (e_resource_busy , -54);
BEGIN
    IF (p_new_maxsal < p_new_minsal) THEN
        RAISE e_sal_error;
    END IF;
    SELECT 1 INTO v_dummy
        FROM jobs
        WHERE job_id = p_jobid
        FOR UPDATE OF min_salary NOWAIT;
    UPDATE jobs
        SET min_salary =  p_new_minsal,
            max_salary =  p_new_maxsal
        WHERE job_id  = p_jobid;
EXCEPTION
    WHEN e_resource_busy THEN
        RAISE_APPLICATION_ERROR (-20001,
            'Job information is currently locked, try later.');
    WHEN NO_DATA_FOUND THEN

```

```

        RAISE_APPLICATION_ERROR(-20001, 'This job ID does not
exist');

WHEN e_sal_error THEN
    RAISE_APPLICATION_ERROR(-20001,
        'Data error: Max salary should be more than min salary');
END upd_jobsal;
/
SHOW ERRORS

```

The screenshot shows the 'Script Output' window from Oracle SQL Developer. It displays the message 'PROCEDURE UPD\_JOBSAL compiled' and 'No Errors.' at the bottom. Above this, it says 'Task completed in 0.052 seconds'. The window has standard OS X-style icons for file operations.

- Aktivieren Sie SERVEROUTPUT, und führen Sie die Prozedur UPD\_JOBSAL mithilfe der Tätigkeits-ID SY\_ANAL, einem Mindestgehalt von 7000 und einem Höchstgehalt von 140 aus.

**Hinweis:** Hierbei sollte eine Exception-Meldung generiert werden.

**Entfernen Sie die Kommentarzeichen, und wählen Sie den Code in Aufgabe 2 der zusätzlichen Übung 4 zu Lektion 1. Der Code und die Ergebnisse werden wie folgt angezeigt:**

```

SET SERVEROUTPUT ON

EXECUTE upd_jobsal('SY_ANAL', 7000, 140)

```

The screenshot shows the 'Script Output' window from Oracle SQL Developer. It displays an error report starting at line 171. The error message is 'ORA-20001: Data error: Max salary should be more than min salary'. It also includes the error code 'ORA-06512' and its location 'at "ORA62.UPD\_JOBSAL", line 28'. The window also shows 'Error report:' and 'Error starting at line 171 in command: EXECUTE upd\_jobsal('SY\_ANAL', 7000, 140)'. Above this, it says 'Task completed in 1.517 seconds'. The window has standard OS X-style icons for file operations.

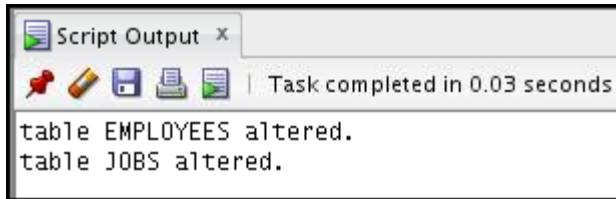
- Deaktivieren Sie Trigger für die Tabellen EMPLOYEES und JOBS.

**Entfernen Sie die Kommentarzeichen, und wählen Sie den Code in Aufgabe 3 der zusätzlichen Übung 4 zu Lektion 1. Der Code und die Ergebnisse werden wie folgt angezeigt:**

```

ALTER TABLE employees DISABLE ALL TRIGGERS;
ALTER TABLE jobs DISABLE ALL TRIGGERS;

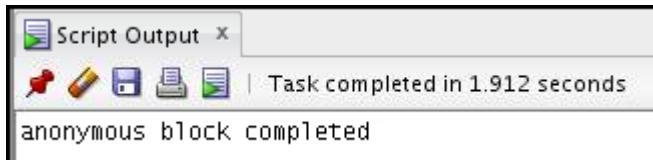
```



- Führen Sie die Prozedur UPD\_JOBSAL mit der Tätigkeits-ID SY\_ANAL, einem Mindestgehalt von 7000 und einem Höchstgehalt von 14000 aus.

**Entfernen Sie die Kommentarzeichen, und wählen Sie den Code in Aufgabe 4 der zusätzlichen Übung 4 zu Lektion 1. Der Code und die Ergebnisse werden wie folgt angezeigt:**

```
EXECUTE upd_jobsal('SY_ANAL', 7000, 14000)
```



- Fragen Sie die Tabelle JOBS ab, um die Änderungen anzuzeigen. Schreiben Sie die Änderungen anschließend fest.

**Entfernen Sie die Kommentarzeichen, und wählen Sie den Code in Aufgabe 5 der zusätzlichen Übung 4 zu Lektion 1. Der Code und die Ergebnisse werden wie folgt angezeigt:**

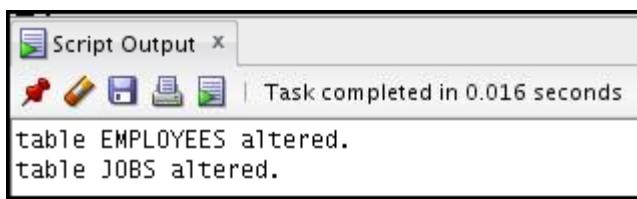
```
SELECT *
FROM   jobs
WHERE  job_id = 'SY_ANAL';
```

| JOB_ID  | JOB_TITLE      | MIN_SALARY | MAX_SALARY |
|---------|----------------|------------|------------|
| SY_ANAL | System Analyst | 7000       | 14000      |

- Aktivieren Sie die Trigger für die Tabellen EMPLOYEES und JOBS.

**Entfernen Sie die Kommentarzeichen, und wählen Sie den Code in Aufgabe 6 der zusätzlichen Übung 4 zu Lektion 1. Der Code und die Ergebnisse werden wie folgt angezeigt:**

```
ALTER TABLE employees ENABLE ALL TRIGGERS;
ALTER TABLE jobs ENABLE ALL TRIGGERS;
```



# Übung 5 zu Lektion 1 – Mitarbeitergehälter überwachen

---

## Überblick

In dieser zusätzlichen Übung erstellen Sie eine Prozedur, mit der Sie überwachen, ob Mitarbeiter das für ihre Tätigkeit gültige Durchschnittsgehalt überschritten haben.

## Aufgaben

1. Deaktivieren Sie den Trigger SECURE\_EMPLOYEES.
  2. Fügen Sie in der Tabelle EMPLOYEES eine Spalte EXCEED\_AVGSAL hinzu, um bis zu drei Zeichen und den Standardwert NO zu speichern. Verwenden Sie ein CHECK-Constraint, um die Werte YES oder NO zuzulassen.
  3. Erstellen Sie eine Stored Procedure namens CHECK\_AVGSAL, die prüft, ob alle Mitarbeitergehälter das für die JOB\_ID gültige Durchschnittsgehalt überschreiten.
    - a. Das Durchschnittsgehalt für eine Tätigkeit wird anhand der Informationen in der Tabelle JOBS berechnet.
    - b. Wenn das Gehalt des Mitarbeiters das für die Tätigkeit errechnete Durchschnittsgehalt übersteigt, aktualisieren Sie die Spalte EXCEED\_AVGSAL in der Tabelle EMPLOYEES auf den Wert YES. Stellen Sie den Wert andernfalls auf NO ein.
    - c. Verwenden Sie einen Cursor, um die Zeilen des Mitarbeiters mit der Option FOR UPDATE in der Abfrage zu wählen.
    - d. Programmieren Sie die notwendige Exception-Behandlung für den Fall, dass ein Record gesperrt ist.

**Tipp:** Die Fehlernummer für gesperrte/belegte Ressourcen ist -54.
  - e. Erstellen und verwenden Sie eine lokale Funktion namens GET\_JOB\_AVGSAL, um das Durchschnittsgehalt für eine als Parameter angegebene Tätigkeits-ID zu bestimmen.
4. Führen Sie die Prozedur CHECK\_AVGSAL aus. Um das Ergebnis Ihrer Änderungen anzuzeigen, erstellen Sie eine Abfrage zum Anzeigen der folgenden Informationen: Personalnummer, Tätigkeitskennung, Durchschnittsgehalt der Tätigkeit, Gehalt des Mitarbeiters und die Indikatorpalte exceed\_avgsal für Mitarbeiter, deren Gehalt das Durchschnittsgehalt für ihre Tätigkeit überschreitet. Schreiben Sie die Änderungen zuletzt fest.
- Hinweis:** Bei diesen Übungen können Sie zusätzliche praktische Erfahrung beim Erstellen von Funktionen sammeln.

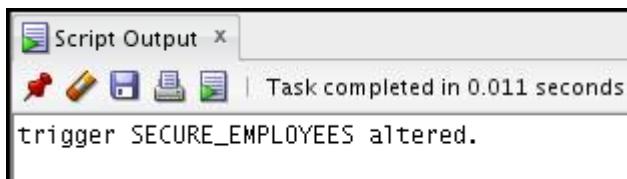
## Übung 5 zu Lektion 1 – Lösung: Mitarbeitergehälter überwachen

In dieser Übung erstellen Sie eine Prozedur, mit der Sie überwachen, ob Mitarbeiter das für ihre Tätigkeit gültige Durchschnittsgehalt überschritten haben.

- Deaktivieren Sie den Trigger SECURE\_EMPLOYEES.

**Entfernen Sie die Kommentarzeichen, und wählen Sie den Code in Aufgabe 1 der zusätzlichen Übung 5 zu Lektion 1. Der Code und die Ergebnisse werden wie folgt angezeigt:**

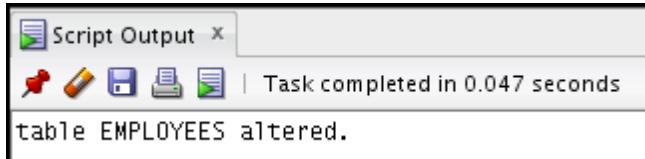
```
ALTER TRIGGER secure_employees DISABLE;
```



- Fügen Sie in der Tabelle EMPLOYEES eine Spalte EXCEED\_AVGSAL hinzu, um bis zu drei Zeichen und den Standardwert NO zu speichern. Verwenden Sie ein CHECK-Constraint, um die Werte YES oder NO zuzulassen.

**Entfernen Sie die Kommentarzeichen, und wählen Sie den Code in Aufgabe 2 der zusätzlichen Übung 5 zu Lektion 1. Der Code und die Ergebnisse werden wie folgt angezeigt:**

```
ALTER TABLE employees ADD (exceed_avgsal VARCHAR2(3) DEFAULT
  'NO'
  CONSTRAINT employees_exceed_avgsal_ck
  CHECK (exceed_avgsal IN ('YES', 'NO')));
```



- Erstellen Sie eine Stored Procedure namens CHECK\_AVGSAL, die prüft, ob alle Mitarbeitergehälter das für die JOB\_ID gültige Durchschnittsgehalt überschreiten.
  - Das Durchschnittsgehalt für eine Tätigkeit Job wird anhand der Informationen in der Tabelle JOBS berechnet.
  - Wenn das Gehalt des Mitarbeiters das für die Tätigkeit gültige Durchschnittsgehalt übersteigt, aktualisieren Sie die Spalte EXCEED\_AVGSAL in der Tabelle EXCEED\_AVGSAL auf den Wert YES. Stellen Sie den Wert andernfalls auf NO ein.
  - Verwenden Sie einen Cursor, um die Zeilen des Mitarbeiters mit der Option FOR UPDATE in der Abfrage auszuwählen.
  - Programmieren Sie die notwendige Exception-Behandlung für den Fall, dass ein Record gesperrt ist.

**Tipp:** Die Fehlernummer für gesperrte/belegte Ressourcen ist -54.

- e. Erstellen und verwenden Sie eine lokale Funktion namens GET\_JOB\_AVGSAL, um das Durchschnittsgehalt für eine als Parameter angegebene Tätigkeits-ID zu bestimmen.

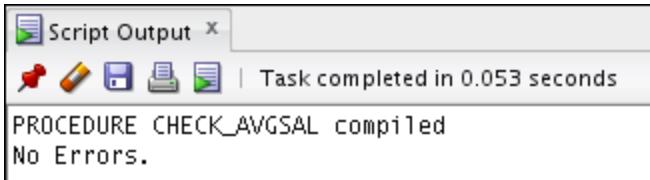
**Entfernen Sie die Kommentarzeichen, und wählen Sie den Code in Aufgabe 3 der zusätzlichen Übung 5 zu Lektion 1. Der Code und die Ergebnisse werden wie folgt angezeigt:**

```

CREATE OR REPLACE PROCEDURE check_avgsal IS
    emp_exceed_avgsal_type employees.exceed_avgsal%type;
    CURSOR c_emp_csr IS
        SELECT employee_id, job_id, salary
        FROM employees
        FOR UPDATE;
    e_resource_busy EXCEPTION;
    PRAGMA EXCEPTION_INIT(e_resource_busy, -54);
    FUNCTION get_job_avgsal (jobid VARCHAR2) RETURN NUMBER IS
        avg_sal employees.salary%type;
    BEGIN
        SELECT (max_salary + min_salary) / 2 INTO avg_sal
        FROM jobs
        WHERE job_id = jobid;
        RETURN avg_sal;
    END;

