

Oracle Database 11g: SQL Fundamentals II

Volume II • Student Guide

D49994GC20

Edition 2.0

September 2009

D62999

ORACLE®

Authors

Chaitanya Koratamaddi
Brian Pottle
Tulika Srivastava

Technical Contributors and Reviewers

Claire Bennett
Ken Cooper
Yanti Chang
Laszlo Czinkoczki
Burt Demchick
Gerlinde Frenzen
Joel Goodman
Laura Garza
Richard Green
Nancy Greenberg
Akira Kinutani
Wendy Lo
Isabelle Marchand
Timothy Mcglue
Alan Paulson
Srinivas Putrevu
Bryan Roberts
Clinton Shaffer
Abhishek Singh
Jenny Tsai Smith
James Spiller
Lori Tritz
Lex van der Werff
Marcie Young

Editors

Amitha Narayan
Daniel Milne

Graphic Designer

Satish Bettgowda

Publisher

Veena Narasimhan

Copyright © 2009, Oracle. All rights reserved.

Disclaimer

This course provides an overview of features and enhancements planned in release 11g. It is intended solely to help you assess the business benefits of upgrading to 11g and to plan your IT projects.

This course in any form, including its course labs and printed matter, contains proprietary information that is the exclusive property of Oracle. This course and the information contained herein may not be disclosed, copied, reproduced, or distributed to anyone outside Oracle without prior written consent of Oracle. This course and its contents are not part of your license agreement nor can they be incorporated into any contractual agreement with Oracle or its subsidiaries or affiliates.

This course is for informational purposes only and is intended solely to assist you in planning for the implementation and upgrade of the product features described. It is not a commitment to deliver any material, code, or functionality, and should not be relied upon in making purchasing decisions. The development, release, and timing of any features or functionality described in this document remain at the sole discretion of Oracle.

This document contains proprietary information and is protected by copyright and other intellectual property laws. You may copy and print this document solely for your own use in an Oracle training course. The document may not be modified or altered in any way. Except where your use constitutes "fair use" under copyright law, you may not use, share, download, upload, copy, print, display, perform, reproduce, publish, license, post, transmit, or distribute this document in whole or in part without the express authorization of Oracle.

The information contained in this document is subject to change without notice. If you find any problems in the document, please report them in writing to: Oracle University, 500 Oracle Parkway, Redwood Shores, California 94065 USA. This document is not warranted to be error-free.

Restricted Rights Notice

If this documentation is delivered to the United States Government or anyone using the documentation on behalf of the United States Government, the following notice is applicable:

U.S. GOVERNMENT RIGHTS

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

Trademark Notice

Oracle is a registered trademark of Oracle Corporation and/or its affiliates. Other names may be trademarks of their respective owners.

Contents

I Introduction

Lesson Objectives	I-2
Lesson Agenda	I-3
Course Objectives	I-4
Course Prerequisites	I-5
Course Agenda	I-6
Lesson Agenda	I-7
Tables Used in This Course	I-8
Appendixes Used in This Course	I-9
Development Environments	I-10
Lesson Agenda	I-11
Review of Restricting Data	I-12
Review of Sorting Data	I-13
Review of SQL Functions	I-14
Review of Single-Row Functions	I-15
Review of Types of Group Functions	I-16
Review of Using Subqueries	I-17
Review of Manipulating Data	I-18
Lesson Agenda	I-19
Oracle Database 11g SQL Documentation	I-20
Additional Resources	I-21
Summary	I-22
Practice I: Overview	I-23

1 Controlling User Access

Objectives	1-2
Lesson Agenda	1-3
Controlling User Access	1-4
Privileges	1-5
System Privileges	1-6
Creating Users	1-7
User System Privileges	1-8
Granting System Privileges	1-9
Lesson Agenda	1-10
What Is a Role?	1-11
Creating and Granting Privileges to a Role	1-12

Changing Your Password 1-13
 Lesson Agenda 1-14
 Object Privileges 1-15
 Granting Object Privileges 1-17
 Passing On Your Privileges 1-18
 Confirming Granted Privileges 1-19
 Lesson Agenda 1-20
 Revoking Object Privileges 1-21
 Quiz 1-23
 Summary 1-24
 Practice 1: Overview 1-25

2 Managing Schema Objects

Objectives 2-2
 Lesson Agenda 2-3
 ALTER TABLE Statement 2-4
 Adding a Column 2-6
 Modifying a Column 2-7
 Dropping a Column 2-8
 SET UNUSED Option 2-9
 Lesson Agenda 2-11
 Adding a Constraint Syntax 2-12
 Adding a Constraint 2-13
 ON DELETE Clause 2-14
 Deferring Constraints 2-15
 Difference Between INITIALLY DEFERRED and INITIALLY IMMEDIATE 2-16
 Dropping a Constraint 2-18
 Disabling Constraints 2-19
 Enabling Constraints 2-20
 Cascading Constraints 2-22
 Renaming Table Columns and Constraints 2-24
 Lesson Agenda 2-25
 Overview of Indexes 2-26
 CREATE INDEX with the CREATE TABLE Statement 2-27
 Function-Based Indexes 2-29
 Removing an Index 2-30
 DROP TABLE ... PURGE 2-31
 Lesson Agenda 2-32
 FLASHBACK TABLE Statement 2-33
 Using the FLASHBACK TABLE Statement 2-35

Lesson Agenda 2-36
 Temporary Tables 2-37
 Creating a Temporary Table 2-38
 Lesson Agenda 2-39
 External Tables 2-40
 Creating a Directory for the External Table 2-41
 Creating an External Table 2-43
 Creating an External Table by Using ORACLE_LOADER 2-45
 Querying External Tables 2-47
 Creating an External Table by Using ORACLE_DATAPUMP: Example 2-48
 Quiz 2-49
 Summary 2-51
 Practice 2: Overview 2-52

3 Managing Objects with Data Dictionary Views

Objectives 3-2
 Lesson Agenda 3-3
 Data Dictionary 3-4
 Data Dictionary Structure 3-5
 How to Use the Dictionary Views 3-7
 USER_OBJECTS and ALL_OBJECTS Views 3-8
 USER_OBJECTS View 3-9
 Lesson Agenda 3-10
 Table Information 3-11
 Column Information 3-12
 Constraint Information 3-14
 USER_CONSTRAINTS: Example 3-15
 Querying USER_CONS_COLUMNS 3-16
 Lesson Agenda 3-17
 View Information 3-18
 Sequence Information 3-19
 Confirming Sequences 3-20
 Index Information 3-21
 USER_INDEXES: Examples 3-22
 Querying USER_IND_COLUMNS 3-23
 Synonym Information 3-24
 Lesson Agenda 3-25
 Adding Comments to a Table 3-26

Quiz 3-27
Summary 3-28
Practice 3: Overview 3-29

4 Manipulating Large Data Sets

Objectives 4-2
Lesson Agenda 4-3
Using Subqueries to Manipulate Data 4-4
Retrieving Data by Using a Subquery as Source 4-5
Inserting by Using a Subquery as a Target 4-7
Using the `WITH CHECK OPTION` Keyword on DML Statements 4-9
Lesson Agenda 4-11
Overview of the Explicit Default Feature 4-12
Using Explicit Default Values 4-13
Copying Rows from Another Table 4-14
Lesson Agenda 4-15
Overview of Multitable `INSERT` Statements 4-16
Types of Multitable `INSERT` Statements 4-18
Multitable `INSERT` Statements 4-19
Unconditional `INSERT ALL` 4-21
Conditional `INSERT ALL`: Example 4-23
Conditional `INSERT ALL` 4-24
Conditional `INSERT FIRST`: Example 4-26
Conditional `INSERT FIRST` 4-27
Pivoting `INSERT` 4-29
Lesson Agenda 4-32
`MERGE` Statement 4-33
`MERGE` Statement Syntax 4-34
Merging Rows: Example 4-35
Lesson Agenda 4-38
Tracking Changes in Data 4-39
Example of the Flashback Version Query 4-40
`VERSIONS BETWEEN` Clause 4-42
Quiz 4-43
Summary 4-44
Practice 4: Overview 4-45

5 Managing Data in Different Time Zones

Objectives 5-2
Lesson Agenda 5-3

Time Zones	5-4
TIME_ZONE Session Parameter	5-5
CURRENT_DATE, CURRENT_TIMESTAMP, and LOCALTIMESTAMP	5-6
Comparing Date and Time in a Session's Time Zone	5-7
DBTIMEZONE and SESSIONTIMEZONE	5-9
TIMESTAMP Data Types	5-10
TIMESTAMP Fields	5-11
Difference Between DATE and TIMESTAMP	5-12
Comparing TIMESTAMP Data Types	5-13
Lesson Agenda	5-14
INTERVAL Data Types	5-15
INTERVAL Fields	5-17
INTERVAL YEAR TO MONTH: Example	5-18
INTERVAL DAY TO SECOND Data Type: Example	5-20
Lesson Agenda	5-21
EXTRACT	5-22
TZ_OFFSET	5-23
FROM_TZ	5-25
TO_TIMESTAMP	5-26
TO_YMINTERVAL	5-27
TO_DSINTERVAL	5-28
Daylight Saving Time	5-29
Quiz	5-31
Summary	5-32
Practice 5: Overview	5-33
6 Retrieving Data by Using Subqueries	
Objectives	6-2
Lesson Agenda	6-3
Multiple-Column Subqueries	6-4
Column Comparisons	6-5
Pairwise Comparison Subquery	6-6
Nonpairwise Comparison Subquery	6-8
Lesson Agenda	6-10
Scalar Subquery Expressions	6-11
Scalar Subqueries: Examples	6-12
Lesson Agenda	6-14
Correlated Subqueries	6-15
Using Correlated Subqueries	6-17
Lesson Agenda	6-19

Using the `EXISTS` Operator 6-20
 Find All Departments That Do Not Have Any Employees 6-22
 Correlated `UPDATE` 6-23
 Using Correlated `UPDATE` 6-24
 Correlated `DELETE` 6-26
 Using Correlated `DELETE` 6-27
 Lesson Agenda 6-28
`WITH` Clause 6-29
`WITH` Clause: Example 6-30
 Recursive `WITH` Clause: Example 6-33
 Quiz 6-34
 Summary 6-35
 Practice 6: Overview 6-37

7 Regular Expression Support

Objectives 7-2
 Lesson Agenda 7-3
 What Are Regular Expressions? 7-4
 Benefits of Using Regular Expressions 7-5
 Using the Regular Expressions Functions and Conditions in SQL and PL/SQL 7-6
 Lesson Agenda 7-7
 What Are Metacharacters? 7-8
 Using Metacharacters with Regular Expressions 7-9
 Lesson Agenda 7-11
 Regular Expressions Functions and Conditions: Syntax 7-12
 Performing a Basic Search by Using the `REGEXP_LIKE` Condition 7-13
 Replacing Patterns by Using the `REGEXP_REPLACE` Function 7-14
 Finding Patterns by Using the `REGEXP_INSTR` Function 7-15
 Extracting Substrings by Using the `REGEXP_SUBSTR` Function 7-16
 Lesson Agenda 7-17
 Subexpressions 7-18
 Using Subexpressions with Regular Expression Support 7-19
 Why Access the *n*th Subexpression? 7-20
`REGEXP_SUBSTR`: Example 7-21
 Lesson Agenda 7-22
 Using the `REGEXP_COUNT` Function 7-23
 Regular Expressions and Check Constraints: Examples 7-24
 Quiz 7-25
 Summary 7-26
 Practice 7: Overview 7-27

Appendix A: Practice Solutions

Appendix B: Table Descriptions

Appendix C: Using SQL Developer

Objectives	C-2
What Is Oracle SQL Developer?	C-3
Specifications of SQL Developer	C-4
SQL Developer 1.5 Interface	C-5
Creating a Database Connection	C-7
Browsing Database Objects	C-10
Displaying the Table Structure	C-11
Browsing Files	C-12
Creating a Schema Object	C-13
Creating a New Table: Example	C-14
Using the SQL Worksheet	C-15
Executing SQL Statements	C-18
Saving SQL Scripts	C-19
Executing Saved Script Files: Method 1	C-20
Executing Saved Script Files: Method 2	C-21
Formatting the SQL Code	C-22
Using Snippets	C-23
Using Snippets: Example	C-24
Debugging Procedures and Functions	C-25
Database Reporting	C-26
Creating a User-Defined Report	C-27
Search Engines and External Tools	C-28
Setting Preferences	C-29
Resetting the SQL Developer Layout	C-30
Summary	C-31

Appendix D: Using SQL*Plus

Objectives	D-2
SQL and SQL*Plus Interaction	D-3
SQL Statements Versus SQL*Plus Commands	D-4
Overview of SQL*Plus	D-5
Logging In to SQL*Plus	D-6
Displaying the Table Structure	D-7
SQL*Plus Editing Commands	D-9
Using LIST, n, and APPEND	D-11

Using the `CHANGE` Command D-12
SQL*Plus File Commands D-13
Using the `SAVE` and `START` Commands D-14
`SERVEROUTPUT` Command D-15
Using the SQL*Plus `SPOOL` Command D-16
Using the `AUTOTRACE` Command D-17
Summary D-18

Appendix E: Using JDeveloper

Objectives E-2
Oracle JDeveloper E-3
Database Navigator E-4
Creating a Connection E-5
Browsing Database Objects E-6
Executing SQL Statements E-7
Creating Program Units E-8
Compiling E-9
Running a Program Unit E-10
Dropping a Program Unit E-11
Structure Window E-12
Editor Window E-13
Application Navigator E-14
Deploying Java Stored Procedures E-15
Publishing Java to PL/SQL E-16
How Can I Learn More About JDeveloper 11g ?
Summary E-18

Appendix F: Generating Reports by Grouping Related Data

Objectives F-2
Review of Group Functions F-3
Review of the `GROUP BY` Clause F-4
Review of the `HAVING` Clause F-5
`GROUP BY` with `ROLLUP` and `CUBE` Operators F-6
`ROLLUP` Operator F-7
`ROLLUP` Operator: Example F-8
`CUBE` Operator F-9
`CUBE` Operator: Example F-10
`GROUPING` Function F-11
`GROUPING` Function: Example F-12
`GROUPING SETS` F-13

GROUPING SETS: Example F-15
 Composite Columns F-17
 Composite Columns: Example F-19
 Concatenated Groupings F-21
 Concatenated Groupings: Example F-22
 Summary F-23