    BEGIN
        FOR emprec IN c_emp_csr
        LOOP
            emp_exceed_avgsal_type := 'NO';
            IF emprec.salary >= get_job_avgsal(emprec.job_id) THEN
                emp_exceed_avgsal_type := 'YES';
            END IF;
            UPDATE employees
            SET exceed_avgsal = emp_exceed_avgsal_type
            WHERE CURRENT OF c_emp_csr;
        END LOOP;
    EXCEPTION
        WHEN e_resource_busy THEN
            ROLLBACK;
            RAISE_APPLICATION_ERROR (-20001, 'Record is busy, try
later.');
        END check_avgsal;
    /
    SHOW ERRORS

```



4. Führen Sie die Prozedur CHECK\_AVGSAL aus. Um das Ergebnis Ihrer Änderungen anzuzeigen, erstellen Sie eine Abfrage zum Anzeigen der folgenden Informationen: Personalnummer, Tätigkeitskennung, Durchschnittsgehalt der Tätigkeit, Gehalt des Mitarbeiters und die Indikatorpalte exceed\_avgsal für Mitarbeiter, deren Gehalt das Durchschnittsgehalt für ihre Tätigkeit überschreitet. Schreiben Sie die Änderungen zuletzt fest.

**Hinweis:** Bei diesen Übungen können Sie zusätzliche praktische Erfahrung beim Erstellen von Funktionen sammeln.

**Entfernen Sie die Kommentarzeichen, und wählen Sie den Code in Aufgabe 4 der zusätzlichen Übung 5 zu Lektion 1. Der Code und die Ergebnisse werden wie folgt angezeigt:**

```
EXECUTE check_avgsal

SELECT e.employee_id, e.job_id, (j.max_salary-j.min_salary/2)
job_avgsal,
       e.salary, e.exceed_avgsal avg_exceeded
FROM   employees e, jobs j
WHERE  e.job_id = j.job_id
and e.exceed_avgsal = 'YES';

COMMIT;
```

Script Output X

| Task completed in 0.035 seconds

anonymous block completed

| EMPLOYEE_ID | JOB_ID     | JOB_AVGSAL | SALARY | AVG_EXCEEDED |
|-------------|------------|------------|--------|--------------|
| 113         | FI_ACCOUNT | 6900       | 6900   | YES          |
| 112         | FI_ACCOUNT | 6900       | 7800   | YES          |
| 111         | FI_ACCOUNT | 6900       | 7700   | YES          |
| 110         | FI_ACCOUNT | 6900       | 8200   | YES          |
| 109         | FI_ACCOUNT | 6900       | 9000   | YES          |
| 206         | AC_ACCOUNT | 6900       | 8300   | YES          |
| 174         | SA_REP     | 9008       | 11000  | YES          |
| 170         | SA_REP     | 9008       | 9600   | YES          |
| 169         | SA_REP     | 9008       | 10000  | YES          |
| 168         | SA_REP     | 9008       | 11500  | YES          |
| 163         | SA_REP     | 9008       | 9500   | YES          |
| 162         | SA_REP     | 9008       | 10500  | YES          |
| 157         | SA_REP     | 9008       | 9500   | YES          |
| 156         | SA_REP     | 9008       | 10000  | YES          |
| 151         | SA_REP     | 9008       | 9500   | YES          |
| 150         | SA_REP     | 9008       | 10000  | YES          |
| 122         | ST_MAN     | 5750       | 7900   | YES          |
| 121         | ST_MAN     | 5750       | 8200   | YES          |
| 120         | ST_MAN     | 5750       | 8000   | YES          |
| 137         | ST_CLERK   | 3996       | 3600   | YES          |
| 192         | SH_CLERK   | 4250       | 4000   | YES          |
| 185         | SH_CLERK   | 4250       | 4100   | YES          |
| 184         | SH_CLERK   | 4250       | 4200   | YES          |
| 103         | IT_PROG    | 8000       | 9000   | YES          |
| 201         | MK_MAN     | 10500      | 13000  | YES          |
| 203         | HR REP     | 7000       | 6500   | YES          |
| 204         | PR REP     | 8250       | 10000  | YES          |

27 rows selected

committed.

# Übung 6 zu Lektion 1 – Gesamtanzahl der Dienstjahre eines Mitarbeiters abrufen

---

## Überblick

In dieser Übung erstellen Sie ein Unterprogramm, um die Anzahl von Dienstjahren eines bestimmten Mitarbeiters zu ermitteln.

## Aufgaben

1. Erstellen Sie eine Stored Function namens `GET_YEARS_SERVICE`, die die Gesamtzahl der Dienstjahre eines bestimmten Mitarbeiters ermittelt. Die Funktion soll die Personalnummer als Parameter annehmen und die Anzahl der Dienstjahre zurückgeben. Programmieren Sie die notwendige Fehlerbehandlung für den Fall, dass eine ungültige Personalnummer angegeben wird.
2. Rufen Sie die Funktion `GET_YEARS_SERVICE` in einem Aufruf von `DBMS_OUTPUT.PUT_LINE` für einen Mitarbeiter mit der Nummer 999 auf.
3. Zeigen Sie die Anzahl von Dienstjahren des Mitarbeiters 106 mit `DBMS_OUTPUT.PUT_LINE` an, und rufen Sie dabei die Funktion `GET_YEARS_SERVICE` auf. Aktivieren Sie `SERVEROUTPUT`.
4. Fragen Sie die Daten des angegebenen Mitarbeiters aus den Tabellen `JOB_HISTORY` und `EMPLOYEES` ab, um zu prüfen, ob die Angaben zutreffen. Die in den Ergebnissen auf dieser Seite dargestellten Werte können von Ihren Abfrageergebnissen abweichen.

## Übung 6 zu Lektion 1 – Lösung: Gesamtanzahl der Dienstjahre eines Mitarbeiters abrufen

---

In dieser Übung erstellen Sie ein Unterprogramm, um die Anzahl von Dienstjahren eines bestimmten Mitarbeiters zu ermitteln.

1. Erstellen Sie eine Stored Function namens GET\_YEARS\_SERVICE, die die Gesamtzahl der Dienstjahre eines bestimmten Mitarbeiters ermittelt. Die Funktion soll die Personalnummer als Parameter annehmen und die Anzahl der Dienstjahre zurückgeben. Programmieren Sie die notwendige Fehlerbehandlung für den Fall, dass eine ungültige Personalnummer angegeben wird.

**Entfernen Sie die Kommentarzeichen, und wählen Sie den Code in Aufgabe 1 der zusätzlichen Übung 6 zu Lektion 1. Der Code und die Ergebnisse werden wie folgt angezeigt:**

```
CREATE OR REPLACE FUNCTION get_years_service(
    p_emp.empid_type IN employees.employee_id%TYPE) RETURN NUMBER
IS
    CURSOR c_jobh_csr IS
        SELECT MONTHS_BETWEEN(end_date, start_date)/12
    v_years_in_job
    v_years_service NUMBER(2) := 0;
    v_years_in_job NUMBER(2) := 0;
BEGIN
    FOR jobh_rec IN c_jobh_csr
    LOOP
        EXIT WHEN c_jobh_csr%NOTFOUND;
        v_years_service := v_years_service +
    jobh_rec.v_years_in_job;
    END LOOP;
    SELECT MONTHS_BETWEEN(SYSDATE, hire_date)/12 INTO
    v_years_in_job
    FROM employees
    WHERE employee_id = p_emp.empid_type;
    v_years_service := v_years_service + v_years_in_job;
    RETURN ROUND(v_years_service);
EXCEPTION
    WHEN NO_DATA_FOUND THEN
        RAISE_APPLICATION_ERROR(-20348,
            'Employee with ID '|| p_emp.empid_type || ' does not
            exist.');
        RETURN NULL;
END get_years_service;
```

```
/  
SHOW ERRORS
```

Script Output | Task completed in 0.062 seconds  
FUNCTION GET\_YEARS\_SERVICE compiled  
No Errors.

2. Rufen Sie die Funktion GET\_YEARS\_SERVICE in einem Aufruf von DBMS\_OUTPUT.PUT\_LINE für einen Mitarbeiter mit der Nummer 999 auf.

**Entfernen Sie die Kommentarzeichen, und wählen Sie den Code in Aufgabe 2 der zusätzlichen Übung 6 zu Lektion 1. Der Code und die Ergebnisse werden wie folgt angezeigt:**

```
SET SERVEROUTPUT ON  
EXECUTE DBMS_OUTPUT.PUT_LINE(get_years_service(999))
```

Script Output | Task completed in 2.38 seconds  
Error starting at line 3 in command:  
EXECUTE DBMS\_OUTPUT.PUT\_LINE(get\_years\_service(999))  
Error report:  
ORA-20348: Employee with ID 999 does not exist.  
ORA-06512: at "ORA62.GET\_YEARS\_SERVICE", line 22  
ORA-06512: at line 1

3. Zeigen Sie die Anzahl von Dienstjahren des Mitarbeiters 106 mit DBMS\_OUTPUT.PUT\_LINE an, und rufen Sie dabei die Funktion GET\_YEARS\_SERVICE auf. Aktivieren Sie SERVEROUTPUT.

**Entfernen Sie die Kommentarzeichen, und wählen Sie den Code in Aufgabe 1 der zusätzlichen Übung 6 zu Lektion 1. Der Code und die Ergebnisse werden wie folgt angezeigt:**

```
SET SERVEROUTPUT ON  
  
BEGIN  
  DBMS_OUTPUT.PUT_LINE (  
    'Employee 106 has worked ' || get_years_service(106) || '  
 years');  
END;  
/
```

Script Output | Task completed in 0.005 seconds  
anonymous block completed  
Employee 106 has worked 7 years

4. Fragen Sie die Daten des angegebenen Mitarbeiters aus den Tabellen JOB\_HISTORY und EMPLOYEES ab, um zu prüfen, ob die Angaben zutreffen. Die in den Ergebnissen auf dieser Seite dargestellten Werte können von Ihren Abfrageergebnissen abweichen.

**Entfernen Sie die Kommentarzeichen, und wählen Sie den Code in Aufgabe 4 der zusätzlichen Übung 6 zu Lektion 1. Der Code und die Ergebnisse werden wie folgt angezeigt:**

```
SELECT employee_id, job_id,
       MONTHS_BETWEEN(end_date, start_date)/12 duration
  FROM job_history;

SELECT job_id, MONTHS_BETWEEN(SYSDATE, hire_date)/12 duration
  FROM employees
 WHERE employee_id = 106;
```

The screenshot shows the 'Script Output' window from Oracle SQL Developer. The title bar says 'Script Output x'. Below it are toolbar icons for script, edit, file, and help. A message bar indicates 'Task completed in 0.008 seconds'. The main area displays two sets of query results. The first result set is a table with columns 'EMPLOYEE\_ID', 'JOB\_ID', and 'DURATION'. It lists 11 rows for various employees and their job durations. The second result set is a table with columns 'JOB\_ID' and 'DURATION', showing one row for 'SY\_ANAL' with a duration of 0.

| EMPLOYEE_ID | JOB_ID     | DURATION                                   |
|-------------|------------|--------------------------------------------|
| 102         | IT_PROG    | 5.52956989247311827956989247311827956989   |
| 101         | AC_ACCOUNT | 4.09946236559139784946236559139784946237   |
| 101         | AC_MGR     | 3.38172043010752688172043010752688172043   |
| 201         | MK_REP     | 3.83870967741935483870967741935483870968   |
| 114         | ST_CLERK   | 1.7688172043010752688172043010752688172    |
| 122         | ST_CLERK   | 0.9973118279569892473118279569892473118283 |
| 200         | AD_ASST    | 5.75                                       |
| 176         | SA REP     | 0.7688172043010752688172043010752688172042 |
| 176         | SA MAN     | 0.9973118279569892473118279569892473118283 |
| 200         | AC_ACCOUNT | 4.49731182795698924731182795698924731183   |
| 106         | IT_PROG    | 6.79549258263639984070091596973317403425   |

11 rows selected

| JOB_ID  | DURATION |
|---------|----------|
| SY_ANAL | 0        |

# Übung 7 zu Lektion 1 – Gesamtanzahl verschiedener Tätigkeiten eines Mitarbeiters abrufen

## Überblick

In dieser Übung erstellen Sie ein Programm, das die Anzahl verschiedener Tätigkeiten ermittelt, die ein Mitarbeiter im Laufe einer Beschäftigung ausgeübt hat.

## Aufgaben

1. Erstellen Sie eine Stored Function namens `GET_JOB_COUNT`, um die Gesamtzahl unterschiedlicher Jobs abzufragen, die ein Mitarbeiter ausgeübt hat.
  - a. Die Funktion soll die Personalnummer als Parameter annehmen und die Anzahl verschiedener Tätigkeiten des Mitarbeiters einschließlich der aktuellen Tätigkeit zurückgeben.
  - b. Programmieren Sie die notwendige Exception-Behandlung für den Fall, dass eine ungültige Mitarbeiter-ID angegeben wird.  
**Tipp:** Verwenden Sie die eindeutigen Tätigkeits-IDs in der Tabelle `JOB_HISTORY`, und schließen Sie die aktuelle Tätigkeits-ID aus, falls es sich um eine der Tätigkeiten handelt, die der Mitarbeiter schon einmal innehatte.
  - c. Erstellen Sie aus zwei Abfragen eine Vereinigungsmenge, und zählen Sie die in eine PL/SQL-Tabelle abgerufenen Zeilen.
  - d. Führen Sie einen `FETCH`-Vorgang mit `BULK COLLECT INTO` aus, um die eindeutigen Tätigkeiten des Mitarbeiters abzufragen.
2. Rufen Sie die Funktion für den Mitarbeiter mit der ID 176 auf. Aktivieren Sie `SERVEROUTPUT`.  
**Hinweis:** Mit diesen Übungen können Sie zusätzliche praktische Erfahrung beim Erstellen von Packages sammeln.

## Übung 7 zu Lektion 1 – Lösung: Gesamtanzahl verschiedener Tätigkeiten eines Mitarbeiters abrufen

---

In dieser Übung erstellen Sie ein Programm, das die Anzahl verschiedener Tätigkeiten ermittelt, die ein Mitarbeiter im Laufe einer Beschäftigung ausgeübt hat.

1. Erstellen Sie eine Stored Function namens GET\_JOB\_COUNT, um die Gesamtzahl unterschiedlicher Jobs abzufragen, die ein Mitarbeiter ausgeübt hat.
  - a. Die Funktion soll die Personalnummer als Parameter annehmen und die Anzahl verschiedener Tätigkeiten des Mitarbeiters einschließlich der aktuellen Tätigkeit zurückgeben.
  - b. Programmieren Sie die notwendige Exception-Behandlung für den Fall, dass eine ungültige Mitarbeiter-ID angegeben wird.

**Tipp:** Verwenden Sie die eindeutigen Tätigkeits-IDs in der Tabelle JOB\_HISTORY, und schließen Sie die aktuelle Tätigkeits-ID aus, falls es sich um eine der Tätigkeiten handelt, die der Mitarbeiter schon einmal ausgeübt hat.

  - c. Erstellen Sie aus zwei Abfragen eine Vereinigungsmenge, und zählen Sie die in eine PL/SQL-Tabelle abgerufenen Zeilen.
  - d. Führen Sie einen FETCH-Vorgang mit BULK COLLECT INTO aus, um die eindeutigen Tätigkeiten des Mitarbeiters abzufragen.

**Entfernen Sie die Kommentarzeichen, und wählen Sie den Code in Aufgabe 1 der zusätzlichen Übung 7 zu Lektion 1. Der Code und die Ergebnisse werden wie folgt angezeigt:**

```

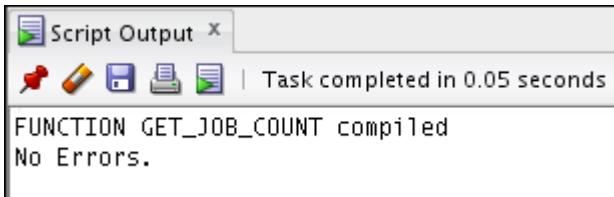
CREATE OR REPLACE FUNCTION get_job_count(
  p_emp.empid%TYPE) RETURN NUMBER
IS
  TYPE jobs_table_type IS TABLE OF jobs.job_id%type;
  v_jobtab jobs_table_type;
  CURSOR c_empjob_csr IS
    SELECT job_id
    FROM job_history
    WHERE employee_id = p_emp.empid%TYPE
    UNION
    SELECT job_id
    FROM employees
    WHERE employee_id = p_emp.empid%TYPE;
BEGIN
  OPEN c_empjob_csr;
  FETCH c_empjob_csr BULK COLLECT INTO v_jobtab;
  CLOSE c_empjob_csr;
  RETURN v_jobtab.count;
EXCEPTION
  WHEN NO_DATA_FOUND THEN

```

```

        RAISE_APPLICATION_ERROR(-20348,
        'Employee with ID '|| p_emp.empid_type || ' does not
        exist!');
        RETURN NULL;
END get_job_count;
/
SHOW ERRORS

```



2. Rufen Sie die Funktion für den Mitarbeiter mit der ID 176 auf. Aktivieren Sie SERVEROUTPUT.

**Hinweis:** Mit diesen Übungen können Sie zusätzliche praktische Erfahrung beim Erstellen von Packages sammeln.

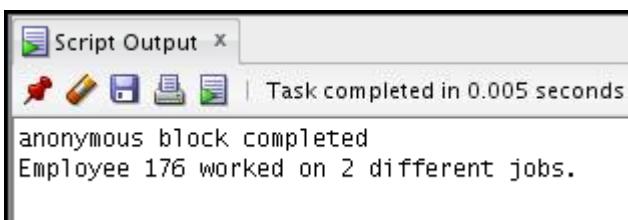
**Entfernen Sie die Kommentarzeichen, und wählen Sie den Code in Aufgabe 2 der zusätzlichen Übung 7 zu Lektion 1. Der Code und die Ergebnisse werden wie folgt angezeigt:**

```

SET SERVEROUTPUT ON

BEGIN
    DBMS_OUTPUT.PUT_LINE('Employee 176 worked on ' ||
        get_job_count(176) || ' different jobs.');
END;
/

```



## Übung 8 zu Lektion 1 – Neues Package erstellen, das die neu erstellten Prozeduren und Funktionen enthält

### Überblick

In dieser Übung erstellen Sie das Package `EMPJOB_PKG`, das die Prozeduren `NEW_JOB`, `ADD_JOB_HIST`, `UPD_JOBSAL` enthält sowie die Funktionen `GET_YEARS_SERVICE` und `GET_JOB_COUNT`.

### Aufgaben

1. Erstellen Sie die Packagespezifikation mit allen Unterprogrammkonstrukten als öffentliche Konstrukte. Verschieben Sie alle lokal definierten Unterprogrammtypen in die Packagespezifikation.
2. Erstellen Sie den Packagebody mit der Unterprogrammimplementierung. Entfernen Sie dabei unbedingt alle Unterprogrammtypen aus den Unterprogrammimplementierungen, die Sie in die Packagespezifikation verschoben haben.
3. Rufen Sie Ihre Prozedur `EMPJOB_PKG.NEW_JOB` auf, um eine neue Tätigkeit mit der ID `PR_MAN`, der Tätigkeitskennung Public Relations Manager und dem Gehalt 6250 zu erstellen. Aktivieren Sie `SERVEROUTPUT`.
4. Um die Tätigkeits-ID des Mitarbeiters mit der Nummer 110 in `PR_MAN` zu ändern, rufen Sie die Prozedur `EMPJOB_PKG.ADD_JOB_HIST` auf.

**Hinweis:** Deaktivieren den Trigger `UPDATE_JOB_HISTORY`, bevor Sie die Prozedur `ADD_JOB_HIST` ausführen können, und aktivieren Sie den Trigger nach Ausführung der Prozedur erneut.

5. Fragen Sie die Tabellen `JOBS`, `JOB_HISTORY` und `EMPLOYEES` ab, um die Ergebnisse zu prüfen.

**Hinweis:** Mit diesen Übungen können Sie zusätzliche praktische Erfahrung beim Erstellen von Datenbanktriggern sammeln.

## Übung 8 zu Lektion 1 – Lösung: Neues Package erstellen, das die neu erstellten Prozeduren und Funktionen enthält

In dieser Übung erstellen Sie das Package EMPJOB\_PKG, das die Prozeduren NEW\_JOB, ADD\_JOB\_HIST, UPD\_JOBSAL enthält sowie die Funktionen GET\_YEARS\_SERVICE und GET\_JOB\_COUNT.

1. Erstellen Sie die Packagespezifikation mit allen Unterprogrammkonstrukten als öffentliche Konstrukte. Verschieben Sie alle lokal definierten Unterprogrammtypen in die Packagespezifikation.

**Entfernen Sie die Kommentarzeichen, und wählen Sie den Code in Aufgabe 1 der zusätzlichen Übung 8 zu Lektion 1. Der Code und die Ergebnisse werden wie folgt angezeigt:**

```
CREATE OR REPLACE PACKAGE empjob_pkg IS
    TYPE jobs_table_type IS TABLE OF jobs.job_id%TYPE;

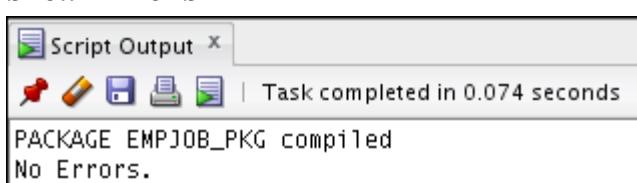
    PROCEDURE add_job_hist(
        p_emp_id IN employees.employee_id%TYPE,
        p_new_jobid IN jobs.job_id%TYPE);

    FUNCTION get_job_count(
        p_emp_id IN employees.employee_id%TYPE) RETURN NUMBER;

    FUNCTION get_years_service(
        p_emp_id IN employees.employee_id%TYPE) RETURN NUMBER;

    PROCEDURE new_job(
        p_jobid IN jobs.job_id%TYPE,
        p_title IN jobs.job_title%TYPE,
        p_minsal IN jobs.min_salary%TYPE);

    PROCEDURE upd_jobsal(
        p_jobid IN jobs.job_id%TYPE,
        p_new_minsal IN jobs.min_salary%TYPE,
        p_new_maxsal IN jobs.max_salary%TYPE);
END empjob_pkg;
/
SHOW ERRORS
```



2. Erstellen Sie den Packagebody mit der Unterprogrammimplementierung. Entfernen Sie dabei alle Unterprogrammtypen aus den Unterprogrammimplementierungen, die Sie in die Packagespezifikation verschoben haben.

**Entfernen Sie die Kommentarzeichen, und wählen Sie den Code in Aufgabe 2 der zusätzlichen Übung 8 zu Lektion 1. Der Code und die Ergebnisse werden wie folgt angezeigt:**

```

CREATE OR REPLACE PACKAGE BODY empjob_pkg IS
  PROCEDURE add_job_hist(
    p_emp_id IN employees.employee_id%TYPE,
    p_new_jobid IN jobs.job_id%TYPE) IS
  BEGIN
    INSERT INTO job_history
      SELECT employee_id, hire_date, SYSDATE, job_id,
             department_id
        FROM employees
       WHERE employee_id = p_emp_id;
    UPDATE employees
      SET hire_date = SYSDATE,
          job_id = p_new_jobid,
          salary = (SELECT min_salary + 500
                     FROM jobs
                    WHERE job_id = p_new_jobid)
       WHERE employee_id = p_emp_id;
    DBMS_OUTPUT.PUT_LINE ('Added employee ' || p_emp_id ||
                          ' details to the JOB_HISTORY table');
    DBMS_OUTPUT.PUT_LINE ('Updated current job of employee ' ||
                          p_emp_id|| ' to '|| p_new_jobid);
  EXCEPTION
    WHEN NO_DATA_FOUND THEN
      RAISE_APPLICATION_ERROR (-20001, 'Employee does not
                                exist!');
    END add_job_hist;

  FUNCTION get_job_count(
    p_emp_id IN employees.employee_id%TYPE) RETURN NUMBER IS
    v_jobtab jobs_table_type;
    CURSOR c_empjob_csr IS
      SELECT job_id
        FROM job_history
       WHERE employee_id = p_emp_id
         UNION
      SELECT job_id
        FROM employees
  
```

```

        WHERE employee_id = p_emp_id;
BEGIN
    OPEN c_empjob_csr;
    FETCH c_empjob_csr BULK COLLECT INTO v_jobtab;
    CLOSE c_empjob_csr;
    RETURN v_jobtab.count;
EXCEPTION
    WHEN NO_DATA_FOUND THEN
        RAISE_APPLICATION_ERROR(-20348,
            'Employee with ID '|| p_emp_id ||' does not exist!');
        RETURN 0;
END get_job_count;