Appendix G: Hierarchical Retrieval

Objectives G-2
 Sample Data from the EMPLOYEES Table G-3
 Natural Tree Structure G-4
 Hierarchical Queries G-5
 Walking the Tree G-6
 Walking the Tree: From the Bottom Up G-8
 Walking the Tree: From the Top Down G-9
 Ranking Rows with the LEVEL Pseudocolumn G-10
 Formatting Hierarchical Reports Using LEVEL and LPAD G-11
 Pruning Branches G-13
 Summary G-14

Appendix H: Writing Advanced Scripts

Objectives H-2
 Using SQL to Generate SQL H-3
 Creating a Basic Script H-4
 Controlling the Environment H-5
 The Complete Picture H-6
 Dumping the Contents of a Table to a File H-7
 Generating a Dynamic Predicate H-9
 Summary H-11

Appendix I: Oracle Database Architectural Components

Objectives I-2
 Oracle Database Architecture: Overview I-3
 Oracle Database Server Structures I-4
 Connecting to the Database I-5
 Interacting with an Oracle Database I-6
 Oracle Memory Architecture I-8
 Process Architecture I-10
 Database Writer Process I-12
 Log Writer Process I-13
 Checkpoint Process I-14

System Monitor Process	I-15
Process Monitor Process	I-16
Oracle Database Storage Architecture	I-17
Logical and Physical Database Structures	I-19
Processing a SQL Statement	I-21
Processing a Query	I-22
Shared Pool	I-23
Database Buffer Cache	I-25
Program Global Area (PGA)	I-26
Processing a DML Statement	I-27
Redo Log Buffer	I-29
Rollback Segment	I-30
COMMIT Processing	I-31
Summary of the Oracle Database Architecture	I-33

Additional Practices and Solutions

Appendix A

Practices and Solutions

Table of Contents

Practices and Solutions for Lesson I	3
Practice I-1: Accessing SQL Developer Resources	4
Practice I-2: Using SQL Developer	5
Practice Solutions I-1: Accessing SQL Developer Resources	7
Practice Solutions I-2: Using SQL Developer	8
Practices and Solutions for Lesson 1	17
Practice 1-1: Controlling User Access	17
Practice Solutions 1-1: Controlling User Access	20
Practices and Solutions for Lesson 2	25
Practice 2-1: Managing Schema Objects	25
Practice Solutions 2-1: Managing Schema Objects	31
Practices and Solutions for Lesson 3	39
Practice 3-1: Managing Objects with Data Dictionary Views	39
Practice Solutions 3-1: Managing Objects with Data Dictionary Views	43
Practices and Solutions for Lesson 4	47
Practice 4-1: Manipulating Large Data Sets	47
Practice Solutions 4-1: Manipulating Large Data Sets	51
Practices and Solutions for Lesson 5	56
Practice 5-1: Managing Data in Different Time Zones	56
Practice Solutions 5-1: Managing Data in Different Time Zones	59
Practices and Solutions for Lesson 6	62
Practice 6-1: Retrieving Data by Using Subqueries	62
Practice Solutions 6-1: Retrieving Data by Using Subqueries	66
Practices and Solutions for Lesson 7	70
Practice 7-1: Regular Expression Support	70
Practice Solutions 7-1: Regular Expression Support	72

Practices and Solutions for Lesson I

In this practice, you review the available SQL Developer resources. You also learn about your user account that you use in this course. You then start SQL Developer, create a new database connection, and browse your HR tables. You also set some SQL Developer preferences, execute SQL statements, and execute an anonymous PL/SQL block by using SQL Worksheet. Finally, you access and bookmark the Oracle Database 11g documentation and other useful Web sites that you can use in this course.

Practice I-1: Accessing SQL Developer Resources

In this practice, you do the following:

- 1) Access the SQL Developer home page.
 - a. Access the online SQL Developer home page available at:
http://www.oracle.com/technology/products/database/sql_developer/index.html
 - b. Bookmark the page for easier future access.
- 2) Access the SQL Developer tutorial available online at:
<http://st-curriculum.oracle.com/tutorial/SQLDeveloper/index.htm>. Then review the following sections and associated demos:
 - a) What to Do First
 - b) Working with Database Objects
 - c) Accessing Data

Practice I-2: Using SQL Developer

- 1) Start SQL Developer by using the desktop icon.
- 2) Create a database connection using the following information:
 - a) Connection Name: myconnection
 - b) Username: oraxx, where xx is the number of your PC (Ask your instructor to assign you an ora account out of the ora21-ora40 range of accounts.)
 - c) Password: oraxx
 - d) Hostname: localhost
 - e) Port: 1521
 - f) SID: orcl (or the value provided to you by the instructor)
- 3) Test the new connection. If the status is Success, connect to the database by using this new connection.
 - a) Click the Test button in the New/Select Database Connection window.
 - b) If the status is Success, click the Connect button.
- 4) Browse the structure of the EMPLOYEES table and display its data.
 - a) Expand the myconnection connection by clicking the plus sign next to it.
 - b) Expand the Tables icon by clicking the plus sign next to it.
 - c) Display the structure of the EMPLOYEES table.
 - d) View the data of the DEPARTMENTS table.
- 5) Execute some basic SELECT statements to query the data in the EMPLOYEES table in the SQL Worksheet area. Use both the Execute Statement (or press F9) and the Run Script (or press F5) icons to execute the SELECT statements. Review the results of both methods of executing the SELECT statements on the appropriate tabbed pages.
 - a) Write a query to select the last name and salary for any employee whose salary is less than or equal to \$3,000.
 - b) Write a query to display last name, job ID, and commission for all employees who are not entitled to receive a commission.
- 6) Set your script pathing preference to /home/oracle/labs/sql2.
 - a) Select Tools > Preferences > Database > Worksheet Parameters.
 - b) Enter the value in the Select default path to look for scripts field.
- 7) Enter the following in the Enter SQL Statement box.


```
SELECT employee_id, first_name, last_name,
       FROM employees;
```
- 8) Save the SQL statement to a script file by using the File > Save As menu item.
 - a) Select File > Save As.
 - b) Name the file intro_test.sql.

Practice I-2: Using SQL Developer (continued)

- c) Place the file under your /home/oracle/labs/sql2/labs folder.
- 9) Open and run `confidence.sql` from your /home/oracle/labs/sql2/labs folder, and observe the output.

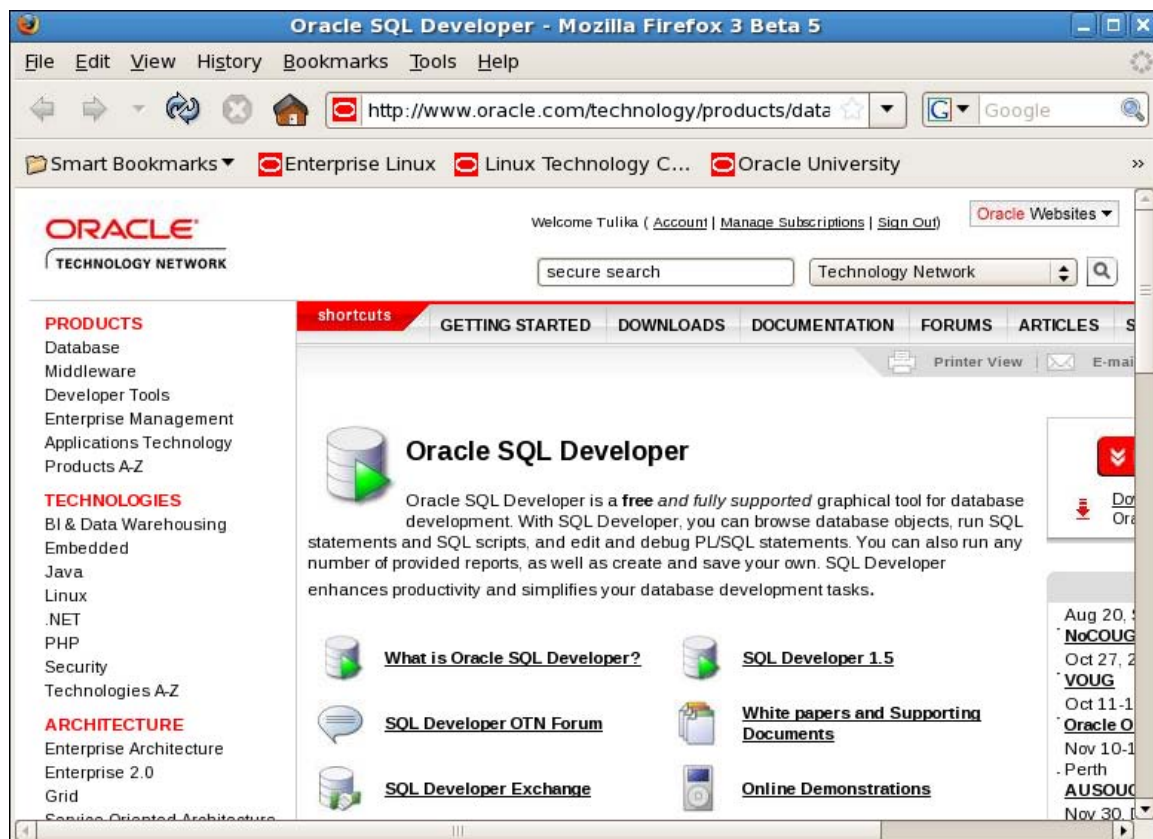
Practice Solutions I-1: Accessing SQL Developer Resources

1) Access the SQL Developer home page.

a) Access the online SQL Developer home page available online at:

http://www.oracle.com/technology/products/database/sql_developer/index.html

The SQL Developer home page is displayed as follows:



b) Bookmark the page for easier future access.

2) Access the SQL Developer tutorial available online at:

<http://st-curriculum.oracle.com/tutorial/SQLDeveloper/index.htm>

Then, review the following sections and associated demos:

a) What to Do First

b) Working with Database Objects

c) Accessing Data

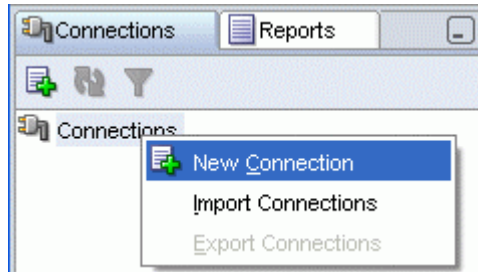
Practice Solutions I-2: Using SQL Developer

1) Start SQL Developer by using the desktop icon.



2) Create a database connection using the following information:

- a. Connection Name: myconnection
- b. Username: oraxx (Ask your instructor to assign you one ora account out of the ora21–ora40 range of accounts.)
- c. Password: oraxx
- d. Hostname: localhost
- e. Port: 1521
- f. SID: orcl (or the value provided to you by the instructor)



Practice Solutions I-2: Using SQL Developer (continued)

Connection Name: myconnection
Username: ora21
Password: *****
☒ Save Password
Oracle
Role: default
Connection Type: Basic
☐ OS Authentication
☐ Kerberos Authentication
☐ Proxy Connection
Hostname: localhost
Port: 1521
☒ SID: orcl
☐ Service name:
Status :
Buttons: Help, Save, Clear, Test, Connect, Cancel

- 3) Test the new connection. If the status is Success, connect to the database by using this new connection.

a) Click the Test button in the New/Select Database Connection window.

Status :
Buttons: Help, Save, Clear, Test, Connect, Cancel

b) If the status is Success, click the Connect button.

Status : Success
Buttons: Help, Save, Clear, Test, Connect, Cancel

Browsing the Tables

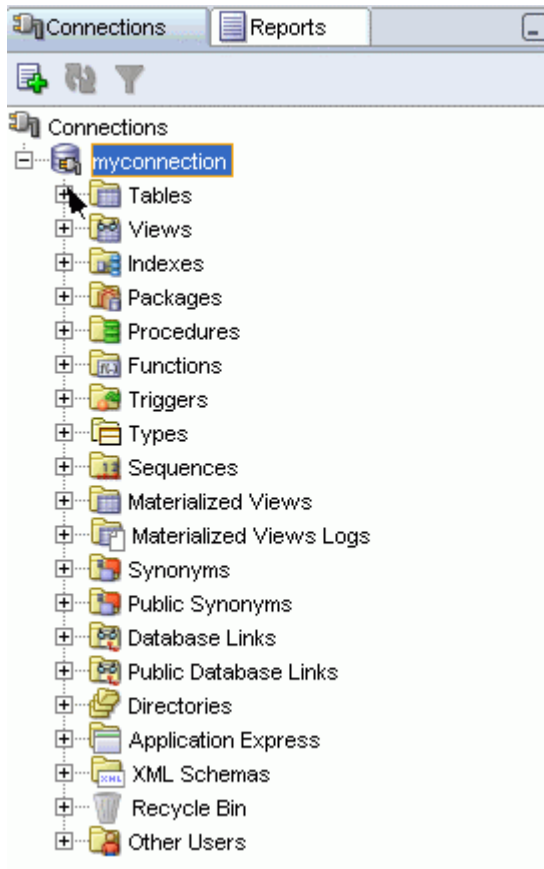
- 4) Browse the structure of the EMPLOYEES table and display its data.

a) Expand the myconnection connection by clicking the plus sign next to it.

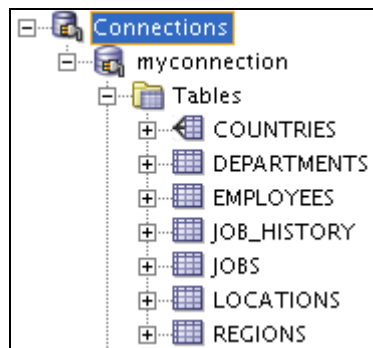
Practice Solutions I-2: Using SQL Developer (continued)



b) Expand the Tables icon by clicking the plus sign next to it.



Practice Solutions I-2: Using SQL Developer (continued)



- c) Display the structure of the EMPLOYEES table.