FUNCTION get_years_service(
    p_emp_id IN employees.employee_id%TYPE) RETURN NUMBER IS
CURSOR c_jobh_csr IS
    SELECT MONTHS_BETWEEN(end_date, start_date)/12
v_years_in_job
    FROM job_history
    WHERE employee_id = p_emp_id;
    v_years_service NUMBER(2) := 0;
    v_years_in_job NUMBER(2) := 0;
BEGIN
    FOR jobh_rec IN c_jobh_csr
    LOOP
        EXIT WHEN c_jobh_csr%NOTFOUND;
        v_years_service := v_years_service +
jobh_rec.v_years_in_job;
    END LOOP;
    SELECT MONTHS_BETWEEN(SYSDATE, hire_date)/12 INTO
v_years_in_job
    FROM employees
    WHERE employee_id = p_emp_id;
    v_years_service := v_years_service + v_years_in_job;
    RETURN ROUND(v_years_service);
EXCEPTION
    WHEN NO_DATA_FOUND THEN
        RAISE_APPLICATION_ERROR(-20348,
            'Employee with ID '|| p_emp_id ||' does not exist.');
        RETURN 0;
END get_years_service;

PROCEDURE new_job(

```

```

p_jobid IN jobs.job_id%TYPE,
p_title IN jobs.job_title%TYPE,
p_minsal IN jobs.min_salary%TYPE) IS
v_maxsal jobs.max_salary%TYPE := 2 * p_minsal;
BEGIN
    INSERT INTO jobs(job_id, job_title, min_salary, max_salary)
VALUES (p_jobid, p_title, p_minsal, v_maxsal);
    DBMS_OUTPUT.PUT_LINE ('New row added to JOBS table:');
    DBMS_OUTPUT.PUT_LINE (p_jobid || ' ' || p_title || ' ' ||
                           p_minsal || ' ' || v_maxsal);
END new_job;

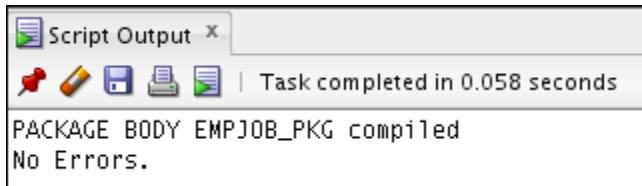
PROCEDURE upd_jobsal(
p_jobid IN jobs.job_id%type,
p_new_minsal IN jobs.min_salary%type,
p_new_maxsal IN jobs.max_salary%type) IS
v_dummy PLS_INTEGER;
e_resource_busy EXCEPTION;
e_sal_error EXCEPTION;
PRAGMA EXCEPTION_INIT (e_resource_busy , -54);
BEGIN
    IF (p_new_maxsal < p_new_minsal) THEN
        RAISE e_sal_error;
    END IF;
    SELECT 1 INTO v_dummy
    FROM jobs
    WHERE job_id = p_jobid
    FOR UPDATE OF min_salary NOWAIT;
    UPDATE jobs
        SET min_salary = p_new_minsal,
            max_salary = p_new_maxsal
    WHERE job_id = p_jobid;
EXCEPTION
    WHEN e_resource_busy THEN
        RAISE_APPLICATION_ERROR (-20001,
                               'Job information is currently locked, try later.');
    WHEN NO_DATA_FOUND THEN
        RAISE_APPLICATION_ERROR(-20001, 'This job ID does not
exist');
    WHEN e_sal_error THEN
        RAISE_APPLICATION_ERROR(-20001,
                               'Data error: Max salary should be more than min
salary');

```

```

    END upd_jobsal;
END empjob_pkg;
/
SHOW ERRORS

```



- IRufen Sie Ihre Prozedur EMPJOB\_PKG.NEW\_JOB auf, um eine neue Tätigkeit mit der ID PR\_MAN, der Tätigkeitskennung Public Relations Manager und dem Gehalt 6250 zu erstellen. Aktivieren Sie SERVEROUTPUT.

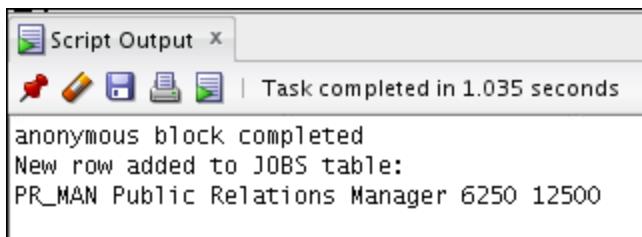
**Entfernen Sie die Kommentarzeichen, und wählen Sie den Code in Aufgabe 3 der zusätzlichen Übung 8 zu Lektion 1. Der Code und die Ergebnisse werden wie folgt angezeigt:**

```

SET SERVEROUTPUT ON

EXECUTE empjob_pkg.new_job('PR_MAN', 'Public Relations Manager',
6250)

```



- Um die Tätigkeits-ID des Mitarbeiters mit der Nummer 110 in PR\_MAN zu ändern, rufen Sie die Prozedur EMPJOB\_PKG.ADD\_JOB\_HIST auf.

**Hinweis:** Sie müssen den Trigger UPDATE\_JOB\_HISTORY deaktivieren, bevor Sie die Prozedur ADD\_JOB\_HIST ausführen können, und den Trigger nach Ausführung der Prozedur erneut aktivieren.

**Entfernen Sie die Kommentarzeichen, und wählen Sie den Code in Aufgabe 4 der zusätzlichen Übung 8 zu Lektion 1. Der Code und die Ergebnisse werden wie folgt angezeigt:**

```

SET SERVEROUTPUT ON
ALTER TRIGGER update_job_history DISABLE;
EXECUTE empjob_pkg.add_job_hist(110, 'PR_MAN')
ALTER TRIGGER update_job_history ENABLE;

```

Script Output | Task completed in 0.02 seconds

```

trigger UPDATE_JOB_HISTORY altered.
anonymous block completed
Added employee 110 details to the JOB_HISTORY table
Updated current job of employee 110 to PR_MAN

trigger UPDATE_JOB_HISTORY altered.

```

- Fragen Sie die Tabellen JOBS, JOB\_HISTORY und EMPLOYEES ab, um die Ergebnisse zu prüfen.

**Hinweis:** Mit diesen Übungen können Sie zusätzliche praktische Erfahrung beim Erstellen von Datenbanktriggern sammeln.

**Entfernen Sie die Kommentarzeichen, und wählen Sie den Code in Aufgabe 5 der zusätzlichen Übung 8 zu Lektion 1. Der Code und die Ergebnisse werden wie folgt angezeigt:**

```

SELECT * FROM jobs WHERE job_id = 'PR_MAN';
SELECT * FROM job_history WHERE employee_id = 110;
SELECT job_id, salary FROM employees WHERE employee_id = 110;

```

Script Output | Task completed in 0.019 seconds

| JOB_ID | JOB_TITLE                | MIN_SALARY | MAX_SALARY |
|--------|--------------------------|------------|------------|
| PR_MAN | Public Relations Manager | 6250       | 12500      |

| EMPLOYEE_ID | START_DATE | END_DATE  | JOB_ID     | DEPARTMENT_ID |
|-------------|------------|-----------|------------|---------------|
| 110         | 28-SEP-05  | 22-NOV-12 | FI_ACCOUNT | 100           |

| JOB_ID | SALARY |
|--------|--------|
| PR_MAN | 6750   |

## Übung 9 zu Lektion 1 – Trigger erstellen, um sicherzustellen, dass die Mitarbeitergehälter innerhalb des akzeptablen Bereichs liegen

### Überblick

In dieser Übung erstellen Sie einen Trigger, der sicherstellt, dass das Mindest- und Höchstgehalt einer Tätigkeit niemals so geändert werden können, dass es für Mitarbeiter mit der angegebenen Tätigkeitskennung außerhalb der für die Tätigkeit festgelegten neuen Gehaltsspanne liegt.

### Aufgaben

1. Erstellen Sie den Trigger `CHECK_SAL_RANGE`. Er soll vor den Zeilen ausgelöst werden, die in den Spalten `MIN_SALARY` und `MAX_SALARY` der Tabelle `JOBS` aktualisiert werden.
  - a. Prüfen Sie für jeden geänderten Mindest- oder Höchstgehaltswert, ob das Gehalt eines bestehenden Mitarbeiters mit der betreffenden Tätigkeits-ID in der Tabelle `EMPLOYEES` in dem neuen Gehaltsbereich liegt, der für die Tätigkeits-ID festgelegt wurde.
  - b. Sehen Sie eine Exception-Behandlung vor, um eine Gehaltsbereichsänderung abzudecken, die sich auf den Record eines vorhandenen Mitarbeiters auswirkt.
2. Testen Sie den Trigger mit der Tätigkeits-ID `SY_ANAL`, indem Sie das neue Mindestgehalt auf 5000 und das neue Höchstgehalt auf 7000 einstellen. Erstellen Sie vor der Änderung eine Abfrage, mit der die aktuelle Gehaltsspanne für die Tätigkeits-ID `SY_ANAL` angezeigt wird, und eine weitere Abfrage, mit der Personalnummer, Nachnamen und Gehalt für dieselbe Tätigkeits-ID angezeigt wird. Führen Sie nach der Aktualisierung eine Abfrage der (eventuellen) Änderungen an der Tabelle `JOBS` für die angegebene Tätigkeits-ID durch.
3. Stellen Sie mithilfe der Tätigkeits-ID `SY_ANAL` das neue Mindestgehalt auf 7000 und das neue Höchstgehalt auf 18000 ein. Erläutern Sie die Ergebnisse.

## Übung 9 zu Lektion 1 – Lösung: Trigger erstellen, um sicherzustellen, dass die Mitarbeitergehälter innerhalb des akzeptablen Bereichs liegen

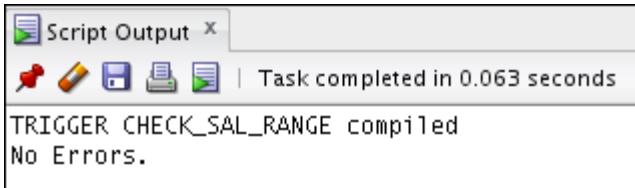
---

In dieser Übung erstellen Sie einen Trigger, der sicherstellt, dass das Mindest- und Höchstgehalt einer Tätigkeit niemals so geändert werden können, dass es für Mitarbeiter mit der angegebenen Tätigkeitskennung außerhalb der für die Tätigkeit festgelegten neuen Gehaltsspanne liegt.

1. Erstellen Sie den Trigger `CHECK_SAL_RANGE`. Er soll vor den Zeilen ausgelöst werden, die in den Spalten `MIN_SALARY` und `MAX_SALARY` der Tabelle `JOBES` aktualisiert werden.
  - a. Prüfen Sie für jeden geänderten Mindest- oder Höchstgehaltswert, ob das Gehalt eines bestehenden Mitarbeiters mit der betreffenden Tätigkeits-ID in der Tabelle `EMPLOYEES` in dem neuen Gehaltsbereich liegt, der für die Tätigkeits-ID festgelegt wurde.
  - b. Sehen Sie eine Exception-Behandlung vor, um eine Gehaltsbereichsänderung abzudecken, die sich auf den Record eines vorhandenen Mitarbeiters auswirkt.

**Entfernen Sie die Kommentarzeichen, und wählen Sie den Code in Aufgabe 1 der zusätzlichen Übung 9 zu Lektion 1. Der Code und die Ergebnisse werden wie folgt angezeigt:**

```
CREATE OR REPLACE TRIGGER check_sal_range
BEFORE UPDATE OF min_salary, max_salary ON jobs
FOR EACH ROW
DECLARE
  v_minsal employees.salary%TYPE;
  v_maxsal employees.salary%TYPE;
  e_invalid_salrange EXCEPTION;
BEGIN
  SELECT MIN(salary), MAX(salary) INTO v_minsal, v_maxsal
  FROM employees
  WHERE job_id = :NEW.job_id;
  IF (v_minsal < :NEW.min_salary) OR (v_maxsal >
:NEW.max_salary) THEN
    RAISE e_invalid_salrange;
  END IF;
EXCEPTION
  WHEN e_invalid_salrange THEN
    RAISE_APPLICATION_ERROR(-20550,
      'Employees exist whose salary is out of the specified
      range. ' ||
      'Therefore the specified salary range cannot be updated.');
END check_sal_range;
/
SHOW ERRORS
```



2. Testen Sie den Trigger mit der Tätigkeits-ID SY\_ANAL, indem Sie das neue Mindestgehalt auf 5000 und das neue Höchstgehalt auf 7000 einstellen. Erstellen Sie vor der Änderung eine Abfrage, mit der die aktuelle Gehaltsspanne für die Tätigkeits-ID SY\_ANAL angezeigt wird, und eine weitere Abfrage, mit der Personalnummer, Nachnamen und Gehalt für dieselbe Tätigkeits-ID angezeigt wird. Führen Sie nach der Aktualisierung eine Abfrage der (eventuellen) Änderungen an der Tabelle JOBS für die angegebene Tätigkeits-ID durch.

**Entfernen Sie die Kommentarzeichen, und wählen Sie den Code in Aufgabe 2 der zusätzlichen Übung 9 zu Lektion 1. Der Code und die Ergebnisse werden wie folgt angezeigt:**

```
SELECT * FROM jobs
WHERE job_id = 'SY_ANAL';

SELECT employee_id, last_name, salary
FROM employees
WHERE job_id = 'SY_ANAL';