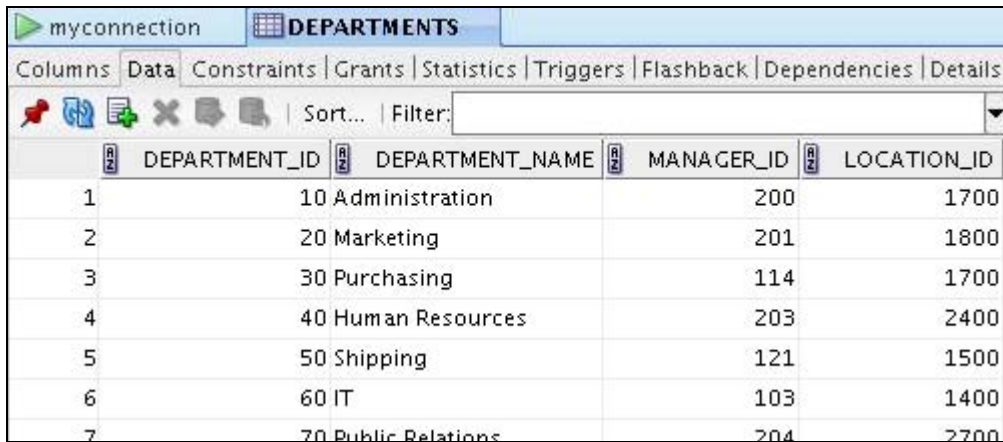
Click the **EMPLOYEES** table. The Columns tab displays the columns in the EMPLOYEES table as follows:

myconnection EMPLOYEES									
Columns Data Constraints Grants Statistics Triggers Flashback Dependencies Details Indexes SQL									
Actions...									
Column Name	Data Type	Nullable	Data Default	COLUMN ID	Primary Key	COMMENTS			
EMPLOYEE_ID	NUMBER(6,0)	No	(null)	1	1	Primary key of employee			
FIRST_NAME	VARCHAR2(20 BYTE)	Yes	(null)	2	(null)	First name of the employee			
LAST_NAME	VARCHAR2(25 BYTE)	No	(null)	3	(null)	Last name of the employee			
EMAIL	VARCHAR2(25 BYTE)	No	(null)	4	(null)	Email id of the employee			
PHONE_NUMBER	VARCHAR2(20 BYTE)	Yes	(null)	5	(null)	Phone number of the employee			
HIRE_DATE	DATE	No	(null)	6	(null)	Date when the employee was hired			
JOB_ID	VARCHAR2(10 BYTE)	No	(null)	7	(null)	Current job of the employee			
SALARY	NUMBER(8,2)	Yes	(null)	8	(null)	Monthly salary of the employee			
COMMISSION_PCT	NUMBER(2,2)	Yes	(null)	9	(null)	Commission percentage of the employee			
MANAGER_ID	NUMBER(6,0)	Yes	(null)	10	(null)	Manager id of the employee			
DEPARTMENT_ID	NUMBER(4,0)	Yes	(null)	11	(null)	Department id where employee works			

- d) View the data of the DEPARTMENTS table.

In the Connections navigator, click the **DEPARTMENTS** table. Then click the Data tab.

Practice Solutions I-2: Using SQL Developer (continued)



The screenshot shows the Oracle SQL Developer interface with the 'DEPARTMENTS' table selected. The 'Data' tab is active, displaying a grid of data. The columns are DEPARTMENT_ID, DEPARTMENT_NAME, MANAGER_ID, and LOCATION_ID. The data is as follows:

	DEPARTMENT_ID	DEPARTMENT_NAME	MANAGER_ID	LOCATION_ID
1	10	Administration	200	1700
2	20	Marketing	201	1800
3	30	Purchasing	114	1700
4	40	Human Resources	203	2400
5	50	Shipping	121	1500
6	60	IT	103	1400
7	70	Public Relations	204	2700

- 5) Execute some basic **SELECT** statements to query the data in the **EMPLOYEES** table in the SQL Worksheet area. Use both the Execute Statement (or press F9) and the Run Script icons (or press F5) to execute the **SELECT** statements. Review the results of both methods of executing the **SELECT** statements on the appropriate tabbed pages.

- a) Write a query to select the last name and salary for any employee whose salary is less than or equal to \$3,000.

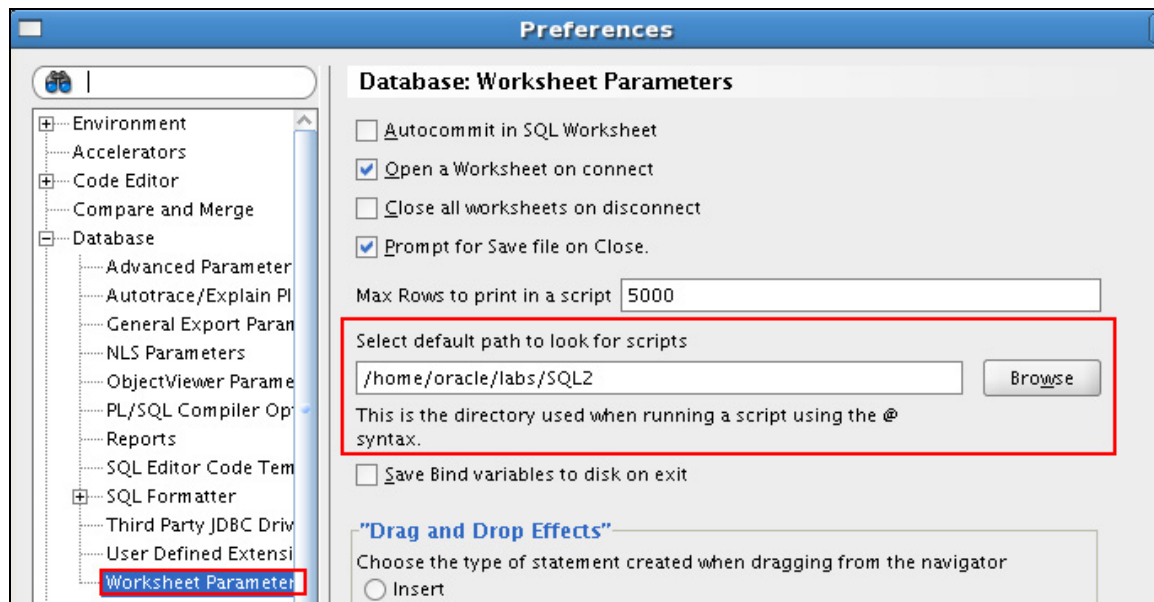
```
SELECT last_name, salary
FROM employees
WHERE salary <= 3000;
```

- b) Write a query to display last name, job ID, and commission for all employees who are not entitled to receive a commission.

```
SELECT last_name, job_id, commission_pct
FROM employees
WHERE commission_pct IS NULL;
```

- 6) Set your script pathing preference to `/home/oracle/labs/sql2`.
- a) Select **Tools > Preferences > Database > Worksheet Parameters**.
- b) Enter the value in the **Select default path to look for scripts** field. Then, click **OK**.

Practice Solutions I-2: Using SQL Developer (continued)

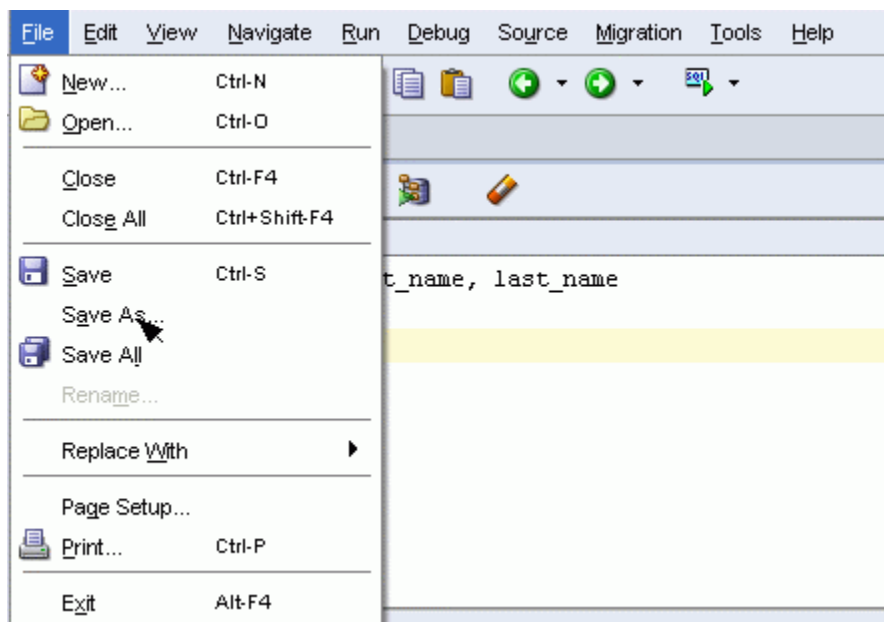


7) Enter the following SQL statement:

```
SELECT employee_id, first_name, last_name  
FROM employees;
```

8) Save the SQL statement to a script file by using the File > Save As menu item.

a) Select File > Save As.

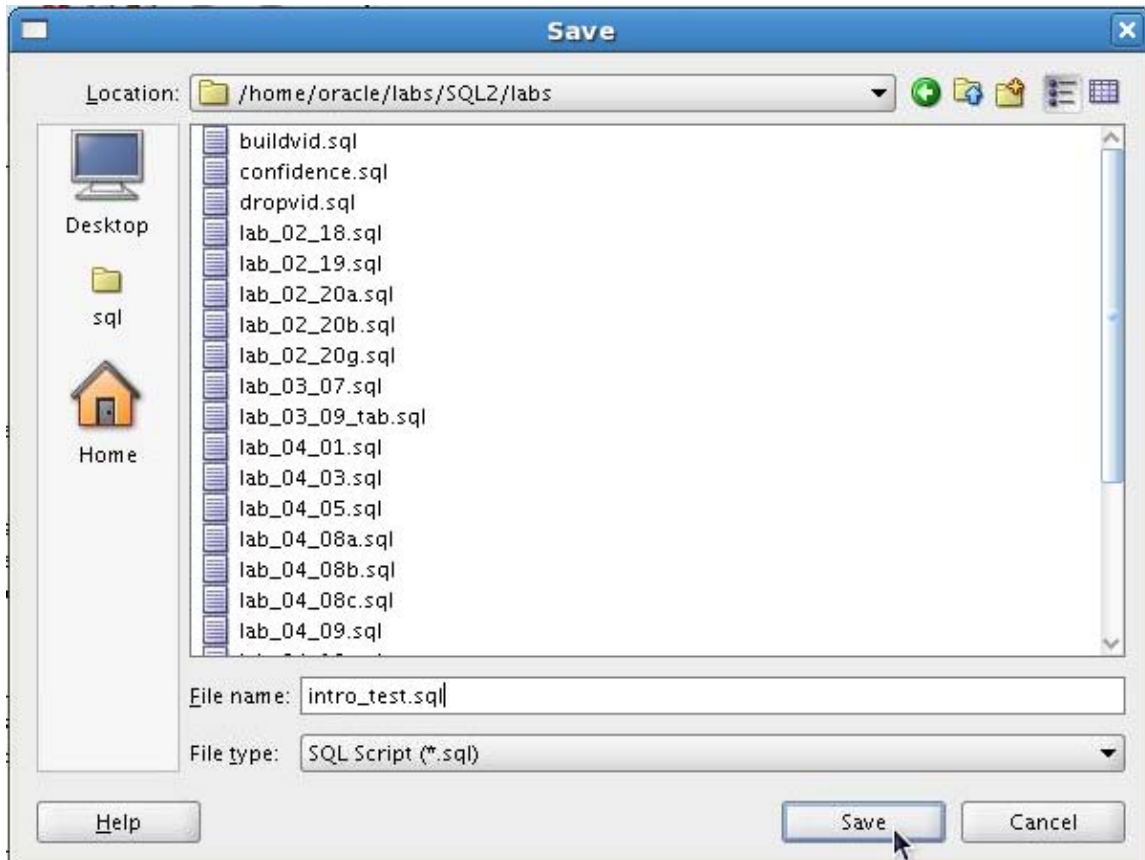


b) Name the file **intro_test.sql**.

Practice Solutions I-2: Using SQL Developer (continued)

Enter `intro_test.sql` in the File_name text box.

c) Place the file under the `/home/oracle/labs/SQL2/labs` folder.

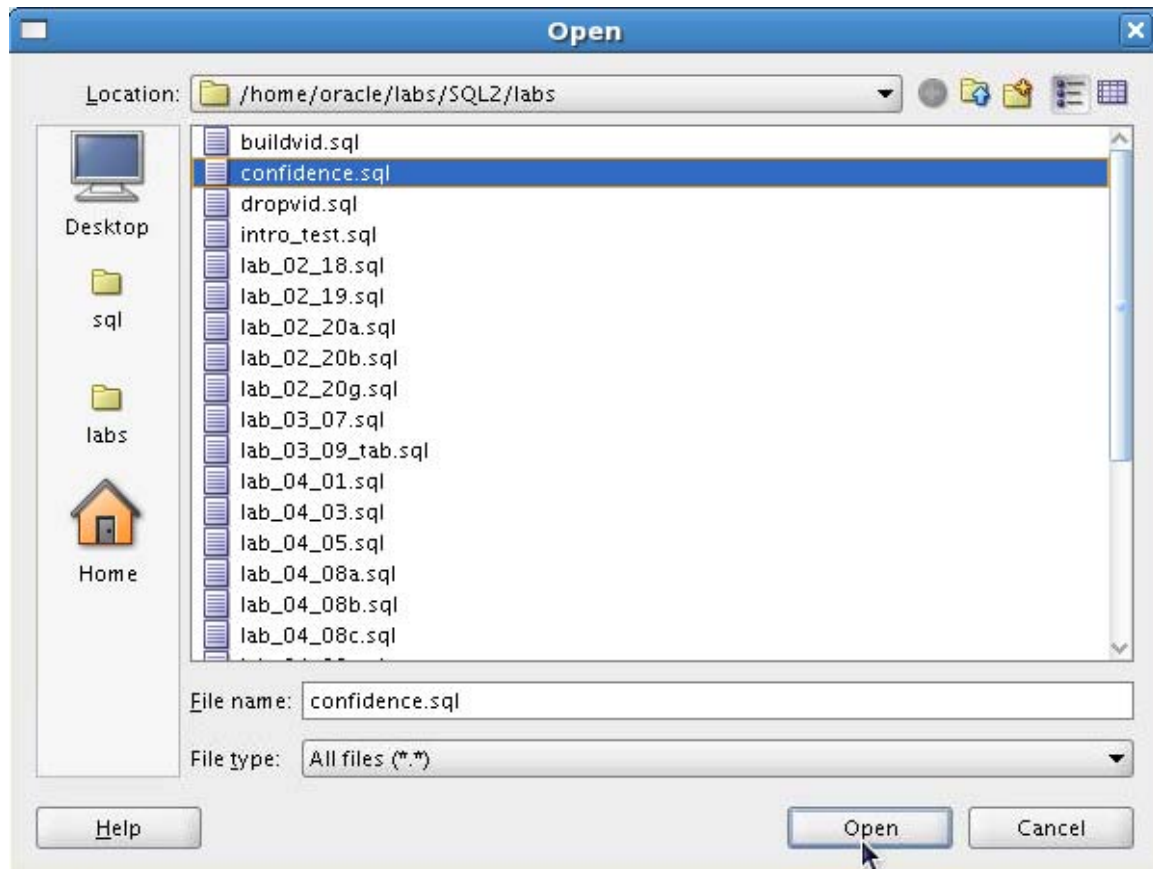


Then, click Save.