UPDATE jobs
SET min_salary = 5000, max_salary = 7000
WHERE job_id = 'SY_ANAL';

SELECT * FROM jobs
WHERE job_id = 'SY_ANAL';
```

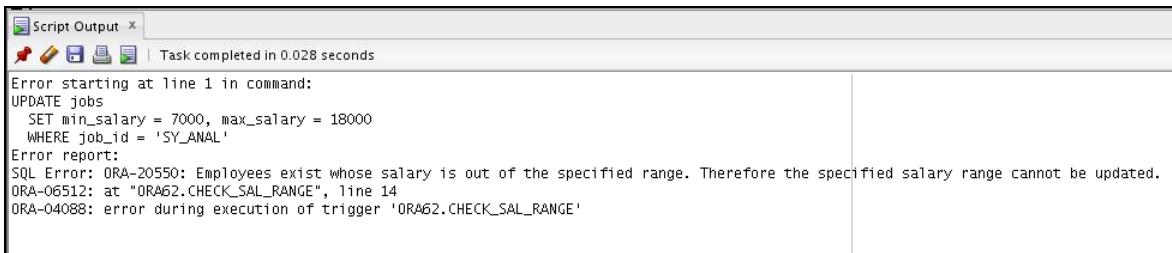
The screenshot shows the 'Script Output' window with the following content:

```
Script Output X
| Task completed in 0.018 seconds
-----+-----+-----+
JOB_ID | JOB_TITLE | MIN_SALARY | MAX_SALARY |
-----+-----+-----+
SY_ANAL | System Analyst | 7000 | 14000 |
-----+-----+-----+
EMPLOYEE_ID | LAST_NAME | SALARY |
-----+-----+-----+
106 | Pataballa | 6500 |
-----+-----+-----+
1 rows updated.
-----+-----+-----+
JOB_ID | JOB_TITLE | MIN_SALARY | MAX_SALARY |
-----+-----+-----+
SY_ANAL | System Analyst | 5000 | 7000 |
-----+-----+-----+
```

3. Stellen Sie mithilfe der Tätigkeits-ID SY\_ANAL das neue Mindestgehalt auf 7000 und das neue Höchstgehalt auf 18000 ein. Erläutern Sie die Ergebnisse.

**Entfernen Sie die Kommentarzeichen, und wählen Sie den Code in Aufgabe 3 der zusätzlichen Übung 9 zu Lektion 1. Der Code und die Ergebnisse werden wie folgt angezeigt:**

```
UPDATE jobs
   SET min_salary = 7000, max_salary = 18000
 WHERE job_id = 'SY_ANAL';
```



The screenshot shows the 'Script Output' window from Oracle SQL Developer. It displays the following text:

```
Script Output X
Task completed in 0.028 seconds
Error starting at line 1 in command:
UPDATE jobs
  SET min_salary = 7000, max_salary = 18000
 WHERE job_id = 'SY_ANAL'
Error report:
SQL Error: ORA-20550: Employees exist whose salary is out of the specified range. Therefore the specified salary range cannot be updated.
ORA-06512: at "ORA62.CHECK_SAL_RANGE", line 14
ORA-04088: error during execution of trigger 'ORA62.CHECK_SAL_RANGE'
```

**Die Aktualisierung kann die Gehaltsspanne aufgrund der durch den Trigger CHECK\_SAL\_RANGE zur Verfügung gestellten Funktion nicht ändern, da der Mitarbeiter 106 mit der Tätigkeits-ID SY\_ANAL ein Gehalt von 6500 hat. Dieser Wert liegt unter dem Mindestgehalt für die neue Gehaltsspanne, die in der Anweisung UPDATE angegeben wurde.**



## **Zusätzliche Übungen 2**

### **Kapitel 14**

## Zusätzliche Übungen 2 – Überblick

---

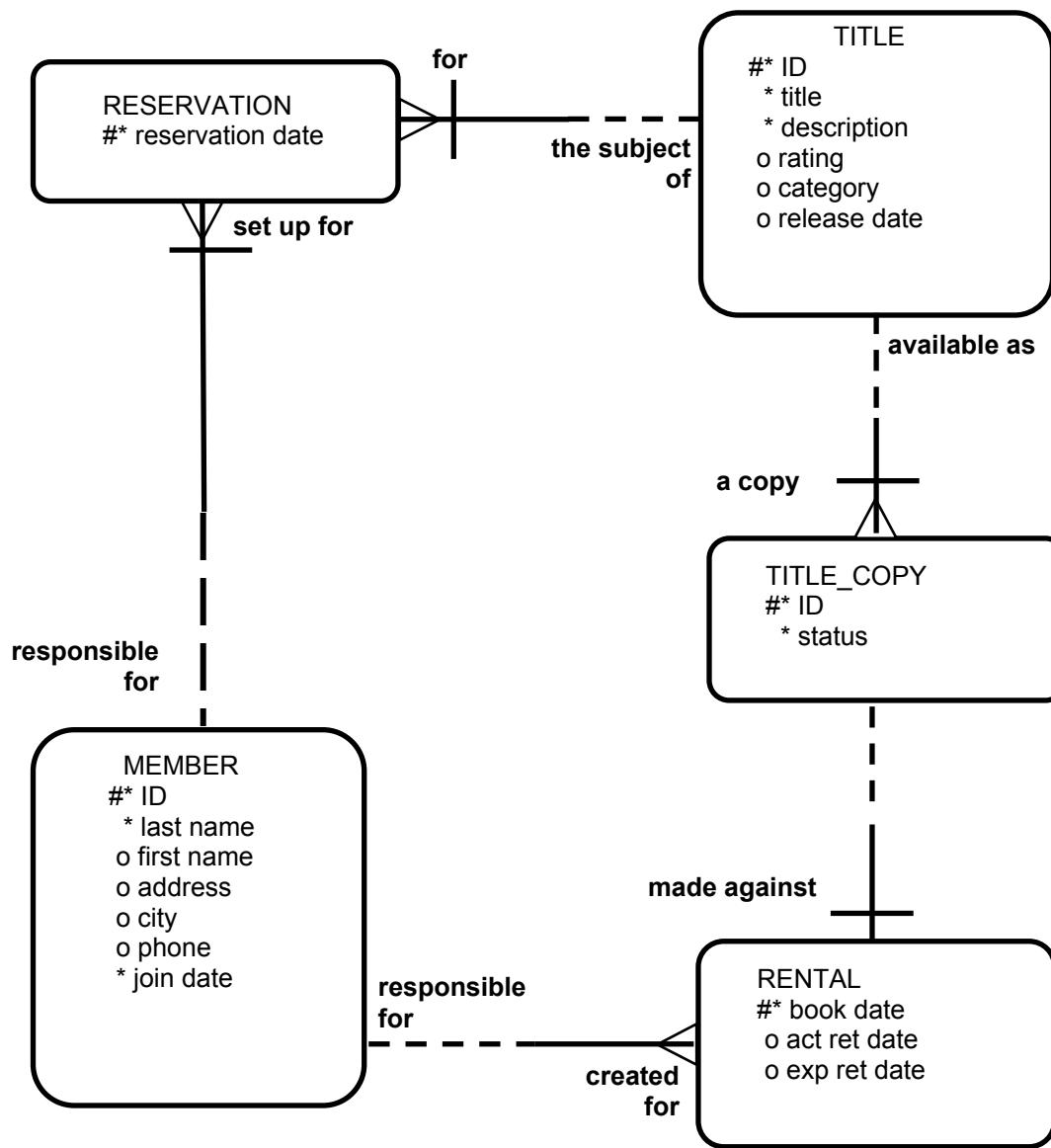
### Lektionsüberblick

In dieser Fallstudie erstellen Sie das Package VIDEO\_PKG, das Prozeduren und Funktionen für die Anwendung eines Videoverleihs enthält. Diese Anwendung ermöglicht es, Kunden zu Mitgliedern des Videoverleihs zu machen. Mitglieder können Videos ausleihen, ausgeliehene Videos zurückbringen und Videos reservieren. Sie erstellen zudem einen Trigger, um sicherzustellen, dass Daten in den Videoverleihtabellen nur während der Geschäftszeiten geändert werden.

Erstellen Sie das Package mit SQL\*Plus, und verwenden Sie das von Oracle bereitgestellte Package DBMS\_OUTPUT für die Anzeige von Meldungen.

Die Datenbank des Videoverleihs enthält folgende Tabellen: TITLE, TITLE\_COPY, RENTAL, RESERVATION und MEMBER.

## Entity Relationship-Diagramm der Datenbank des Videoverleihs



# Übung 1 zu Lektion 2 – Package VIDEO\_PKG erstellen

---

## Überblick

In dieser Übung erstellen Sie das Package VIDEO\_PKG, das Prozeduren und Funktionen für eine Videoverleihanwendung enthält.

## Aufgabe

1. Laden Sie das Skript `/home/oracle/labs/plpu/labs/buildvid1.sql`, und führen Sie es aus, um alle erforderlichen Tabellen und Sequences zu erstellen, die für diese Übung benötigt werden.
2. Laden Sie das Skript `/home/oracle/labs/plpu/labs/buildvid2.sql`, und führen Sie es aus, um alle im Skript buildvid1.sql erstellten Tabellen mit Daten zu füllen.
3. Erstellen Sie das Package VIDEO\_PKG, das folgende Prozeduren und Funktionen enthält:
  - a. **NEW\_MEMBER:** Eine öffentliche Prozedur, die der Tabelle MEMBER ein neues Mitglied hinzufügt. Verwenden Sie die Sequence MEMBER\_ID\_SEQ als Mitgliedsnummer und SYSDATE als Eintrittsdatum. Übergeben Sie alle anderen Werte, die in eine neue Zeile eingefügt werden sollen, als Parameter.
  - b. **NEW\_RENTAL:** Eine überladene öffentliche Funktion, die einen neuen Verleihvorgang aufzeichnet. Übergeben Sie die Film-ID für das Video, das der Kunde ausleihen möchte, und entweder den Nachnamen oder die Mitgliedsnummer des Kunden an die Funktion. Die Funktion soll das Rückgabedatum für das Video zurückgeben. Videos müssen drei Tage nach dem Verleihdatum zurückgegeben werden. Ist der Status einer Filmkopie in der Tabelle TITLE\_COPY als AVAILABLE angegeben, aktualisieren Sie die Tabelle, und ändern Sie den Status in RENTED. Ist keine Kopie verfügbar, soll die Funktion NULL zurückgeben. Fügen Sie in diesem Fall einen neuen Record in die Tabelle RENTAL ein. Geben Sie dabei das Reservierungsdatum (Datum des aktuellen Tages), die Kennnummer des Videos, die Mitgliedsnummer des Kunden, die Film-ID und das erwartete Rückgabedatum an. Beachten Sie, dass mehrere Kunden mit demselben Nachnamen vorhanden sein können. In diesem Fall soll die Funktion NULL zurückgeben und eine Liste der Kunden mit diesem Namen und den zugehörigen Mitgliedsnummern anzeigen.
  - c. **RETURN\_MOVIE:** Eine öffentliche Prozedur, die den Status eines Videos (AVAILABLE, RENTED oder DAMAGED) aktualisiert und das Rückgabedatum festlegt. Übergeben Sie die Film-ID, die Kennnummer des Videos und den Status an diese Prozedur. Prüfen Sie, ob Reservierungen für dieses Video vorliegen, und zeigen Sie in diesem Fall eine Meldung an. Aktualisieren Sie die Tabelle RENTAL, und tragen Sie das Datum des aktuellen Tages als Rückgabedatum ein. Aktualisieren Sie den Status in der Tabelle TITLE\_COPY auf Basis des Statusparameters, der an die Prozedur übergeben wurde.
  - d. **RESERVE\_MOVIE:** Eine private Prozedur, die nur ausgeführt wird, wenn der Status aller in der Prozedur NEW\_RENTAL angeforderten Filmkopien RENTED lautet. Übergeben Sie die Mitgliedsnummer und die Film-ID an diese Prozedur. Fügen Sie einen neuen Record in die Tabelle RESERVATION ein, und speichern Sie das Reservierungsdatum, die Mitgliedsnummer und die Film-ID. Geben Sie eine Meldung aus, die besagt, dass das Video reserviert wurde, und das erwartete Rückgabedatum nennt.

- e. **EXCEPTION\_HANDLER:** Eine private Prozedur, die vom Exception Handler der öffentlichen Programme aufgerufen wird. Übergeben Sie die Nummer SQLCODE und den Namen des Programms, in dem der Fehler auftrat, als Textzeichenfolge an die Prozedur. Verwenden Sie RAISE\_APPLICATION\_ERROR, um einen benutzerdefinierten Fehler auszulösen. Beginnen Sie mit einem Unique-Schlüsselfehler (-1) und einem Fremdschlüsselfehler (-2292). Bei anderen Fehlern soll der Exception Handler einen allgemeinen Fehler auslösen.
4. Testen Sie Ihre Routinen mit den folgenden Skripten im Verzeichnis /home/oracle/labs/plpu/soln:
    - a. Fügen Sie zwei Mitglieder hinzu. Verwenden Sie dazu den Code in Aufgabe 4\_a aus dem Skript sol\_ap2.sql.
    - b. Fügen Sie zwei Videoverleihvorgänge hinzu. Verwenden Sie dazu den Code in der Aufgabe 4\_b aus dem Skript sol\_ap2.sql.
    - c. Geben Sie Filme zurück. Verwenden Sie dazu den Code in der Aufgabe 4\_c aus dem Skript sol\_ap2.sql.
  5. Die Öffnungszeiten des Videoverleihs sind sonntags bis freitags von 8 bis 22 Uhr und samstags von 8 bis 24 Uhr. Um sicherzustellen, dass die Tabellen nur während dieser Zeiten geändert werden können, erstellen Sie eine Stored Procedure, die von den für die Tabellen definierten Triggern aufgerufen wird.
    - a. Erstellen Sie die Stored Procedure TIME\_CHECK, die die aktuelle Uhrzeit mit den Öffnungszeiten vergleicht. Liegt die aktuelle Uhrzeit außerhalb der Öffnungszeiten, soll die Prozedur RAISE\_APPLICATION\_ERROR eine entsprechende Meldung ausgeben.
    - b. Erstellen Sie für jede der fünf Tabellen einen Trigger. Der Trigger soll ausgelöst werden, bevor Daten in die Tabellen eingefügt, aktualisiert oder gelöscht werden. Rufen Sie die Prozedur TIME\_CHECK aus den einzelnen Triggern auf.
    - c. Testen Sie die Trigger.

**Hinweis:** Damit der Trigger nicht erfolgreich ausgeführt wird, müssen Sie die Zeitspanne so festlegen, dass die aktuelle Uhrzeit außerhalb der Zeitspanne liegt. Für den Test könnten Sie beispielsweise 18 bis 8 Uhr als gültige Öffnungszeiten in Ihrem Trigger festlegen.

## Übung 1 zu Lektion 2 – Lösung: Package VIDEO\_PKG erstellen

---

In dieser Übung erstellen Sie das Package VIDEO\_PKG, das Prozeduren und Funktionen für eine Videoverleihanwendung enthält.

1. Laden Sie das Skript /home/oracle/labs/plpu/labs/buildvid1.sql, und führen Sie es aus, um alle erforderlichen Tabellen und Sequences zu erstellen, die für diese Übung benötigt werden.

**Führen Sie das Skript /home/oracle/labs/plpu/labs/buildvid1.sql aus. Code, Verbindungsaufordern und Ergebnisse werden wie folgt angezeigt:**

```
SET ECHO OFF
/* Script to build the Video Application (Part 1 - buildvid1.sql)
   for the Oracle Introduction to Oracle with Procedure Builder
course.
   Created by: Debby Kramer Creation date: 12/10/95
   Last updated: 11/21/12
   Modified by Supriya Ananth on 21-NOV-2012
   For the course Oracle Database: PL/SQL Program Units
   This part of the script creates tables and sequences that are
used
   by Task 4 of the Additional Practices of the course.
*/
DROP TABLE rental CASCADE CONSTRAINTS;
DROP TABLE reservation CASCADE CONSTRAINTS;
DROP TABLE title_copy CASCADE CONSTRAINTS;
DROP TABLE title CASCADE CONSTRAINTS;
DROP TABLE member CASCADE CONSTRAINTS;

PROMPT Please wait while tables are created.....

CREATE TABLE MEMBER
  (member_id    NUMBER (10)          CONSTRAINT member_id_pk PRIMARY
KEY
  , last_name   VARCHAR2(25)
    CONSTRAINT member_last_nn NOT NULL
  , first_name  VARCHAR2(25)
  , address     VARCHAR2(100)
  , city        VARCHAR2(30)
  , phone       VARCHAR2(25)
  , join_date   DATE DEFAULT SYSDATE
    CONSTRAINT join_date_nn NOT NULL)
/
CREATE TABLE TITLE
  (title_id    NUMBER(10)
    CONSTRAINT title_id_pk PRIMARY KEY
  , title       VARCHAR2(60)
    CONSTRAINT title_nn NOT NULL
```

```

, description VARCHAR2(400)
    CONSTRAINT title_desc_nn NOT NULL
, rating      VARCHAR2(4)
    CONSTRAINT title_rating_ck CHECK (rating IN
('G','PG','R','NC17','NR'))
, category     VARCHAR2(20) DEFAULT 'DRAMA'
    CONSTRAINT title_categ_ck CHECK (category IN
('DRAMA','COMEDY','ACTION',
'CHILD','SCIFI','DOCUMENTARY'))
, release_date DATE
/
CREATE TABLE TITLE_COPY
(copy_id      NUMBER(10)
, title_id    NUMBER(10)
    CONSTRAINT copy_title_id_fk
        REFERENCES title(title_id)
, status      VARCHAR2(15)
    CONSTRAINT copy_status_nn NOT NULL
    CONSTRAINT copy_status_ck CHECK (status IN ('AVAILABLE',
'DESTROYED',
'RENTED', 'RESERVED'))
, CONSTRAINT copy_title_id_pk PRIMARY KEY(copy_id, title_id))
/
CREATE TABLE RENTAL
(book_date DATE DEFAULT SYSDATE
, copy_id    NUMBER(10)
, member_id  NUMBER(10)
    CONSTRAINT rental_mbr_id_fk REFERENCES member(member_id)
, title_id   NUMBER(10)
, act_ret_date DATE
, exp_ret_date DATE DEFAULT SYSDATE+2
, CONSTRAINT rental_copy_title_id_fk FOREIGN KEY (copy_id,
title_id)
    REFERENCES title_copy(copy_id,title_id)
, CONSTRAINT rental_id_pk PRIMARY KEY(book_date, copy_id,
title_id, member_id))
/
CREATE TABLE RESERVATION
(res_date    DATE
, member_id  NUMBER(10)
, title_id   NUMBER(10)
, CONSTRAINT res_id_pk PRIMARY KEY(res_date, member_id, title_id))
/
PROMPT Tables created.

DROP SEQUENCE title_id_seq;
DROP SEQUENCE member_id_seq;

PROMPT Creating Sequences...

```

```
CREATE SEQUENCE member_id_seq
    START WITH 100
    NOCACHE
/
CREATE SEQUENCE title_id_seq
    START WITH 91
    NOCACHE
/
PROMPT Sequences created.

PROMPT Run buildvid2.sql now to populate the above tables.
```



```
Script Output X
| Task completed in 1.53 seconds

Error starting at line 12 in command:
DROP TABLE rental CASCADE CONSTRAINTS
Error report:
SQL Error: ORA-00942: table or view does not exist
00942. 00000 - "table or view does not exist"
*Cause:
*Action:

Error starting at line 13 in command:
DROP TABLE reservation CASCADE CONSTRAINTS
Error report:
SQL Error: ORA-00942: table or view does not exist
00942. 00000 - "table or view does not exist"
*Cause:
*Action:

Error starting at line 14 in command:
DROP TABLE title_copy CASCADE CONSTRAINTS
Error report:
SQL Error: ORA-00942: table or view does not exist
00942. 00000 - "table or view does not exist"
*Cause:
*Action:

Error starting at line 15 in command:
DROP TABLE title CASCADE CONSTRAINTS
Error report:
SQL Error: ORA-00942: table or view does not exist
00942. 00000 - "table or view does not exist"
*Cause:
*Action:

Error starting at line 16 in command:
DROP TABLE member CASCADE CONSTRAINTS
Error report:
SQL Error: ORA-00942: table or view does not exist
00942. 00000 - "table or view does not exist"
```

```

*Cause:
>Action:
Please wait while tables are created....
table MEMBER created.
table TITLE created.
table TITLE_COPY created.
table RENTAL created.
table RESERVATION created.
Tables created.

Error starting at line 80 in command:
DROP SEQUENCE title_id_seq
>Error report:
SQL Error: ORA-02289: sequence does not exist
02289. 00000 - "sequence does not exist"
*Cause: The specified sequence does not exist, or the user does
not have the required privilege to perform this operation.
>Action: Make sure the sequence name is correct, and that you have
the right to perform the desired operation on this sequence.

Error starting at line 81 in command:
DROP SEQUENCE member_id_seq
>Error report:
SQL Error: ORA-02289: sequence does not exist
02289. 00000 - "sequence does not exist"
*Cause: The specified sequence does not exist, or the user does
not have the required privilege to perform this operation.
>Action: Make sure the sequence name is correct, and that you have
the right to perform the desired operation on this sequence.

Creating Sequences...
sequence MEMBER_ID_SEQ created.
sequence TITLE_ID_SEQ created.
Sequences created.
Run buildvid2.sql now to populate the above tables.

```

2. Laden Sie das Skript /home/oracle/labs/plpu/labs/buildvid2.sql, und führen Sie es aus, um alle im Skript buildvid1.sql erstellten Tabellen mit Daten zu füllen.

**Führen Sie das Skript /home/oracle/labs/plpu/labs/buildvid2.sql aus. Code, Verbindungsauforderung und Ergebnisse werden wie folgt angezeigt:**

```

/* Script to build the Video Application (Part 2 -
buildvid2.sql)

This part of the script populates the tables that are created
using buildvid1.sql

These are used by Part B of the Additional Practices of the
course.

You should run the script buildvid1.sql before running this
script to create the above tables.

*/

```

```

INSERT INTO member VALUES  (member_id_seq.NEXTVAL, 'Velasquez',
'Carmen', '283 King Street', 'Seattle', '587-99-6666', '03-MAR-
90');

```

```

INSERT INTO member VALUES (member_id_seq.NEXTVAL, 'Ngao',
'LaDoris', '5 Modrany', 'Bratislava', '586-355-8882', '08-MAR-
90');

INSERT INTO member VALUES (member_id_seq.NEXTVAL, 'Nagayama',
'Midori', '68 Via Centrale', 'Sao Paolo', '254-852-5764', '17-
JUN-91');

INSERT INTO member VALUES (member_id_seq.NEXTVAL, 'Quick-To-
See', 'Mark', '6921 King Way', 'Lagos', '63-559-777', '07-APR-
90');

INSERT INTO member VALUES (member_id_seq.NEXTVAL, 'Ropeburn',
'Audry', '86 Chu Street', 'Hong Kong', '41-559-87', '04-MAR-
90');

INSERT INTO member VALUES (member_id_seq.NEXTVAL, 'Urguhart',
'Molly', '3035 Laurier Blvd.', 'Quebec', '418-542-9988', '18-
JAN-91');

INSERT INTO member VALUES (member_id_seq.NEXTVAL, 'Menchu',
'Roberta', 'Boulevard de Waterloo 41', 'Brussels', '322-504-
2228', '14-MAY-90');

INSERT INTO member VALUES (member_id_seq.NEXTVAL, 'Biri', 'Ben',
'398 High St.', 'Columbus', '614-455-9863', '07-APR-90');

INSERT INTO member VALUES (member_id_seq.NEXTVAL, 'Catchpole',
'Antoinette', '88 Alfred St.', 'Brisbane', '616-399-1411', '09-
FEB-92');

COMMIT;

```

```

INSERT INTO TITLE (title_id, title, description, rating,
category, release_date)
VALUES (TITLE_ID_SEQ.NEXTVAL, 'Willie and Christmas Too', 'All
of Willie''s friends made a Christmas list for Santa, but Willie
has yet to create his own wish list.', 'G', 'CHILD', '05-OCT-
95');

INSERT INTO TITLE (title_id, title, description, rating,
category, release_date)
VALUES (TITLE_ID_SEQ.NEXTVAL, 'Alien Again', 'Another
installment of science fiction history. Can the heroine save the
planet from the alien life form?', 'R', 'SCIFI', '19-
MAY-95');

INSERT INTO TITLE (title_id, title, description, rating,
category, release_date)
VALUES (TITLE_ID_SEQ.NEXTVAL, 'The Glob', 'A meteor crashes near
a small American town and unleashes carnivorous goo in this
classic.', 'NR', 'SCIFI', '12-AUG-95');

INSERT INTO TITLE (title_id, title, description, rating,
category, release_date)

```

```

VALUES (TITLE_ID_SEQ.NEXTVAL, 'My Day Off', 'With a little luck
and a lot of ingenuity, a teenager skips school for a day in New
York.', 'PG', 'COMEDY', '12-JUL-95');
INSERT INTO TITLE (title_id, title, description, rating,
category, release_date)
VALUES (TITLE_ID_SEQ.NEXTVAL, 'Miracles on Ice', 'A six-year-old
has doubts about Santa Claus. But she discovers that miracles
really do exist.', 'PG', 'DRAMA', '12-SEP-95');
INSERT INTO TITLE (title_id, title, description, rating,
category, release_date)
VALUES (TITLE_ID_SEQ.NEXTVAL, 'Soda Gang', 'After discovering a
cache of drugs, a young couple find themselves pitted against a
vicious gang.', 'NR', 'ACTION', '01-JUN-95');
INSERT INTO title (title_id, title, description, rating,
category, release_date)
VALUES (TITLE_ID_SEQ.NEXTVAL, 'Interstellar Wars', 'Futuristic
interstellar action movie. Can the rebels save the humans from
the evil Empire?', 'PG', 'SCIFI','07-JUL-77');