9) Open and run `confidence.sql` from your `/home/oracle/labs/SQL2/labs` folder and observe the output.

Practice Solutions I-2: Using SQL Developer (continued)

Open the confidence.sql script file by using the File > Open menu item.



Then, press F5 to execute the script.

The following is the expected result:

```
COUNT (*)
-----
8

1 rows selected

COUNT (*)
-----
107

1 rows selected

COUNT (*)
-----
25

1 rows selected
```

Practice Solutions I-2: Using SQL Developer (continued)

```
COUNT (*)
-----
4

1 rows selected

COUNT (*)
-----
23

1 rows selected

COUNT (*)
-----
27

1 rows selected

COUNT (*)
-----
19

1 rows selected

COUNT (*)
-----
10

1 rows selected
```

Practices and Solutions for Lesson 1

Practice 1-1: Controlling User Access

1. What privilege should a user be given to log on to the Oracle server? Is this a system privilege or an object privilege?

2. What privilege should a user be given to create tables?

3. If you create a table, who can pass along privileges to other users in your table?

4. You are the DBA. You create many users who require the same system privileges. What should you use to make your job easier?

5. What command do you use to change your password?

6. User21 is the owner of the EMP table and grants the DELETE privilege to User22 by using the WITH GRANT OPTION clause. User22 then grants the DELETE privilege on EMP to User23. User21 now finds that User23 has the privilege and revokes it from User22. Which user can now delete from the EMP table?

7. You want to grant SCOTT the privilege to update data in the DEPARTMENTS table. You also want to enable SCOTT to grant this privilege to other users. What command do you use?

To complete question 8 and the subsequent ones, you need to connect to the database by using SQL Developer. If you are already not connected, do the following to connect:

1. Click the SQL Developer desktop icon.
 2. In the Connections Navigator, use the **oraxx** account and the corresponding password provided by your instructor to log on to the database.
-
8. Grant another user query privilege on your table. Then, verify whether that user can use the privilege.
Note: For this exercise, team up with another group. For example, if you are user ora21, team up with another user ora22.
 - a. Grant another user privilege to view records in your REGIONS table. Include an option for this user to further grant this privilege to other users.
 - b. Have the user query your REGIONS table.
 - c. Have the user pass on the query privilege to a third user (for example, ora23).

Practice 1-1: Controlling User Access (continued)

d. Take back the privilege from the user who performs step b.

Note: Each team can run exercises 9 and 10 independently.

9. Grant another user query and data manipulation privileges on your COUNTRIES table. Make sure that the user cannot pass on these privileges to other users.

10. Take back the privileges on the COUNTRIES table granted to another user.

Note: For exercises 11 through 17, team up with another group.

11. Grant another user access to your DEPARTMENTS table. Have the user grant you query access to his or her DEPARTMENTS table.

12. Query all the rows in your DEPARTMENTS table.

	DEPARTMENT_ID	DEPARTMENT_NAME	MANAGER_ID	LOCATION_ID
1	10	Administration	200	1700
2	20	Marketing	201	1800
3	30	Purchasing	114	1700
4	40	Human Resources	203	2400
5	50	Shipping	121	1500
6	60	IT	103	1400
7	70	Public Relations	204	2700
8	80	Sales	145	2500

...

13. Add a new row to your DEPARTMENTS table. Team 1 should add Education as department number 500. Team 2 should add Human Resources as department number 510. Query the other team's table.

14. Create a synonym for the other team's DEPARTMENTS table.

15. Query all the rows in the other team's DEPARTMENTS table by using your synonym.

Team 1 SELECT statement results:

	DEPARTMENT_ID	DEPARTMENT_NAME	MANAGER_ID	LOCATION_ID
15	150	Shareholder Services	(null)	1700
16	160	Benefits	(null)	1700
17	170	Manufacturing	(null)	1700
18	180	Construction	(null)	1700
19	190	Contracting	(null)	1700
20	200	Operations	(null)	1700
21	210	IT Support	(null)	1700
22	220	NOC	(null)	1700
23	230	IT Helpdesk	(null)	1700
24	240	Government Sales	(null)	1700
25	250	Retail Sales	(null)	1700
26	260	Recruiting	(null)	1700
27	270	Payroll	(null)	1700
28	510	Human Resources	(null)	(null)

Practice 1-1: Controlling User Access (continued)

Team 2 SELECT statement results:

	DEPARTMENT_ID	DEPARTMENT_NAME	MANAGER_ID	LOCATION_ID
15	150	Shareholder Services	(null)	1700
16	160	Benefits	(null)	1700
17	170	Manufacturing	(null)	1700
18	180	Construction	(null)	1700
19	190	Contracting	(null)	1700
20	200	Operations	(null)	1700
21	210	IT Support	(null)	1700
22	220	NOC	(null)	1700
23	230	IT Helpdesk	(null)	1700
24	240	Government Sales	(null)	1700
25	250	Retail Sales	(null)	1700
26	260	Recruiting	(null)	1700
27	270	Payroll	(null)	1700
28	500	Education	(null)	(null)

16. Revoke the SELECT privilege from the other team.

17. Remove the row that you inserted into the DEPARTMENTS table in step 13 and save the changes.

Practice Solutions 1-1: Controlling User Access

To complete question 8 and the subsequent ones, you need to connect to the database by using SQL Developer.

1. What privilege should a user be given to log on to the Oracle server? Is this a system or an object privilege?
The CREATE SESSION system privilege
2. What privilege should a user be given to create tables?
The CREATE TABLE privilege
3. If you create a table, who can pass along privileges to other users in your table?
You can, or anyone you have given those privileges to, by using WITH GRANT OPTION
4. You are the DBA. You create many users who require the same system privileges.
What should you use to make your job easier?
Create a role containing the system privileges and grant the role to the users.
5. What command do you use to change your password?
The ALTER USER statement
6. User21 is the owner of the EMP table and grants DELETE privileges to User22 by using the WITH GRANT OPTION clause. User22 then grants DELETE privileges on EMP to User23. User21 now finds that User23 has the privilege and revokes it from User22. Which user can now delete data from the EMP table?
Only User21
7. You want to grant SCOTT the privilege to update data in the DEPARTMENTS table. You also want to enable SCOTT to grant this privilege to other users. What command do you use?
GRANT UPDATE ON departments TO scott WITH GRANT OPTION;

Practice Solutions 1-1: Controlling User Access (continued)

8. Grant another user query privilege on your table. Then, verify whether that user can use the privilege.

Note: For this exercise, team up with another group. For example, if you are user ora21, team up with another user ora22.

- a) Grant another user privilege to view records in your REGIONS table. Include an option for this user to further grant this privilege to other users.

Team 1 executes this statement:

```
GRANT select
ON regions
TO <team2_oraxx> WITH GRANT OPTION;
```

- b) Have the user query your REGIONS table.

Team 2 executes this statement:

```
SELECT * FROM <team1_oraxx>.regions;
```

- c) Have the user pass on the query privilege to a third user (for example, ora23).

Team 2 executes this statement.

```
GRANT select
ON <team1_oraxx>.regions
TO <team3_oraxx>;
```

- d) Take back the privilege from the user who performs step b.

Team 1 executes this statement.

```
REVOKE select
ON regions
FROM <team2_oraxx>;
```

9. Grant another user query and data manipulation privileges on your COUNTRIES table. Make sure the user cannot pass on these privileges to other users.

Team 1 executes this statement.

```
GRANT select, update, insert
ON COUNTRIES
TO <team2_oraxx>;
```

Practice Solutions 1-1: Controlling User Access (continued)

10. Take back the privileges on the COUNTRIES table granted to another user.

Team 1 executes this statement.

```
REVOKE select, update, insert ON COUNTRIES FROM <team2_oraxx>;
```

Note: For the exercises 11 through 17, team up with another group.

11. Grant another user access to your DEPARTMENTS table. Have the user grant you query access to his or her DEPARTMENTS table.

Team 2 executes the GRANT statement.

```
GRANT select  
ON departments  
TO <team1_oraxx>;
```

Team 1 executes the GRANT statement.

```
GRANT select  
ON departments  
TO <team2_oraxx>;
```

Here, <team1_oraxx> is the username of Team 1 and <team2_oraxx> is the username of Team 2.

12. Query all the rows in your DEPARTMENTS table.

```
SELECT  *  
FROM    departments;
```

13. Add a new row to your DEPARTMENTS table. Team 1 should add Education as department number 500. Team 2 should add Human Resources as department number 510. Query the other team's table.

Team 1 executes this INSERT statement.

```
INSERT INTO departments(department_id, department_name)  
VALUES (500, 'Education');  
COMMIT;
```

Team 2 executes this INSERT statement.

```
INSERT INTO departments(department_id, department_name)  
VALUES (510, 'Human Resources');  
COMMIT;
```

Practice Solutions 1-1: Controlling User Access (continued)

14. Create a synonym for the other team's DEPARTMENTS table.

Team 1 creates a synonym named team2.

```
CREATE SYNONYM team2
      FOR <team2_oraxx>.DEPARTMENTS;
```

Team 2 creates a synonym named team1.

```
CREATE SYNONYM team1
      FOR <team1_oraxx>. DEPARTMENTS;
```

15. Query all the rows in the other team's DEPARTMENTS table by using your synonym.

Team 1 executes this SELECT statement.

```
SELECT *
      FROM team2;
```

Team 2 executes this SELECT statement.

```
SELECT *
      FROM team1;
```

16. Revoke the SELECT privilege from the other team.

Team 1 revokes the privilege.

```
REVOKE select
      ON departments
      FROM <team2_oraxx>;
```

Team 2 revokes the privilege.

```
REVOKE select
      ON departments
      FROM <team1_oraxx>;
```

Practice Solutions 1-1: Controlling User Access (continued)

17. Remove the row that you inserted into the DEPARTMENTS table in step 8 and save the changes.

Team 1 executes this DELETE statement.

```
DELETE FROM departments  
WHERE department_id = 500;  
COMMIT;
```

Team 2 executes this DELETE statement.

```
DELETE FROM departments  
WHERE department_id = 510;  
COMMIT;
```

Practices and Solutions for Lesson 2

Practice 2-1: Managing Schema Objects

In this practice, you use the `ALTER TABLE` command to modify columns and add constraints. You use the `CREATE INDEX` command to create indexes when creating a table, along with the `CREATE TABLE` command. You create external tables.

1. Create the `DEPT2` table based on the following table instance chart. Enter the syntax in the SQL Worksheet. Then, execute the statement to create the table. Confirm that the table is created.

Column Name	ID	NAME
Key Type		
Nulls/Unique		
FK Table		
FK Column		
Data type	NUMBER	VARCHAR2
Length	7	25

Name	Null	Type
-----	-----	-----
ID		NUMBER (7)
NAME		VARCHAR2 (25)
2 rows selected		

2. Populate the `DEPT2` table with data from the `DEPARTMENTS` table. Include only the columns that you need.
3. Create the `EMP2` table based on the following table instance chart. Enter the syntax in the SQL Worksheet. Then execute the statement to create the table. Confirm that the table is created.

Column Name	ID	LAST_NAME	FIRST_NAME	DEPT_ID
Key Type				
Nulls/Unique				
FK Table				
FK Column				
Data type	NUMBER	VARCHAR2	VARCHAR2	NUMBER
Length	7	25	25	7

Practice 2-1: Managing Schema Objects (continued)

Name	Null	Type

ID		NUMBER(7)
LAST_NAME		VARCHAR2(25)
FIRST_NAME		VARCHAR2(25)
DEPT_ID		NUMBER(7)
4 rows selected		

4. Modify the EMP2 table to allow for longer employee last names. Confirm your modification.

Name	Null	Type

ID		NUMBER(7)
LAST_NAME		VARCHAR2(50)
FIRST_NAME		VARCHAR2(25)
DEPT_ID		NUMBER(7)
4 rows selected		

5. Create the EMPLOYEES2 table based on the structure of the EMPLOYEES table. Include only the EMPLOYEE_ID, FIRST_NAME, LAST_NAME, SALARY, and DEPARTMENT_ID columns. Name the columns in your new table ID, FIRST_NAME, LAST_NAME, SALARY, and DEPT_ID, respectively.
6. Drop the EMP2 table.
7. Query the recycle bin to see whether the table is present.

	ORIGINAL_NAME	OPERATION	DROPTIME

17	EMP_NEW_SAL	DROP	2009-05-22:14:44:15
18	EMP2	DROP	2009-05-22:14:57:57

8. Restore the EMP2 table to a state before the DROP statement.

Name	Null	Type

ID		NUMBER(7)
LAST_NAME		VARCHAR2(50)
FIRST_NAME		VARCHAR2(25)
DEPT_ID		NUMBER(7)
4 rows selected		

9. Drop the FIRST_NAME column from the EMPLOYEES2 table. Confirm your modification by checking the description of the table.

Practice 2-1: Managing Schema Objects (continued)

Name	Null	Type
ID		NUMBER(6)
LAST_NAME	NOT NULL	VARCHAR2(25)
SALARY		NUMBER(8,2)
DEPT_ID		NUMBER(4)
4 rows selected		

10. In the EMPLOYEES2 table, mark the DEPT_ID column as UNUSED. Confirm your modification by checking the description of the table.

Name	Null	Type
ID		NUMBER(6)
LAST_NAME	NOT NULL	VARCHAR2(25)
SALARY		NUMBER(8,2)
3 rows selected		

11. Drop all the UNUSED columns from the EMPLOYEES2 table. Confirm your modification by checking the description of the table.
12. Add a table-level PRIMARY KEY constraint to the EMP2 table on the ID column. The constraint should be named at creation. Name the constraint my_emp_id_pk.
13. Create a PRIMARY KEY constraint to the DEPT2 table using the ID column. The constraint should be named at creation. Name the constraint my_dept_id_pk.
14. Add a foreign key reference on the EMP2 table that ensures that the employee is not assigned to a nonexistent department. Name the constraint my_emp_dept_id_fk.
15. Modify the EMP2 table. Add a COMMISSION column of the NUMBER data type, precision 2, scale 2. Add a constraint to the COMMISSION column that ensures that a commission value is greater than zero.
16. Drop the EMP2 and DEPT2 tables so that they cannot be restored. Verify the recycle bin.
17. Create the DEPT_NAMED_INDEX table based on the following table instance chart. Name the index for the PRIMARY KEY column as DEPT_PK_IDX.