COMMIT;

INSERT INTO title_copy VALUES (1,92, 'AVAILABLE');
INSERT INTO title_copy VALUES (1,93, 'AVAILABLE');
INSERT INTO title_copy VALUES (2,93, 'RENTED');
INSERT INTO title_copy VALUES (1,94, 'AVAILABLE');
INSERT INTO title_copy VALUES (1,95, 'AVAILABLE');
INSERT INTO title_copy VALUES (2,95, 'AVAILABLE');
INSERT INTO title_copy VALUES (3,95, 'RENTED');
INSERT INTO title_copy VALUES (1,96, 'AVAILABLE');
INSERT INTO title_copy VALUES (1,97, 'AVAILABLE');
COMMIT;

INSERT INTO reservation VALUES (sysdate-1, 101, 93);
INSERT INTO reservation VALUES (sysdate-2, 106, 102);

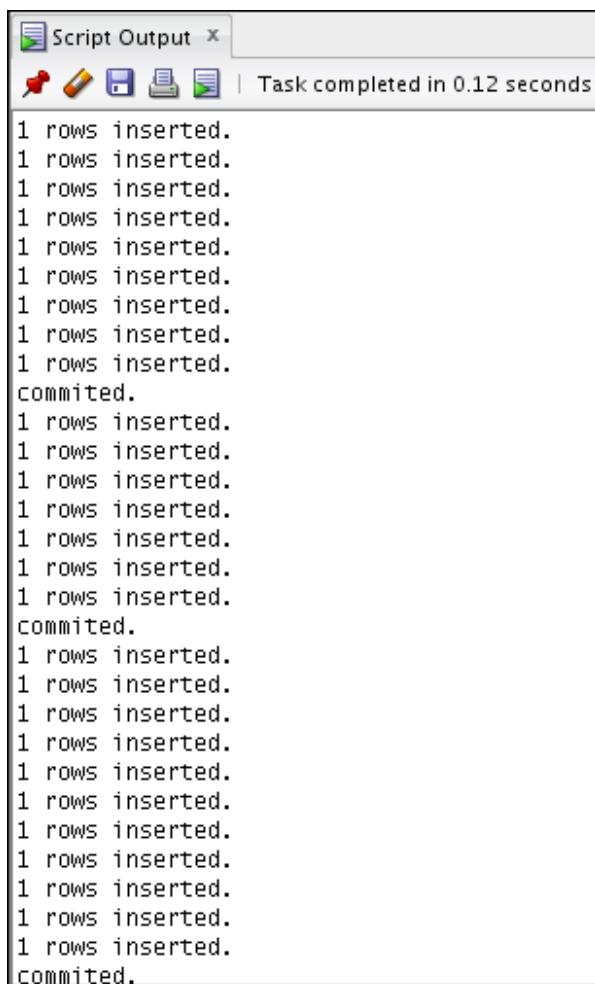
COMMIT;

INSERT INTO rental VALUES (sysdate-1, 2, 101, 93, null,
sysdate+1);
INSERT INTO rental VALUES (sysdate-2, 3, 102, 95, null,
sysdate);
INSERT INTO rental VALUES (sysdate-4, 1, 106, 97, sysdate-2,
sysdate-2);

```

```
INSERT INTO rental VALUES (sysdate-3, 1, 101, 92, sysdate-2,  
sysdate-1);  
  
COMMIT;
```

PROMPT \*\* Tables built and data loaded \*\*



```

1 rows inserted.
1 rows inserted.
committed.
1 rows inserted.
committed.
** Tables built and data loaded **

```

3. Erstellen Sie das Package `VIDEO_PKG`, das folgende Prozeduren und Funktionen enthält:
  - a. **NEW\_MEMBER:** Eine öffentliche Prozedur, die der Tabelle `MEMBER` ein neues Mitglied hinzufügt. Verwenden Sie die Sequence `MEMBER_ID_SEQ` als Mitgliedsnummer und `SYSDATE` als Eintrittsdatum. Übergeben Sie alle anderen Werte, die in eine neue Zeile eingefügt werden sollen, als Parameter.
  - b. **NEW\_RENTAL:** Eine überladene öffentliche Funktion, die einen neuen Verleihvorgang aufzeichnet. Übergeben Sie die Film-ID für das Video, das der Kunde ausleihen möchte, und entweder den Nachnamen oder die Mitgliedsnummer des Kunden an die Funktion. Die Funktion soll das Rückgabedatum für das Video zurückgeben. Videos müssen drei Tage nach dem Verleihdatum zurückgegeben werden. Ist der Status einer Filmkopie in der Tabelle `TITLE_COPY` als `AVAILABLE` angegeben, aktualisieren Sie die Tabelle, und ändern Sie den Status in `RENTED`. Ist keine Kopie verfügbar, soll die Funktion `NULL` zurückgeben. Fügen Sie in diesem Fall einen neuen Record in die Tabelle `RENTAL` ein. Geben Sie dabei das Reservierungsdatum (Datum des aktuellen Tages), die Kennnummer des Videos, die Mitgliedsnummer des Kunden, die Film-ID und das erwartete Rückgabedatum an. Beachten Sie, dass mehrere Kunden mit demselben Nachnamen vorhanden sein können. In diesem Fall soll die Funktion `NULL` zurückgeben und eine Liste der Kunden mit diesem Namen und den zugehörigen Mitgliedsnummern anzeigen.
  - c. **RETURN\_MOVIE:** Eine öffentliche Prozedur, die den Status eines Videos (`AVAILABLE`, `RENTED` oder `DAMAGED`) aktualisiert und das Rückgabedatum festlegt. Übergeben Sie die Film-ID, die Kennnummer des Videos und den Status an diese Prozedur. Prüfen Sie, ob Reservierungen für dieses Video vorliegen, und zeigen Sie in diesem Fall eine Meldung an. Aktualisieren Sie die Tabelle `RENTAL`, und tragen Sie das Datum des aktuellen Tages als Rückgabedatum ein. Aktualisieren Sie den Status in der Tabelle `TITLE_COPY` auf Basis des Statusparameters, der an die Prozedur übergeben wurde.
  - d. **RESERVE\_MOVIE:** Eine private Prozedur, die nur ausgeführt wird, wenn der Status aller in der Prozedur `NEW_RENTAL` angeforderten Filmkopien `RENTED` lautet. Übergeben Sie die Mitgliedsnummer und die Film-ID an diese Prozedur. Fügen Sie einen neuen Record in die Tabelle `RESERVATION` ein, und speichern Sie das Reservierungsdatum, die Mitgliedsnummer und die Film-ID. Geben Sie eine Meldung aus, die besagt, dass das Video reserviert wurde, und das erwartete Rückgabedatum nennt.
  - e. **EXCEPTION\_HANDLER:** Eine private Prozedur, die vom Exception Handler der öffentlichen Programme aufgerufen wird. Übergeben Sie die Nummer `SQLCODE` und (als Textzeichenfolge) den Namen des Programms, in dem der Fehler auftrat, an diese Prozedur. Verwenden Sie `RAISE_APPLICATION_ERROR`, um einen benutzerdefinierten Fehler auszulösen. Beginnen Sie mit einem Unique-Schlüsselfehler (-1) und einem Fremdschlüsselfehler (-2292). Bei anderen Fehlern soll der Exception Handler einen allgemeinen Fehler auslösen.

**Entfernen Sie die Kommentarzeichen, und führen Sie den Code in der Aufgabe 3 des Skripts /home/oracle/labs/plpu/solns/sol\_ap2.sql aus. Code, Verbindungsauforderung und Ergebnisse werden wie folgt angezeigt:**

### **VIDEO\_PKG Package Specification**

```

CREATE OR REPLACE PACKAGE video_pkg IS
  PROCEDURE new_member
    (p_lname      IN member.last_name%TYPE,
     p_fname      IN member.first_name%TYPE      DEFAULT NULL,
     p_address    IN member.address%TYPE         DEFAULT NULL,
     p_city       IN member.city%TYPE            DEFAULT NULL,
     p_phone      IN member.phone%TYPE           DEFAULT NULL);

  FUNCTION new_rental
    (p_memberid   IN rental.member_id%TYPE,
     p_titleid    IN rental.title_id%TYPE)
    RETURN DATE;

  FUNCTION new_rental
    (p_membername IN member.last_name%TYPE,
     p_titleid    IN rental.title_id%TYPE)
    RETURN DATE;

  PROCEDURE return_movie
    (p_titleid    IN rental.title_id%TYPE,
     p_copyid     IN rental.copy_id%TYPE,
     p_sts        IN title_copy.status%TYPE);
END video_pkg;
/
SHOW ERRORS

CREATE OR REPLACE PACKAGE BODY video_pkg IS
  PROCEDURE exception_handler(errcode IN NUMBER, p_context IN VARCHAR2) IS
  BEGIN
    IF errcode = -1 THEN
      RAISE_APPLICATION_ERROR(-20001,
        'The number is assigned to this member is already in use,
        |||'
        'try again.');
    ELSIF errcode = -2291 THEN
      RAISE_APPLICATION_ERROR(-20002, p_context ||
        ' has attempted to use a foreign key value that is
        invalid');
    ELSE
      RAISE_APPLICATION_ERROR(-20999, 'Unhandled error in ' ||
        p_context || '. Please contact your application ' ||
        'administrator with the following information:
        || CHR(13) || SQLERRM);
    END IF;
  END;

```

```

        END IF;
    END exception_handler;

PROCEDURE reserve_movie
    (p_memberid  IN reservation.member_id%TYPE,
     p_titleid   IN reservation.title_id%TYPE) IS
CURSOR c_rented_csr IS
    SELECT exp_ret_date
      FROM rental
     WHERE title_id = p_titleid
       AND act_ret_date IS NULL;
BEGIN
    INSERT INTO reservation (res_date, member_id, title_id)
VALUES (SYSDATE, p_memberid, p_titleid);
    COMMIT;
    FOR rented_rec IN c_rented_csr LOOP
        DBMS_OUTPUT.PUT_LINE('Movie reserved. Expected back on: '
        || rented_rec.exp_ret_date);
        EXIT WHEN c_rented_csr%found;
    END LOOP;
EXCEPTION
    WHEN OTHERS THEN
        exception_handler(SQLCODE, 'RESERVE_MOVIE');
END reserve_movie;

PROCEDURE return_movie(
    p_titleid IN rental.title_id%TYPE,
    p_copyid  IN rental.copy_id%TYPE,
    p_sts     IN title_copy.status%TYPE) IS
v_dummy VARCHAR2(1);
CURSOR c_res_csr IS
    SELECT *
      FROM reservation
     WHERE title_id = p_titleid;
BEGIN
    SELECT '' INTO v_dummy
      FROM title
     WHERE title_id = p_titleid;
    UPDATE rental
        SET act_ret_date = SYSDATE
      WHERE title_id = p_titleid
        AND copy_id = p_copyid AND act_ret_date IS NULL;
    UPDATE title_copy
        SET status = UPPER(p_sts)
      WHERE title_id = p_titleid AND copy_id = p_copyid;
    FOR res_rec IN c_res_csr LOOP
        IF c_res_csr%FOUND THEN
            DBMS_OUTPUT.PUT_LINE('Put this movie on hold -- ' ||
            'reserved by member #' || res_rec.member_id);
        END IF;
    END LOOP;
EXCEPTION
    WHEN OTHERS THEN

```

```

        exception_handler(SQLCODE, 'RETURN_MOVIE');
END return_movie;

FUNCTION new_rental(
    p_memberid  IN rental.member_id%TYPE,
    p_titleid   IN rental.title_id%TYPE) RETURN DATE IS
CURSOR c_copy_csr IS
    SELECT * FROM title_copy
    WHERE title_id = p_titleid
    FOR UPDATE;
    v_flag      BOOLEAN := FALSE;
BEGIN
    FOR copy_rec IN c_copy_csr LOOP
        IF copy_rec.status = 'AVAILABLE' THEN
            UPDATE title_copy
                SET status = 'RENTED'
                WHERE CURRENT OF c_copy_csr;
            INSERT INTO rental(book_date, copy_id, member_id,
                               title_id, exp_ret_date)
            VALUES (SYSDATE, copy_rec.copy_id, p_memberid,
                    p_titleid, SYSDATE + 3);
            v_flag := TRUE;
            EXIT;
        END IF;
    END LOOP;
    COMMIT;
    IF v_flag THEN
        RETURN (SYSDATE + 3);
    ELSE
        reserve_movie(p_memberid, p_titleid);
        RETURN NULL;
    END IF;
EXCEPTION
    WHEN OTHERS THEN
        exception_handler(SQLCODE, 'NEW_RENTAL');
        RETURN NULL;
END new_rental;