Column Name	Deptno	Dname
Primary Key	Yes	
Data Type	Number	VARCHAR2
Length	4	30

18. Create an external table library_items_ext. Use the ORACLE_LOADER access driver.

Practice 2-1: Managing Schema Objects (continued)

Note: The `emp_dir` directory and `library_items.dat` file are already created for this exercise. `library_items.dat` has records in the following format:

```
2354, 2264, 13.21, 150,  
2355, 2289, 46.23, 200,  
2355, 2264, 50.00, 100,
```

- Open the `lab_02_18.sql` file. Observe the code snippet to create the `library_items_ext` external table. Then replace `<TODO1>`, `<TODO2>`, `<TODO3>`, and `<TODO4>` as appropriate and save the file as `lab_02_18_soln.sql`. Run the script to create the external table.
- Query the `library_items_ext` table.

	CATEGOR...	BOO...	BOOK_P...	QUAN...
1	2354	2264	13.21	150
2	2355	2289	46.23	200
3	2355	2264	50	100

- The HR department needs a report of the addresses of all departments. Create an external table as `dept_add_ext` using the `ORACLE_DATAPUMP` access driver. The report should show the location ID, street address, city, state or province, and country in the output. Use a `NATURAL JOIN` to produce the results.

Note: The `emp_dir` directory is already created for this exercise.

- Open the `lab_02_19.sql` file. Observe the code snippet to create the `dept_add_ext` external table. Then, replace `<TODO1>`, `<TODO2>`, and `<TODO3>` with the appropriate code. Replace `<oraxx_emp4.exp>` and `<oraxx_emp5.exp>` with the appropriate file names. For example, if you are the `ora21` user, your file names are `ora21_emp4.exp` and `ora21_emp5.exp`. Save the script as `lab_02_19_soln.sql`.
- Run the `lab_02_19_soln.sql` script to create the external table.
- Query the `dept_add_ext` table.

Practice 2-1: Managing Schema Objects (continued)

	LOCAT...	STREET_ADDRESS	CITY	STATE_PROVINCE	COUNTRY_NAME
1	1000	1297 Via Cola di Rie	Roma	(null)	Italy
2	1100	93091 Calle della Testa	Venice	(null)	Italy
3	1200	2017 Shinjuku-ku	Tokyo	Tokyo Prefecture	Japan
4	1300	9450 Kamiya-cho	Hiroshima	(null)	Japan
5	1400	2014 Jabberwocky Rd	Southlake	Texas	United States of Amer
6	1500	2011 Interiors Blvd	South San Francisco	California	United States of Amer
7	1600	2007 Zagora St	South Brunswick	New Jersey	United States of Amer
8	1700	2004 Charade Rd	Seattle	Washington	United States of Amer

Note: When you perform the preceding step, two files `oraxx_emp4.exp` and `oraxx_emp5.exp` are created under the default directory `emp_dir`.

20. Create the `emp_books` table and populate it with data. Set the primary key as deferred and observe what happens at the end of the transaction.
- Run the `lab_02_20_a.sql` file to create the `emp_books` table. Observe that the `emp_books_pk` primary key is not created as deferrable.

```
create table succeeded.
```

- Run the `lab_02_20_b.sql` file to populate data into the `emp_books` table. What do you observe?

```
1 rows inserted

Error starting at line 2 in command:
insert into emp_books values(300,'Change Management')
Error report:
SQL Error: ORA-00001: unique constraint (ORA21.EMP_BOOKS_PK) violated
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.
```

- Set the `emp_books_pk` constraint as deferred. What do you observe?

```
Error starting at line 1 in command:
set constraint emp_books_pk deferred
Error report:
SQL Error: ORA-02447: cannot defer a constraint that is not deferrable
02447. 00000 - "cannot defer a constraint that is not deferrable"
*Cause:      An attempt was made to defer a nondeferrable constraint
*Action:     Drop the constraint and create a new one that is deferrable
```

- Drop the `emp_books_pk` constraint.

```
alter table emp_books succeeded.
```

- Modify the `emp_books` table definition to add the `emp_books_pk` constraint as deferrable this time.

Practice 2-1: Managing Schema Objects (continued)

```
alter table emp_books succeeded.
```

f. Set the emp_books_pk constraint as deferred.

```
set constraint succeeded.
```

g. Run the lab_02_20_g.sql file to populate data into the emp_books table.

What do you observe?

```
1 rows inserted  
1 rows inserted  
1 rows inserted
```

h. Commit the transaction. What do you observe?

```
Error report:  
SQL Error: ORA-02091: transaction rolled back  
ORA-00001: unique constraint (ORA21.EMP_BOOKS_PK) violated  
02091. 00000 - "transaction rolled back"  
*Cause:      Also see error 2092. If the transaction is aborted at a remote  
              site then you will only see 2091; if aborted at host then you will  
              see 2092 and 2091.  
*Action:     Add rollback segment and retry the transaction.
```

Practice Solutions 2-1: Managing Schema Objects

1. Create the DEPT2 table based on the following table instance chart. Enter the syntax in the SQL Worksheet. Then, execute the statement to create the table. Confirm that the table is created.

Column Name	ID	NAME
Key Type		
Nulls/Unique		
FK Table		
FK Column		
Data type	NUMBER	VARCHAR2
Length	7	25

```
CREATE TABLE dept2
  (id NUMBER(7),
   name VARCHAR2(25));

DESCRIBE dept2
```

2. Populate the DEPT2 table with data from the DEPARTMENTS table. Include only the columns that you need.

```
INSERT INTO dept2
SELECT department_id, department_name
FROM departments;
```

3. Create the EMP2 table based on the following table instance chart. Enter the syntax in the SQL Worksheet. Then execute the statement to create the table. Confirm that the table is created.

Column Name	ID	LAST_NAME	FIRST_NAME	DEPT_ID
Key Type				
Nulls/Unique				
FK Table				
FK Column				
Data type	NUMBER	VARCHAR2	VARCHAR2	NUMBER
Length	7	25	25	7

Practice Solutions 2-1: Managing Schema Objects (continued)

```
CREATE TABLE emp2
(id          NUMBER(7),
 last_name   VARCHAR2(25),
 first_name  VARCHAR2(25),
 dept_id     NUMBER(7));

DESCRIBE emp2
```

4. Modify the EMP2 table to allow for longer employee last names. Confirm your modification.

```
ALTER TABLE emp2
MODIFY (last_name   VARCHAR2(50));

DESCRIBE emp2
```

5. Create the EMPLOYEES2 table based on the structure of the EMPLOYEES table. Include only the EMPLOYEE_ID, FIRST_NAME, LAST_NAME, SALARY, and DEPARTMENT_ID columns. Name the columns in your new table ID, FIRST_NAME, LAST_NAME, SALARY, and DEPT_ID, respectively.

```
CREATE TABLE employees2 AS
SELECT  employee_id id, first_name, last_name, salary,
        department_id dept_id
FROM    employees;
```

6. Drop the EMP2 table.

```
DROP TABLE emp2;
```

Practice Solutions 2-1: Managing Schema Objects (continued)

7. Query the recycle bin to see whether the table is present.

```
SELECT original_name, operation, droptime
FROM recyclebin;
```

8. Restore the EMP2 table to a state before the DROP statement.

```
FLASHBACK TABLE emp2 TO BEFORE DROP;
DESC emp2;
```

9. Drop the FIRST_NAME column from the EMPLOYEES2 table. Confirm your modification by checking the description of the table.

```
ALTER TABLE employees2
DROP COLUMN first_name;

DESCRIBE employees2
```

10. In the EMPLOYEES2 table, mark the DEPT_ID column as UNUSED. Confirm your modification by checking the description of the table.

```
ALTER TABLE    employees2
SET    UNUSED  (dept_id);

DESCRIBE employees2
```

11. Drop all the UNUSED columns from the EMPLOYEES2 table. Confirm your modification by checking the description of the table.

```
ALTER TABLE employees2
DROP UNUSED COLUMNS;

DESCRIBE employees2
```

Practice Solutions 2-1: Managing Schema Objects (continued)

12. Add a table-level PRIMARY KEY constraint to the EMP2 table on the ID column. The constraint should be named at creation. Name the constraint my_emp_id_pk.

```
ALTER TABLE emp2
ADD CONSTRAINT my_emp_id_pk PRIMARY KEY (id);
```

13. Create a PRIMARY KEY constraint to the DEPT2 table using the ID column. The constraint should be named at creation. Name the constraint my_dept_id_pk.

```
ALTER TABLE dept2
ADD CONSTRAINT my_dept_id_pk PRIMARY KEY(id);
```

14. Add a foreign key reference on the EMP2 table that ensures that the employee is not assigned to a nonexistent department. Name the constraint my_emp_dept_id_fk.

```
ALTER TABLE emp2
ADD CONSTRAINT my_emp_dept_id_fk
FOREIGN KEY (dept_id) REFERENCES dept2(id);
```

15. Modify the EMP2 table. Add a COMMISSION column of the NUMBER data type, precision 2, scale 2. Add a constraint to the COMMISSION column that ensures that a commission value is greater than zero.

```
ALTER TABLE emp2
ADD commission NUMBER(2,2)
CONSTRAINT my_emp_comm_ck CHECK (commission > 0);
```

16. Drop the EMP2 and DEPT2 tables so that they cannot be restored. Check in the recycle bin.

```
DROP TABLE emp2 PURGE;
DROP TABLE dept2 PURGE;

SELECT original_name, operation, droptime
FROM recyclebin;
```

17. Create the DEPT_NAMED_INDEX table based on the following table instance chart. Name the index for the PRIMARY KEY column as DEPT_PK_IDX.

Column Name	Deptno	Dname
Primary Key	Yes	
Data Type	Number	VARCHAR2
Length	4	30

Practice Solutions 2-1: Managing Schema Objects (continued)

```
CREATE TABLE DEPT_NAMED_INDEX
(deptno NUMBER(4)
PRIMARY KEY USING INDEX
(CREATE INDEX dept_pk_idx ON
DEPT_NAMED_INDEX(deptno)),
dname VARCHAR2(30));
```

18. Create an external table `library_items_ext`. Use the `ORACLE_LOADER` access driver.

Note: The `emp_dir` directory and `library_items.dat` are already created for this exercise.

`library_items.dat` has records in the following format:

2354, 2264, 13.21, 150,

2355, 2289, 46.23, 200,

2355, 2264, 50.00, 100,

- a) Open the `lab_02_18.sql` file. Observe the code snippet to create the `library_items_ext` external table. Then, replace `<TODO1>`, `<TODO2>`, `<TODO3>`, and `<TODO4>` as appropriate and save the file as `lab_02_18_soln.sql`.
Run the script to create the external table.

```
CREATE TABLE library_items_ext ( category_id    number(12)
                                , book_id      number(6)
                                , book_price   number(8,2)
                                , quantity     number(8)
                                )

ORGANIZATION EXTERNAL
(TYPE ORACLE_LOADER
DEFAULT DIRECTORY emp_dir
ACCESS PARAMETERS (RECORDS DELIMITED BY NEWLINE
                   FIELDS TERMINATED BY ','))
LOCATION ('library_items.dat')
)
REJECT LIMIT UNLIMITED;
```

Practice Solutions 2-1: Managing Schema Objects (continued)

b) Query the `library_items_ext` table.

```
SELECT * FROM library_items_ext;
```

19. The HR department needs a report of addresses of all the departments. Create an external table as `dept_add_ext` using the `ORACLE_DATAPUMP` access driver. The report should show the location ID, street address, city, state or province, and country in the output. Use a `NATURAL JOIN` to produce the results.

Note: The `emp_dir` directory is already created for this exercise.

- a) Open the `lab_02_19.sql` file. Observe the code snippet to create the `dept_add_ext` external table. Then, replace `<TODO1>`, `<TODO2>`, and `<TODO3>` with appropriate code. Replace `<oraxx_emp4.exp>` and `<oraxx_emp5.exp>` with appropriate file names. For example, if you are user `ora21`, your file names are `ora21_emp4.exp` and `ora21_emp5.exp`. Save the script as `lab_02_19_soln.sql`.

```
CREATE TABLE dept_add_ext (location_id,
                           street_address, city,
                           state_province,
                           country_name)

ORGANIZATION EXTERNAL(
TYPE ORACLE_DATAPUMP
DEFAULT DIRECTORY emp_dir
LOCATION ('oraxx_emp4.exp','oraxx_emp5.exp'))
PARALLEL
AS
SELECT location_id, street_address, city, state_province,
country_name
FROM locations
NATURAL JOIN countries;
```

Note: When you perform the preceding step, two files `oraxx_emp4.exp` and `oraxx_emp5.exp` are created under the default directory `emp_dir`.

Run the `lab_02_19_soln.sql` script to create the external table.

Practice Solutions 2-1: Managing Schema Objects (continued)

b) Query the dept_add_ext table.

```
SELECT * FROM dept_add_ext;
```

20. Create the emp_books table and populate it with data. Set the primary key as deferred and observe what happens at the end of the transaction.

a) Run the lab_02_20a.sql script to create the emp_books table. Observe that the emp_books_pk primary key is not created as deferrable.

```
CREATE TABLE emp_books (book_id number,  
                           title varchar2(20), CONSTRAINT  
emp_books_pk PRIMARY KEY (book_id));
```

b) Run the lab_02_20b.sql script to populate data into the emp_books table.

What do you observe?

```
INSERT INTO emp_books VALUES (300, 'Organizations');  
INSERT INTO emp_books VALUES (300, 'Change Management');
```

The first row is inserted. However, you see the ora-00001 error with the second row insertion.

c) Set the emp_books_pk constraint as deferred. What do you observe?

```
SET CONSTRAINT emp_books_pk DEFERRED;
```

You see the following error: “ORA-02447: Cannot defer a constraint that is not deferrable.”

d) Drop the emp_books_pk constraint.

```
ALTER TABLE emp_books DROP CONSTRAINT emp_books_pk;
```

e) Modify the emp_books table definition to add the emp_books_pk constraint as deferrable this time.

```
ALTER TABLE emp_books ADD (CONSTRAINT emp_books_pk PRIMARY KEY  
(book_id) DEFERRABLE);
```

f) Set the emp_books_pk constraint as deferred.

```
SET CONSTRAINT emp_books_pk DEFERRED;
```

Practice Solutions 2-1: Managing Schema Objects (continued)

g) Run the lab_02_20g.sql script to populate data into the emp_books table.

What do you observe?

```
INSERT INTO emp_books VALUES (300, 'Change Management');  
INSERT INTO emp_books VALUES (300, 'Personality');  
INSERT INTO emp_books VALUES (350, 'Creativity');
```

You see that all the rows are inserted.

h) Commit the transaction. What do you observe?

```
COMMIT;
```

You see that the transaction is rolled back.