FUNCTION new_rental(
    p_membername IN member.last_name%TYPE,
    p_titleid   IN rental.title_id%TYPE) RETURN DATE IS
CURSOR c_copy_csr IS
    SELECT * FROM title_copy
    WHERE title_id = p_titleid
    FOR UPDATE;
    v_flag      BOOLEAN := FALSE;
    v_memberid  member.member_id%TYPE;
    CURSOR c_member_csr IS
        SELECT member_id, last_name, first_name
        FROM member
        WHERE LOWER(last_name) = LOWER(p_membername)
        ORDER BY last_name, first_name;
BEGIN

```

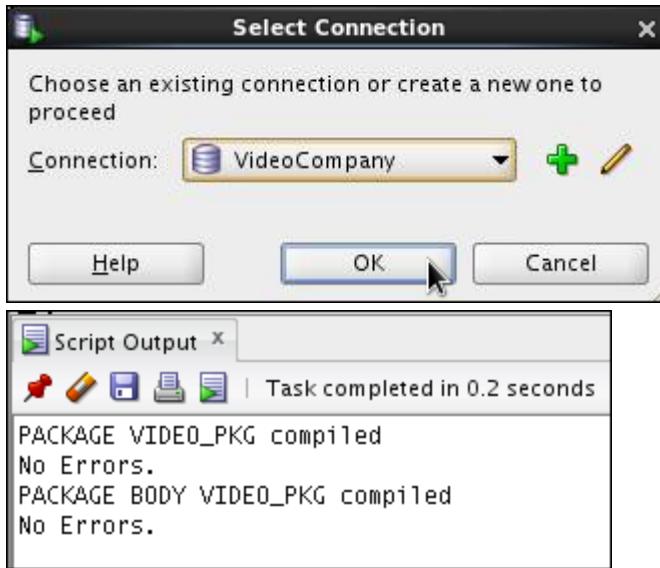
```

SELECT member_id INTO v_memberid
  FROM member
 WHERE lower(last_name) = lower(p_membername);
FOR copy_rec IN c_copy_csr LOOP
  IF copy_rec.status = 'AVAILABLE' THEN
    UPDATE title_copy
      SET status = 'RENTED'
        WHERE CURRENT OF c_copy_csr;
    INSERT INTO rental (book_date, copy_id, member_id,
                        title_id, exp_ret_date)
      VALUES (SYSDATE, copy_rec.copy_id, v_memberid,
              p_titleid, SYSDATE + 3);
    v_flag := TRUE;
    EXIT;
  END IF;
END LOOP;
COMMIT;
IF v_flag THEN
  RETURN(SYSDATE + 3);
ELSE
  reserve_movie(v_memberid, p_titleid);
  RETURN NULL;
END IF;
EXCEPTION
  WHEN TOO_MANY_ROWS THEN
    DBMS_OUTPUT.PUT_LINE(
      'Warning! More than one member by this name.');
    FOR member_rec IN c_member_csr LOOP
      DBMS_OUTPUT.PUT_LINE(member_rec.member_id || CHR(9) ||
                           member_rec.last_name || ', ' || member_rec.first_name);
    END LOOP;
    RETURN NULL;
  WHEN OTHERS THEN
    exception_handler(SQLCODE, 'NEW_RENTAL');
    RETURN NULL;
END new_rental;

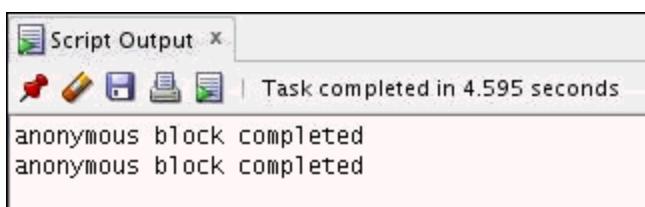
PROCEDURE new_member(
  p_lname      IN member.last_name%TYPE,
  p_fname       IN member.first_name%TYPE      DEFAULT NULL,
  p_address     IN member.address%TYPE         DEFAULT NULL,
  p_city        IN member.city%TYPE            DEFAULT NULL,
  p_phone       IN member.phone%TYPE           DEFAULT NULL) IS
BEGIN
  INSERT INTO member(member_id, last_name, first_name,
                     address, city, phone, join_date)
    VALUES(member_id_seq.NEXTVAL, p_lname, p_fname,
           p_address, p_city, p_phone, SYSDATE);
  COMMIT;
EXCEPTION
  WHEN OTHERS THEN
    exception_handler(SQLCODE, 'NEW_MEMBER');
END new_member;

```

```
END video_pkg;
/
SHOW ERRORS
```



4. Testen Sie Ihre Routinen mit den folgenden Skripten im Verzeichnis /home/oracle/labs/plpu/soln. Aktivieren Sie SERVEROUTPUT:
    - a. Fügen Sie mit dem Code in Aufgabe 4\_a zwei Mitglieder hinzu.  
**Entfernen Sie die Kommentarzeichen, und führen Sie den Code in Aufgabe 4\_a aus. Code und Ergebnisse werden wie folgt angezeigt:**
- ```
EXECUTE video_pkg.new_member('Haas', 'James', 'Chestnut Street',
  'Boston', '617-123-4567')
EXECUTE video_pkg.new_member('Biri', 'Allan', 'Hiawatha
Drive', 'New York', '516-123-4567')
```



4. Fügen Sie mit dem Code in Aufgabe 4\_b neue Videoverleihvorgänge hinzu.  
**Entfernen Sie die Kommentarzeichen, und führen Sie den Code in Aufgabe 4\_b aus. Code und Ergebnisse werden wie folgt angezeigt:**

```
SET SERVEROUTPUT ON
```

```
EXEC DBMS_OUTPUT.PUT_LINE(video_pkg.new_rental(110, 98))
EXEC DBMS_OUTPUT.PUT_LINE(video_pkg.new_rental(109, 93))
EXEC DBMS_OUTPUT.PUT_LINE(video_pkg.new_rental(107, 98))
```

```
EXEC DBMS_OUTPUT.PUT_LINE(video_pkg.new_rental('Biri', 97))
```

The screenshot shows the 'Script Output' window from Oracle SQL Developer. It displays the following text:

```
anonymous block completed
anonymous block completed
14-DEC-12
anonymous block completed
anonymous block completed
Warning! More than one member by this name.
110 Biri, Allan
107 Biri, Ben
```

- c. Geben Sie die Videos zurück. Verwenden Sie den Code in Aufgabe 4\_c.

**Entfernen Sie die Kommentarzeichen, und führen Sie den Code in Aufgabe 4\_b aus. Code und Ergebnisse werden wie folgt angezeigt:**

```
SET SERVEROUTPUT ON
```

```
EXECUTE video_pkg.return_movie(92, 3, 'AVAILABLE')
EXECUTE video_pkg.return_movie(95, 3, 'AVAILABLE')
EXECUTE video_pkg.return_movie(93, 1, 'RENTED')
```

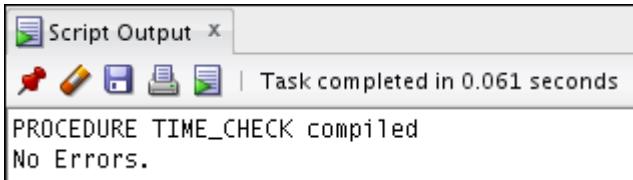
The screenshot shows the 'Script Output' window from Oracle SQL Developer. It displays the following text:

```
anonymous block completed
anonymous block completed
anonymous block completed
Put this movie on hold -- reserved by member #101
```

5. Die Öffnungszeiten des Videoverleihs sind sonntags bis freitags von 8 bis 22 Uhr und samstags von 8 bis 24 Uhr. Um sicherzustellen, dass die Tabellen nur während dieser Zeiten geändert werden können, erstellen Sie eine Stored Procedure, die von den für die Tabellen definierten Triggern aufgerufen wird.
- Erstellen Sie die Stored Procedure `TIME_CHECK`, die die aktuelle Uhrzeit mit den Öffnungszeiten vergleicht. Liegt die aktuelle Uhrzeit außerhalb der Öffnungszeiten, soll die Prozedur `RAISE_APPLICATION_ERROR` eine entsprechende Meldung ausgeben.

**Entfernen Sie die Kommentarzeichen, und führen Sie den Code in Aufgabe 5\_a aus. Code und Ergebnisse werden wie folgt angezeigt:**

```
CREATE OR REPLACE PROCEDURE time_check IS
BEGIN
    IF ((TO_CHAR(SYSDATE, 'D') BETWEEN 1 AND 6) AND
        (TO_DATE(TO_CHAR(SYSDATE, 'hh24:mi'), 'hh24:mi') NOT
        BETWEEN
            TO_DATE('08:00', 'hh24:mi') AND TO_DATE('22:00',
            'hh24:mi'))) OR ((TO_CHAR(SYSDATE, 'D') = 7)
        AND (TO_DATE(TO_CHAR(SYSDATE, 'hh24:mi'), 'hh24:mi') NOT
        BETWEEN
            TO_DATE('08:00', 'hh24:mi') AND TO_DATE('24:00',
            'hh24:mi'))) THEN
        RAISE_APPLICATION_ERROR(-20999,
            'Data changes restricted to office hours.');
    END IF;
END time_check;
/
SHOW ERRORS
```



- b. Erstellen Sie für jede der fünf Tabellen einen Trigger. Der Trigger soll ausgelöst werden, bevor Daten in die Tabellen eingefügt, aktualisiert oder gelöscht werden. Rufen Sie die Prozedur TIME\_CHECK aus den einzelnen Triggern auf.

**Entfernen Sie die Kommentarzeichen, und führen Sie den Code in Aufgabe 5\_b aus. Code und Ergebnisse werden wie folgt angezeigt:**

```
CREATE OR REPLACE TRIGGER member_trig
    BEFORE INSERT OR UPDATE OR DELETE ON member
    CALL time_check
/

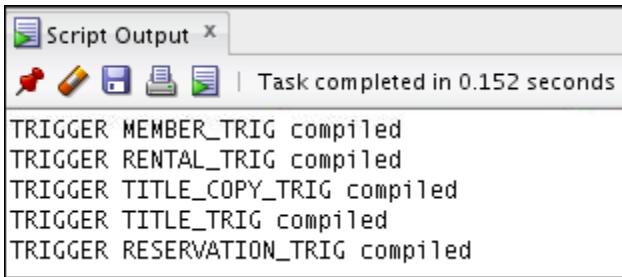
CREATE OR REPLACE TRIGGER rental_trig
    BEFORE INSERT OR UPDATE OR DELETE ON rental
    CALL time_check
/

CREATE OR REPLACE TRIGGER title_copy_trig
    BEFORE INSERT OR UPDATE OR DELETE ON title_copy
    CALL time_check
```

```

/
CREATE OR REPLACE TRIGGER title_trig
    BEFORE INSERT OR UPDATE OR DELETE ON title
CALL time_check
/
CREATE OR REPLACE TRIGGER reservation_trig
    BEFORE INSERT OR UPDATE OR DELETE ON reservation
CALL time_check
/

```



- c. Testen Sie die Trigger.

**Hinweis:** Damit der Trigger nicht erfolgreich ausgeführt wird, müssen Sie die Zeitspanne so festlegen, dass die aktuelle Uhrzeit außerhalb der Zeitspanne liegt. Für den Test könnten Sie beispielsweise 18 bis 8 Uhr als gültige Öffnungszeiten in Ihrem Trigger festlegen.

**Entfernen Sie die Kommentarzeichen, und führen Sie den Code in Aufgabe 5\_c aus. Code und Ergebnisse werden wie folgt angezeigt:**

```

-- First determine current timezone and time
SELECT SESSIONTIMEZONE,
       TO_CHAR(CURRENT_DATE, 'DD-MON-YYYY HH24:MI') CURR_DATE
  FROM DUAL;

-- Change your time zone usinge [+|-]HH:MI format such that --
-- the current time returns a time between 6pm and 8am

ALTER SESSION SET TIME_ZONE='+07:00';

-- Add a new member (for a sample test)

EXECUTE video_pkg.new_member('Elias', 'Elliane', 'Vine
Street', 'California', '789-123-4567')

BEGIN video_pkg.new_member('Elias', 'Elliane', 'Vine Street',
'California', '789-123-4567'); END;

```

```
-- Restore the original time zone for your session.  
ALTER SESSION SET TIME_ZONE='+00:00';
```

Script Output	
Task completed in 0.006 seconds	
SESSIONTIMEZONE	CURR_DATE
+00:00	22-NOV-2012 08:04
session SET altered. anonymous block completed session SET altered.	