Practices and Solutions for Lesson 3

Practice 3-1: Managing Objects with Data Dictionary Views

In this practice, you query the dictionary views to find information about objects in your schema.

1. Query the USER_TABLES data dictionary view to see information about the tables that you own.

	TABLE_NAME
1	REGIONS
2	LOCATIONS
3	DEPARTMENTS
4	JOBS
5	EMPLOYEES
6	JOB_HISTORY
7	EMP_NEW_SAL
8	EMPLOYEES2
9	DEPT_NAMED_INDEX

...

2. Query the ALL_TABLES data dictionary view to see information about all the tables that you can access. Exclude the tables that you own.

Note: Your list may not exactly match the following list:

	TABLE_NAME	OWNER
1	DUAL	SYS
2	SYSTEM_PRIVILEGE_MAP	SYS
3	TABLE_PRIVILEGE_MAP	SYS

...

98	PLAN_TABLE\$	SYS
99	WRI\$_ADV_ASA_RECO_DATA	SYS
100	PSTUBTBL	SYS

3. For a specified table, create a script that reports the column names, data types, and data types' lengths, as well as whether nulls are allowed. Prompt the user to enter the table name. Give appropriate aliases to the DATA_PRECISION and DATA_SCALE columns. Save this script in a file named lab_03_01.sql.

For example, if the user enters DEPARTMENTS, the following output results:

Practice 3-1: Managing Objects with Data Dictionary Views (continued)

	COLUMN_NAME	DATA_TYPE	DATA_LENGTH	PRECISION	SCALE	NULLABLE
1	DEPARTMENT_ID	NUMBER	22	4	0	N
2	DEPARTMENT_NAME	VARCHAR2	30	(null)	(null)	N
3	MANAGER_ID	NUMBER	22	6	0	Y
4	LOCATION_ID	NUMBER	22	4	0	Y

- Create a script that reports the column name, constraint name, constraint type, search condition, and status for a specified table. You must join the USER_CONSTRAINTS and USER_CONS_COLUMNS tables to obtain all this information. Prompt the user to enter the table name. Save the script in a file named lab_03_04.sql. For example, if the user enters DEPARTMENTS, the following output results:

	COLUMN_NAME	CONSTRAINT_NAME	CONSTR...	SEARCH_CONDITION	STATUS
1	DEPARTMENT_NAME	DEPT_NAME_NN	C	"DEPARTMENT_NAME" IS NOT ...	ENABLED
2	DEPARTMENT_ID	DEPT_ID_PK	P	(null)	ENABLED
3	LOCATION_ID	DEPT_LOC_FK	R	(null)	ENABLED
4	MANAGER_ID	DEPT_MGR_FK	R	(null)	ENABLED

- Add a comment to the DEPARTMENTS table. Then query the USER_TAB_COMMENTS view to verify that the comment is present.

	COMMENTS
1	Company department information including name, code, and location.

- Create a synonym for your EMPLOYEES table. Call it EMP. Then find the names of all synonyms that are in your schema.

	SYNONYM_NAME	TABLE_OWNER	TABLE_NAME	DB_LINK
1	TEAM2	ORA22	DEPARTMENTS	(null)
2	EMP	ORA21	EMPLOYEES	(null)

- Run lab_03_07.sql to create the dept50 view for this exercise. You need to determine the names and definitions of all the views in your schema. Create a report that retrieves view information: the view name and text from the USER_VIEWS data dictionary view.

Note: The EMP_DETAILS_VIEW was created as part of your schema.

Note: You can see the complete definition of the view if you use Run Script (or press F5) in SQL Developer. If you use Execute Statement (or press F9) in SQL Developer, scroll horizontally in the result pane. If you use SQL*Plus, to see more contents of a LONG column, use the SET LONG n command, where n is the value of the number of characters of the LONG column that you want to see.

Practice 3-1: Managing Objects with Data Dictionary Views (continued)

VIEW_NAME	TEXT
DEPT50	SELECT employee_id empno, last_name employee, department_id deptno
EMP_DETAILS_VIEW	SELECT e.employee_id, e.job_id, e.manager_id, e.department_id, d.location_id,

- Find the names of your sequences. Write a query in a script to display the following information about your sequences: sequence name, maximum value, increment size, and last number. Name the script `lab_03_08.sql`. Run the statement in your script.

[illegible]

Run the `lab_03_09_tab.sql` script as a prerequisite for exercises 9 through 11. Alternatively, open the script file to copy the code and paste it into your SQL Worksheet. Then execute the script. This script:

- Drops if there are existing tables DEPT2 and EMP2
- Creates the DEPT2 and EMP2 tables

Note: In Practice 2, you should have already dropped the DEPT2 and EMP2 tables so that they cannot be restored.

9. Confirm that both the DEPT2 and EMP2 tables are stored in the data dictionary.

	TABLE_NAME
1	DEPT2
2	EMP2




10. Confirm that the constraints were added by querying the `USER_CONSTRAINTS` view. Note the types and names of the constraints.

R2	CONSTRAINT_NAME	R2	CONSTRAINT_TYPE
1	MY_DEPT_ID_PK		P
2	MY_EMP_ID_PK		P
3	MY_EMP_DEPT_ID_FK		R

11. Display the object names and types from the USER_OBJECTS data dictionary view for the EMP2 and DEPT2 tables.
12. Create the SALES_DEPT table based on the following table instance chart. Name the index for the PRIMARY KEY column SALES_PK_IDX. Then query the data dictionary view to find the index name, table name, and whether the index is unique.

Practice 3-1: Managing Objects with Data Dictionary Views (continued)

Column Name	Team_Id	Location
Primary Key	Yes	
Data Type	Number	VARCHAR2
Length	3	30

 INDEX_NAME	 TABLE_NAME	 UNIQUENESS
1 SALES_PK_IDX	SALES_DEPT	NONUNIQUE

Practice Solutions 3-1: Managing Objects with Data Dictionary Views

1. Query the data dictionary to see information about the tables you own.

```
SELECT table_name
FROM   user_tables;
```

2. Query the dictionary view to see information about all the tables that you can access. Exclude tables that you own.

```
SELECT table_name, owner
FROM   all_tables
WHERE  owner <> 'ORAXX';
```

3. For a specified table, create a script that reports the column names, data types, and data types' lengths, as well as whether nulls are allowed. Prompt the user to enter the table name. Give appropriate aliases to the DATA_PRECISION and DATA_SCALE columns. Save this script in a file named lab_03_01.sql.

```
SELECT column_name, data_type, data_length,
       data_precision PRECISION, data_scale SCALE, nullable
FROM   user_tab_columns
WHERE  table_name = UPPER('&tab_name');
```

To test, run the script and enter DEPARTMENTS as the table name.

4. Create a script that reports the column name, constraint name, constraint type, search condition, and status for a specified table. You must join the USER_CONSTRAINTS and USER_CONS_COLUMNS tables to obtain all this information. Prompt the user to enter the table name. Save the script in a file named lab_03_04.sql.

```
SELECT ucc.column_name, uc.constraint_name,
       uc.constraint_type,
       uc.search_condition, uc.status
FROM   user_constraints uc JOIN user_cons_columns ucc
ON     uc.table_name = ucc.table_name
AND    uc.constraint_name = ucc.constraint_name
AND    uc.table_name = UPPER('&tab_name');
```

To test, run the script and enter DEPARTMENTS as the table name.

Practice Solutions 3-1: Managing Objects with Data Dictionary Views (continued)

5. Add a comment to the DEPARTMENTS table. Then query the USER_TAB_COMMENTS view to verify that the comment is present.

```
COMMENT ON TABLE departments IS
    'Company department information including name, code, and
    location.';

SELECT COMMENTS
FROM   user_tab_comments
WHERE  table_name = 'DEPARTMENTS';
```

6. Create a synonym for your EMPLOYEES table. Call it EMP. Then, find the names of all the synonyms that are in your schema.

```
CREATE SYNONYM emp FOR EMPLOYEES;
SELECT *
FROM   user_synonyms;
```

7. Run lab_03_07.sql to create the dept50 view for this exercise. You need to determine the names and definitions of all the views in your schema. Create a report that retrieves view information: the view name and text from the USER_VIEWS data dictionary view.

Note: The EMP_DETAILS_VIEW was created as part of your schema.

Note: You can see the complete definition of the view if you use Run Script (or press F5) in SQL Developer. If you use Execute Statement (or press F9) in SQL Developer, scroll horizontally in the result pane. If you use SQL*Plus to see more contents of a LONG column, use the SET LONG n command, where n is the value of the number of characters of the LONG column that you want to see.

```
SELECT   view_name, text
FROM     user_views;
```


Practice Solutions 3-1: Managing Objects with Data Dictionary Views (continued)

8. Find the names of your sequences. Write a query in a script to display the following information about your sequences: sequence name, maximum value, increment size, and last number. Name the script lab_03_08.sql. Run the statement in your script.

```
SELECT  sequence_name, max_value, increment_by, last_number
FROM    user_sequences;
```

Run the lab_03_09_tab.sql script as a prerequisite for exercises 9 through 11. Alternatively, open the script file to copy the code and paste it into your SQL Worksheet. Then execute the script. This script:

- Drops the DEPT2 and EMP2 tables
- Creates the DEPT2 and EMP2 tables

Note: In Practice 2, you should have already dropped the DEPT2 and EMP2 tables so that they cannot be restored.

9. Confirm that both the DEPT2 and EMP2 tables are stored in the data dictionary.

```
SELECT  table_name
FROM    user_tables
WHERE   table_name IN ('DEPT2', 'EMP2');
```

10. Query the data dictionary to find out the constraint names and types for both the tables.

```
SELECT  constraint_name, constraint_type
FROM    user_constraints
WHERE   table_name IN ('EMP2', 'DEPT2');
```

11. Query the data dictionary to display the object names and types for both the tables.

```
SELECT  object_name, object_type
FROM    user_objects
WHERE   object_name LIKE 'EMP%'
OR      object_name LIKE 'DEPT%';
```

Practice Solutions 3-1: Managing Objects with Data Dictionary Views (continued)

12. Create the SALES_DEPT table based on the following table instance chart. Name the index for the PRIMARY KEY column as SALES_PK_IDX. Then query the data dictionary view to find the index name, table name, and whether the index is unique.

Column Name	Team_Id	Location
Primary Key	Yes	
Data Type	Number	VARCHAR2
Length	3	30

```
CREATE TABLE SALES_DEPT
  (team_id NUMBER(3)
   PRIMARY KEY USING INDEX
   (CREATE INDEX sales_pk_idx ON
    SALES_DEPT(team_id),
    location VARCHAR2(30));

SELECT INDEX_NAME, TABLE_NAME, UNIQUENESS
FROM USER_INDEXES
WHERE TABLE_NAME = 'SALES_DEPT';
```

Practices and Solutions for Lesson 4

Practice 4-1: Manipulating Large Data Sets

In this practice, you perform multitable INSERT and MERGE operations, and track row versions.

1. Run the lab_04_01.sql script in the lab folder to create the SAL_HISTORY table.
2. Display the structure of the SAL_HISTORY table.

Name	Null	Type
EMPLOYEE_ID		NUMBER(6)
HIRE_DATE		DATE
SALARY		NUMBER(8,2)
3 rows selected		

3. Run the lab_04_03.sql script in the lab folder to create the MGR_HISTORY table.
4. Display the structure of the MGR_HISTORY table.

Name	Null	Type
EMPLOYEE_ID		NUMBER(6)
MANAGER_ID		NUMBER(6)
SALARY		NUMBER(8,2)
3 rows selected		

5. Run the lab_04_05.sql script in the lab folder to create the SPECIAL_SAL table.
6. Display the structure of the SPECIAL_SAL table.

Name	Null	Type
EMPLOYEE_ID		NUMBER(6)
SALARY		NUMBER(8,2)
2 rows selected		

7. a. Write a query to do the following:
 - Retrieve details such as the employee ID, hire date, salary, and manager ID of those employees whose employee ID is less than 125 from the EMPLOYEES table.
 - If the salary is more than \$20,000, insert details such as the employee ID and salary into the SPECIAL_SAL table.

Practice 4-1: Manipulating Large Data Sets (continued)

- Insert details such as the employee ID, hire date, and salary into the SAL_HISTORY table.
- Insert details such as the employee ID, manager ID, and salary into the MGR_HISTORY table.

- b. Display the records from the SPECIAL_SAL table.

	EMPLOYEE_ID	SALARY
1	100	24000

- c. Display the records from the SAL_HISTORY table.

	EMPLOYEE_ID	HIRE_DATE	SALARY
1	101	21-SEP-89	17000
2	102	13-JAN-93	17000
3	103	03-JAN-90	9000
4	104	21-MAY-91	6000
5	105	25-JUN-97	4800
6	106	05-FEB-98	4800
7	107	07-FEB-99	4200

- d. Display the records from the MGR_HISTORY table.

	EMPLOYEE_ID	MANAGER_ID	SALARY
1	101	100	17000
2	102	100	17000
3	103	102	9000
4	104	103	6000
5	105	103	4800
6	106	103	4800
7	107	103	4200

8.

- Run the lab_04_08a.sql script in the lab folder to create the SALES_WEEK_DATA table.
- Run the lab_04_08b.sql script in the lab folder to insert records into the SALES_WEEK_DATA table.

Practice 4-1: Manipulating Large Data Sets (continued)

- c. Display the structure of the SALES_WEEK_DATA table.

Name	Null	Type
ID		NUMBER(6)
WEEK_ID		NUMBER(2)
QTY_MON		NUMBER(8,2)
QTY_TUE		NUMBER(8,2)
QTY_WED		NUMBER(8,2)
QTY_THUR		NUMBER(8,2)
QTY_FRI		NUMBER(8,2)
7 rows selected		

- d. Display the records from the SALES_WEEK_DATA table.

ID	WEEK_ID	QTY_MON	QTY_TUE	QTY_WED	QTY_THUR	QTY_FRI	
1	200	6	2050	2200	1700	1200	3000

- e. Run the lab_04_08_e.sql script in the lab folder to create the EMP_SALES_INFO table.
- f. Display the structure of the EMP_SALES_INFO table.

Name	Null	Type
ID		NUMBER(6)
WEEK		NUMBER(2)
QTY_SALES		NUMBER(8,2)
3 rows selected		

- g. Write a query to do the following:

- Retrieve details such as ID, week ID, sales quantity on Monday, sales quantity on Tuesday, sales quantity on Wednesday, sales quantity on Thursday, and sales quantity on Friday from the SALES_WEEK_DATA table.
- Build a transformation such that each record retrieved from the SALES_WEEK_DATA table is converted into multiple records for the EMP_SALES_INFO table.

Hint: Use a pivoting INSERT statement.

- h. Display the records from the EMP_SALES_INFO table.

	AZ	ID	AZ	WEEK	AZ	QTY_SALES
1		200		6		2050
2		200		6		2200
3		200		6		1700
4		200		6		1200
5		200		6		3000

9. You have the data of past employees stored in a flat file called emp.data. You want to store the names and email IDs of all employees, past and present, in a table. To do

Practice 4-1: Manipulating Large Data Sets (continued)

this, first create an external table called EMP_DATA using the emp.dat source file in the emp_dir directory. Use the lab_04_09.sql script to do this.

10. Next, run the lab_04_10.sql script to create the EMP_HIST table.
 - a. Increase the size of the email column to 45.
 - b. Merge the data in the EMP_DATA table created in the last lab into the data in the EMP_HIST table. Assume that the data in the external EMP_DATA table is the most up-to-date. If a row in the EMP_DATA table matches the EMP_HIST table, update the email column of the EMP_HIST table to match the EMP_DATA table row. If a row in the EMP_DATA table does not match, insert it into the EMP_HIST table. Rows are considered matching when the employee's first and last names are identical.
 - c. Retrieve the rows from EMP_HIST after the merge.

	FIRST_NAME	LAST_NAME	EMAIL
1	Ellen	Abel	EABEL
2	Sundar	Ande	SANDE
3	Mozhe	Atkinson	MATKINSO
4	David	Austin	DAUSTIN
5	Hermann	Baer	HBAER
6	Shelli	Baida	SBAIDA
7	Amit	Banda	ABANDA
8	Elizabeth	Bates	EBATES
9	Sarah	Bell	SBELL
10	David	Bernstein	DBERNSTE
11	Laura	Bissot	LBISSOT

11. Create the EMP3 table by using the lab_04_11.sql script. In the EMP3 table, change the department for Kochhar to 60 and commit your change. Next, change the department for Kochhar to 50 and commit your change. Track the changes to Kochhar by using the Row Versions feature.

	START_DATE	END_DATE	DEPARTMENT_ID
1	18-JUN-09 06.04.26.0000000000 PM	(null)	50
2	18-JUN-09 06.04.26.0000000000 PM	18-JUN-09 06.04.26.0000000000 PM	60
3	(null)	18-JUN-09 06.04.26.0000000000 PM	90

Practice Solutions 4-1: Manipulating Large Data Sets

1. Run the `lab_04_01.sql` script in the lab folder to create the `SAL_HISTORY` table.
2. Display the structure of the `SAL_HISTORY` table.

```
DESC sal_history
```

3. Run the `lab_04_03.sql` script in the lab folder to create the `MGR_HISTORY` table.
4. Display the structure of the `MGR_HISTORY` table.

```
DESC mgr_history
```

5. Run the `lab_04_05.sql` script in the lab folder to create the `SPECIAL_SAL` table.
6. Display the structure of the `SPECIAL_SAL` table.

```
DESC special_sal
```

7. a) Write a query to do the following:
 - Retrieve details such as the employee ID, hire date, salary, and manager ID of those employees whose employee ID is less than 125 from the `EMPLOYEES` table.
 - If the salary is more than \$20,000, insert details such as the employee ID and salary into the `SPECIAL_SAL` table.
 - Insert details such as the employee ID, hire date, and salary into the `SAL_HISTORY` table.
 - Insert details such as the employee ID, manager ID, and salary into the `MGR_HISTORY` table.

Practice Solutions 4-1: Manipulating Large Data Sets (continued)

```
INSERT ALL
WHEN SAL > 20000 THEN
INTO special_sal VALUES (EMPID, SAL)
ELSE
INTO sal_history VALUES (EMPID, HIREDATE, SAL)
INTO mgr_history VALUES (EMPID, MGR, SAL)
SELECT employee_id EMPID, hire_date HIREDATE,
salary SAL, manager_id MGR
FROM employees
WHERE employee_id < 125;
```

- b) Display the records from the SPECIAL_SAL table.

```
SELECT * FROM special_sal;
```

- c) Display the records from the SAL_HISTORY table.

```
SELECT * FROM sal_history;
```

- d) Display the records from the MGR_HISTORY table.

```
SELECT * FROM mgr_history;
```

8. a) Run the lab_04_08a.sql script in the lab folder to create the SALES_WEEK_DATA table.
- b) Run the lab_04_08b.sql script in the lab folder to insert records into the SALES_WEEK_DATA table.
- c) Display the structure of the SALES_WEEK_DATA table.

```
DESC sales_week_data
```

- d) Display the records from the SALES_WEEK_DATA table.

```
SELECT * FROM SALES_WEEK_DATA;
```


Practice Solutions 4-1: Manipulating Large Data Sets (continued)

e) Run the lab_04_08_e.sql script in the lab folder to create the EMP_SALES_INFO table.

f) Display the structure of the EMP_SALES_INFO table.

```
DESC emp_sales_info
```

g) Write a query to do the following:

- Retrieve details such as the employee ID, week ID, sales quantity on Monday, sales quantity on Tuesday, sales quantity on Wednesday, sales quantity on Thursday, and sales quantity on Friday from the SALES_WEEK_DATA table.
- Build a transformation such that each record retrieved from the SALES_WEEK_DATA table is converted into multiple records for the EMP_SALES_INFO table.

Hint: Use a pivoting INSERT statement.

```
INSERT ALL
  INTO emp_sales_info VALUES (id, week_id, QTY_MON)
  INTO emp_sales_info VALUES (id, week_id, QTY_TUE)
  INTO emp_sales_info VALUES (id, week_id, QTY_WED)
  INTO emp_sales_info VALUES (id, week_id, QTY_THUR)
  INTO emp_sales_info VALUES (id, week_id, QTY_FRI)
SELECT ID, week_id, QTY_MON, QTY_TUE, QTY_WED,
       QTY_THUR, QTY_FRI FROM sales_week_data;
```

h) Display the records from the SALES_INFO table.

```
SELECT * FROM emp_sales_info;
```

Practice Solutions 4-1: Manipulating Large Data Sets (continued)

9. You have the data of past employees stored in a flat file called `emp.dat`. You want to store the names and email IDs of all employees past and present in a table. To do this, first create an external table called `EMP_DATA` using the `emp.dat` source file in the `emp_dir` directory. You can use the script in `lab_04_09.sql` to do this.

```
CREATE TABLE emp_data
  (first_name  VARCHAR2(20)
  ,last_name   VARCHAR2(20)
  , email      VARCHAR2(30)
  )
ORGANIZATION EXTERNAL
(
  TYPE oracle_loader
  DEFAULT DIRECTORY emp_dir
  ACCESS PARAMETERS
  (
    RECORDS DELIMITED BY NEWLINE CHARACTERSET US7ASCII
    NOBADFILE
    NOLOGFILE
    FIELDS
    ( first_name POSITION ( 1:20) CHAR
    , last_name  POSITION (22:41) CHAR
    , email      POSITION (43:72) CHAR )
  )
  LOCATION ('emp.dat') ) ;
```

10. Next, run the `lab_04_10.sql` script to create the `EMP_HIST` table.
- a) Increase the size of the email column to 45.

```
ALTER TABLE emp_hist MODIFY email varchar(45);
```

- b) Merge the data in the `EMP_DATA` table created in the last lab into the data in the `EMP_HIST` table. Assume that the data in the external `EMP_DATA` table is the most up-to-date. If a row in the `EMP_DATA` table matches the `EMP_HIST` table, update the email column of the `EMP_HIST` table to match the `EMP_DATA` table row. If a row in the `EMP_DATA` table does not match, insert it into the `EMP_HIST` table. Rows are considered matching when the employee's first and last names are identical.

```
MERGE INTO EMP_HIST f USING EMP_DATA h
ON (f.first_name = h.first_name
AND f.last_name = h.last_name)
```

Practice Solutions 4-1: Manipulating Large Data Sets (continued)

```
WHEN MATCHED THEN
  UPDATE SET f.email = h.email
WHEN NOT MATCHED THEN
  INSERT (f.first_name
        , f.last_name
        , f.email)
VALUES (h.first_name
      , h.last_name
      , h.email);
```

- c) Retrieve the rows from EMP_HIST after the merge.

```
SELECT * FROM emp_hist;
```

11. Create the EMP3 table using the lab_04_11.sql script. In the EMP3 table, change the department for Kochhar to 60 and commit your change. Next, change the department for Kochhar to 50 and commit your change. Track the changes to Kochhar using the Row Versions feature.

```
UPDATE emp3 SET department_id = 60
WHERE last_name = 'Kochhar';
COMMIT;
UPDATE emp3 SET department_id = 50
WHERE last_name = 'Kochhar';
COMMIT;
```

```
SELECT VERSIONS_STARTTIME "START_DATE",
       VERSIONS_ENDTIME "END_DATE", DEPARTMENT_ID
FROM EMP3
     VERSIONS BETWEEN SCN MINVALUE AND MAXVALUE
WHERE LAST_NAME = 'Kochhar';
```

Practices and Solutions for Lesson 5

Practice 5-1: Managing Data in Different Time Zones

In this practice, you display time zone offsets, `CURRENT_DATE`, `CURRENT_TIMESTAMP`, and `LOCALTIMESTAMP`. You also set time zones and use the `EXTRACT` function.

1. Alter the session to set `NLS_DATE_FORMAT` to `DD-MON-YYYY HH24:MI:SS`.
2. a. Write queries to display the time zone offsets (`TZ_OFFSET`) for the following time zones.

- *US/Pacific-New*

	<code>TZ_OFFSET('US/PACIFIC-NEW')</code>
1	-07:00

- *Singapore*

	<code>TZ_OFFSET('SINGAPORE')</code>
1	+08:00

- *Egypt*

	<code>TZ_OFFSET('EGYPT')</code>
1	+03:00

- b. Alter the session to set the `TIME_ZONE` parameter value to the time zone offset of US/Pacific-New.
- c. Display `CURRENT_DATE`, `CURRENT_TIMESTAMP`, and `LOCALTIMESTAMP` for this session.
- d. Alter the session to set the `TIME_ZONE` parameter value to the time zone offset of Singapore.
- e. Display `CURRENT_DATE`, `CURRENT_TIMESTAMP`, and `LOCALTIMESTAMP` for this session.

Note: The output might be different based on the date when the command is executed.

	<code>CURRENT_DATE</code>	<code>CURRENT_TIMESTAMP</code>	<code>LOCALTIMESTAMP</code>
1	23-JUN-2009 15:12:08	23-JUN-09 03.12.08.000000000 PM +08:00	23-JUN-09 03.12.08.000000000 PM

Note: Observe in the preceding practice that `CURRENT_DATE`, `CURRENT_TIMESTAMP`, and `LOCALTIMESTAMP` are sensitive to the session time zone.

3. Write a query to display `DBTIMEZONE` and `SESSIONTIMEZONE`.

	<code>DBTIMEZONE</code>	<code>SESSIONTIMEZONE</code>
1	+00:00	+08:00

4. Write a query to extract the `YEAR` from the `HIRE_DATE` column of the `EMPLOYEES` table for those employees who work in department 80.

Practice 5-1: Managing Data in Different Time Zones (continued)

R	LAST_NAME	R	EXTRACT(YEARFROMHIRE_DATE)
1	Russell		1996
2	Partners		1997
3	Errazuriz		1997
4	Cambrault		1999
5	Zlotkey		2000
6	Tucker		1997
7	Bernstein		1997

5. Alter the session to set NLS_DATE_FORMAT to DD-MON-YYYY.
6. Examine and run the lab_05_06.sql script to create the SAMPLE_DATES table and populate it.
 - a. Select from the table and view the data.

R	DATE_COL
1	23-JUN-2009

- b. Modify the data type of the DATE_COL column and change it to TIMESTAMP. Select from the table to view the data.

R	DATE_COL
1	23-JUN-09 02.14.52.000000000 PM

- c. Try to modify the data type of the DATE_COL column and change it to TIMESTAMP WITH TIME ZONE. What happens?
7. Create a query to retrieve last names from the EMPLOYEES table and calculate the review status. If the year hired was 1998, display Needs Review for the review status; otherwise, display not this year! Name the review status column Review. Sort the results by the HIRE_DATE column.
Hint: Use a CASE expression with the EXTRACT function to calculate the review status.

R	LAST_NAME	R	Review
1	King		not this year!
2	Whalen		not this year!
3	Kochhar		not this year!
4	Hunold		not this year!
5	Ernst		not this year!
6	De Haan		not this year!
7	Mavris		not this year!

...

Practice 5-1: Managing Data in Different Time Zones (continued)

8. Create a query to print the last names and the number of years of service for each employee. If the employee has been employed for five or more years, print 5 years of service. If the employee has been employed for 10 or more years, print 10 years of service. If the employee has been employed for 15 or more years, print 15 years of service. If none of these conditions match, print maybe next year! Sort the results by the HIRE_DATE column. Use the EMPLOYEES table.

Hint: Use CASE expressions and TO_YMINTERVAL.

R	LAST_NAME	R	HIRE_DATE	R	SYSDATE	R	Awards
1	OConnell		21-JUN-1999		23-JUN-2009		10 years of service
2	Grant		13-JAN-2000		23-JUN-2009		5 years of service
3	Whalen		17-SEP-1987		23-JUN-2009		15 years of service
4	Hartstein		17-FEB-1996		23-JUN-2009		10 years of service
5	Fay		17-AUG-1997		23-JUN-2009		10 years of service
6	Mavris		07-JUN-1994		23-JUN-2009		15 years of service

...

Practice Solutions 5-1: Managing Data in Different Time Zones

1. Alter the session to set NLS_DATE_FORMAT to DD-MON-YYYY HH24:MI:SS.

```
ALTER SESSION SET NLS_DATE_FORMAT =  
'DD-MON-YYYY HH24:MI:SS';
```

2. a. Write queries to display the time zone offsets (TZ_OFFSET) for the following time zones: *US/Pacific-New*, *Singapore*, and *Egypt*.

US/Pacific-New

```
SELECT TZ_OFFSET ('US/Pacific-New') from dual;
```

Singapore

```
SELECT TZ_OFFSET ('Singapore') from dual;
```

Egypt

```
SELECT TZ_OFFSET ('Egypt') from dual;
```

- b. Alter the session to set the TIME_ZONE parameter value to the time zone offset of *US/Pacific-New*.

```
ALTER SESSION SET TIME_ZONE = '-7:00';
```

- c. Display CURRENT_DATE, CURRENT_TIMESTAMP, and LOCALTIMESTAMP for this session.

Note: The output may be different based on the date when the command is executed.

```
SELECT CURRENT_DATE, CURRENT_TIMESTAMP,  
LOCALTIMESTAMP FROM DUAL;
```

- d. Alter the session to set the TIME_ZONE parameter value to the time zone offset of *Singapore*.

```
ALTER SESSION SET TIME_ZONE = '+8:00';
```

- e. Display CURRENT_DATE, CURRENT_TIMESTAMP, and LOCALTIMESTAMP for this session.

Note: The output might be different, based on the date when the command is executed.

```
SELECT CURRENT_DATE, CURRENT_TIMESTAMP,  
LOCALTIMESTAMP FROM DUAL;
```

Practice Solutions 5-1: Managing Data in Different Time Zones (continued)

Note: Observe in the preceding practice that `CURRENT_DATE`, `CURRENT_TIMESTAMP`, and `LOCALTIMESTAMP` are all sensitive to the session time zone.

3. Write a query to display `DBTIMEZONE` and `SESSIONTIMEZONE`.

```
SELECT DBTIMEZONE, SESSIONTIMEZONE
FROM DUAL;
```

4. Write a query to extract `YEAR` from the `HIRE_DATE` column of the `EMPLOYEES` table for those employees who work in department 80.

```
SELECT last_name, EXTRACT (YEAR FROM HIRE_DATE)
FROM employees
WHERE department_id = 80;
```

5. Alter the session to set `NLS_DATE_FORMAT` to `DD-MON-YYYY`.

```
ALTER SESSION SET NLS_DATE_FORMAT = 'DD-MON-YYYY';
```

6. Examine and run the `lab_05_06.sql` script to create the `SAMPLE_DATES` table and populate it.

- a. Select from the table and view the data.

```
SELECT * FROM sample_dates;
```

- b. Modify the data type of the `DATE_COL` column and change it to `TIMESTAMP`.
Select from the table to view the data.

```
ALTER TABLE sample_dates MODIFY date_col TIMESTAMP;
SELECT * FROM sample_dates;
```

- c. Try to modify the data type of the `DATE_COL` column and change it to `TIMESTAMP WITH TIME ZONE`. What happens?

```
ALTER TABLE sample_dates MODIFY date_col
TIMESTAMP WITH TIME ZONE;
```


Practice Solutions 5-1: Managing Data in Different Time Zones (continued)

You are unable to change the data type of the DATE_COL column because the Oracle server does not permit you to convert from TIMESTAMP to TIMESTAMP WITH TIMEZONE by using the ALTER statement.

7. Create a query to retrieve last names from the EMPLOYEES table and calculate the review status. If the year hired was 1998, display Needs Review for the review status; otherwise, display not this year! Name the review status column Review. Sort the results by the HIRE_DATE column.

Hint: Use a CASE expression with the EXTRACT function to calculate the review status.

```
SELECT e.last_name
       , (CASE extract(year from e.hire_date)
           WHEN 1998 THEN 'Needs Review'
           ELSE 'not this year!'
         END ) AS "Review "
FROM   employees e
ORDER BY e.hire_date;
```

8. Create a query to print the last names and the number of years of service for each employee. If the employee has been employed five or more years, print 5 years of service. If the employee has been employed 10 or more years, print 10 years of service. If the employee has been employed 15 or more years, print 15 years of service. If none of these conditions match, print maybe next year! Sort the results by the HIRE_DATE column. Use the EMPLOYEES table.

Hint: Use CASE expressions and TO_YMINTERVAL.

```
SELECT e.last_name, hire_date, sysdate,
       (CASE
         WHEN (sysdate -TO_YMINTERVAL('15-0'))>=
              hire_date THEN      '15 years of service'
         WHEN (sysdate -TO_YMINTERVAL('10-0'))>= hire_date
              THEN      '10 years of service'
         WHEN (sysdate - TO_YMINTERVAL('5-0'))>= hire_date
              THEN '5 years of service'
         ELSE 'maybe next year!'
       END) AS "Awards"
FROM   employees e;
```

Practices and Solutions for Lesson 6

Practice 6-1: Retrieving Data by Using Subqueries

In this practice, you write multiple-column subqueries, and correlated and scalar subqueries. You also solve problems by writing the WITH clause.

1. Write a query to display the last name, department number, and salary of any employee whose department number and salary both match the department number and salary of any employee who earns a commission.

	LAST_NAME	DEPARTMENT_ID	SALARY
1	Russell	80	14000
2	Partners	80	13500
3	Errazuriz	80	12000

2. Display the last name, department name, and salary of any employee whose salary and commission match the salary and commission of any employee located in location ID 1700.

	LAST_NAME	DEPARTMENT_NAME	SALARY
1	Whalen	Administration	4400
2	Higgins	Accounting	12000
3	Greenberg	Finance	12000
4	Gietz	Accounting	8300

3. Create a query to display the last name, hire date, and salary for all employees who have the same salary and commission as Kochhar.

Note: Do not display Kochhar in the result set.

	LAST_NAME	HIRE_DATE	SALARY
1	De Haan	13-JAN-1993	17000

4. Create a query to display the employees who earn a salary that is higher than the salary of all the sales managers (JOB_ID = 'SA_MAN'). Sort the results from the highest to the lowest.

Oracle University and NETEC, S.A. de C.V. use only.

	LAST_NAME	JOB_ID	SALARY
1	King	AD_PRES	24000
2	De Haan	AD_VP	17000
3	Kochhar	AD_VP	17000

5. Display details such as the employee ID, last name, and department ID of those employees who live in cities the names of which begin with *T*.

	EMPLOYEE_ID	LAST_NAME	DEPARTMENT_ID
1	202	Fay	20
2	201	Hartstein	20

6. Write a query to find all employees who earn more than the average salary in their departments. Display last name, salary, department ID, and the average salary for the department. Sort by average salary and round to two decimals. Use aliases for the columns retrieved by the query as shown in the sample output.

	ENAME	SALARY	DEPTNO	DEPT_AVG
1	Fripp	8200	50	3475.5555555555555555555555555556
2	Kaufling	7900	50	3475.5555555555555555555555555556
3	Chung	3800	50	3475.5555555555555555555555555556
4	Mourgos	5800	50	3475.5555555555555555555555555556
5	Bell	4000	50	3475.5555555555555555555555555556
6	Rajs	3500	50	3475.5555555555555555555555555556
7	Bull	4100	50	3475.5555555555555555555555555556
8	Everett	3900	50	3475.5555555555555555555555555556

7. Find all employees who are not supervisors.
 - a. First, do this using the NOT EXISTS operator.

Practice 6-1: Retrieving Data by Using Subqueries (continued)

R	LAST_NAME
1	Abel
2	Ande
3	Atkinson
4	Austin
5	Baer
6	Baida

- b. Can this be done by using the NOT IN operator? How, or why not?
8. Write a query to display the last names of the employees who earn less than the average salary in their departments.

R	LAST_NAME
1	Chen
2	Sciarra
3	Urman
4	Popp
5	Khoo
6	Baida

9. Write a query to display the last names of the employees who have one or more coworkers in their departments with later hire dates but higher salaries.

R	LAST_NAME
1	Vargas
2	Patel
3	Olson
4	Marlow
5	Landry
6	Perkins

10. Write a query to display the employee ID, last names, and department names of all the employees.

Note: Use a scalar subquery to retrieve the department name in the SELECT statement.

	EMPLOYEE_ID	LAST_NAME	DEPARTMENT
1	205	Higgins	Accounting
2	206	Gietz	Accounting
3	200	Whalen	Administration
4	100	King	Executive
5	101	Kochhar	Executive

...

Practice 6-1: Retrieving Data by Using Subqueries (continued)

105	196 Walsh	Shipping
106	197 Feeney	Shipping
107	178 Grant	(null)

11. Write a query to display the department names of those departments whose total salary cost is above one-eighth (1/8) of the total salary cost of the whole company. Use the WITH clause to write this query. Name the query SUMMARY.

R	DEPARTMENT_NAME	R	DEPT_TOTAL
1	Sales		304500
2	Shipping		156400

Practice Solutions 6-1: Retrieving Data by Using Subqueries

1. Write a query to display the last name, department number, and salary of any employee whose department number and salary match the department number and salary of any employee who earns a commission.

```
SELECT last_name, department_id, salary
FROM   employees
WHERE  (salary, department_id) IN
      (SELECT salary, department_id
       FROM   employees
       WHERE  commission_pct IS NOT NULL);
```

2. Display the last name, department name, and salary of any employee whose salary and commission match the salary and commission of any employee located in location ID1700.

```
SELECT e.last_name, d.department_name, e.salary
FROM   employees e, departments d
WHERE  e.department_id = d.department_id
AND    (salary, NVL(commission_pct,0)) IN
      (SELECT salary, NVL(commission_pct,0)
       FROM   employees e, departments d
       WHERE  e.department_id = d.department_id
       AND d.location_id = 1700);
```

3. Create a query to display the last name, hire date, and salary for all employees who have the same salary and commission as Kochhar.

Note: Do not display Kochhar in the result set.

```
SELECT last_name, hire_date, salary
FROM   employees
WHERE  (salary, NVL(commission_pct,0)) IN
      (SELECT salary, NVL(commission_pct,0)
       FROM   employees
       WHERE  last_name = 'Kochhar')
AND last_name != 'Kochhar';
```

4. Create a query to display the employees who earn a salary that is higher than the salary of all the sales managers (JOB_ID = 'SA_MAN'). Sort the results on salary from the highest to the lowest.

```
SELECT last_name, job_id, salary
FROM   employees
WHERE  salary > ALL
      (SELECT salary
       FROM   employees
       WHERE  job_id = 'SA_MAN')
ORDER BY salary DESC;
```

Practice Solutions 6-1: Retrieving Data by Using Subqueries (continued)

5. Display details such as the employee ID, last name, and department ID of those employees who live in cities the names of which begin with *T*.

```
SELECT employee_id, last_name, department_id
FROM   employees
WHERE  department_id IN (SELECT department_id
                        FROM departments
                        WHERE location_id IN
                              (SELECT location_id
                               FROM locations
                               WHERE city LIKE 'T%'));
```

6. Write a query to find all employees who earn more than the average salary in their departments. Display last name, salary, department ID, and the average salary for the department. Sort by average salary. Use aliases for the columns retrieved by the query as shown in the sample output.

```
SELECT e.last_name ename, e.salary salary,
       e.department_id deptno, AVG(a.salary) dept_avg
FROM   employees e, employees a
WHERE  e.department_id = a.department_id
AND    e.salary > (SELECT AVG(salary)
                  FROM   employees
                  WHERE  department_id = e.department_id )
GROUP BY e.last_name, e.salary, e.department_id
ORDER BY AVG(a.salary);
```

Practice Solutions 6-1: Retrieving Data by Using Subqueries (continued)

7. Find all employees who are not supervisors.
 - a. First, do this by using the NOT EXISTS operator.

```
SELECT outer.last_name
FROM   employees outer
WHERE  NOT EXISTS (SELECT 'X'
                   FROM employees inner
                   WHERE inner.manager_id =
                       outer.employee_id);
```

- b. Can this be done by using the NOT IN operator? How, or why not?

```
SELECT outer.last_name
FROM   employees outer
WHERE  outer.employee_id
NOT IN (SELECT inner.manager_id
       FROM   employees inner);
```

This alternative solution is not a good one. The subquery picks up a NULL value, so the entire query returns no rows. The reason is that all conditions that compare a NULL value result in NULL. Whenever NULL values are likely to be part of the value set, *do not* use NOT IN as a substitute for NOT EXISTS.

8. Write a query to display the last names of the employees who earn less than the average salary in their departments.

```
SELECT last_name
FROM   employees outer
WHERE  outer.salary < (SELECT AVG(inner.salary)
                     FROM employees inner
                     WHERE inner.department_id
                         = outer.department_id);
```


Practice Solutions 6-1: Retrieving Data by Using Subqueries (continued)

9. Write a query to display the last names of employees who have one or more coworkers in their departments with later hire dates but higher salaries.

```
SELECT last_name
FROM employees outer
WHERE EXISTS (SELECT 'X'
              FROM employees inner
              WHERE inner.department_id =
                    outer.department_id
              AND inner.hire_date > outer.hire_date
              AND inner.salary > outer.salary);
```

10. Write a query to display the employee ID, last names, and department names of all employees.

Note: Use a scalar subquery to retrieve the department name in the SELECT statement.

```
SELECT employee_id, last_name,
       (SELECT department_name
        FROM departments d
        WHERE e.department_id =
              d.department_id ) department
FROM employees e
ORDER BY department;
```

11. Write a query to display the department names of those departments whose total salary cost is above one-eighth (1/8) of the total salary cost of the whole company. Use the WITH clause to write this query. Name the query SUMMARY.

```
WITH
summary AS (
  SELECT d.department_name, SUM(e.salary) AS dept_total
  FROM employees e, departments d
  WHERE e.department_id = d.department_id
  GROUP BY d.department_name)
SELECT department_name, dept_total
FROM summary
WHERE dept_total > ( SELECT SUM(dept_total) * 1/8
                   FROM summary )
ORDER BY dept_total DESC;
```

Practices and Solutions for Lesson 7

Practice 7-1: Regular Expression Support

In this practice, you use regular expressions functions to search for, replace, and manipulate data. You also create a new CONTACTS table and add a CHECK constraint to the p_number column to ensure that phone numbers are entered into the database in a specific standard format.

1. Write a query to search the EMPLOYEES table for all the employees whose first names start with “Ki” or “Ko.”

	FIRST_NAME	LAST_NAME
1	Janette	King
2	Steven	King
3	Neena	Kochhar

2. Create a query that removes the spaces in the STREET_ADDRESS column of the LOCATIONS table in the display. Use “Street Address” as the column heading.

	Street Address
1	1297ViaColadiRie
2	93091Calle dellaTesta
3	2017Shinjuku-ku
4	9450Kamiya-cho
5	2014JabberwockyRd
6	2011InteriorsBlvd
7	2007ZagoraSt

3. Create a query that displays “St” replaced by “Street” in the STREET_ADDRESS column of the LOCATIONS table. Be careful that you do not affect any rows that already have “Street” in them. Display only those rows that are affected.

	REGEXP_REPLACE(STREET_ADDRESS,'ST\$','STREET')
1	2007 Zagora Street
2	6092 Boxwood Street
3	12-98 Victoria Street
4	8204 Arthur Street

4. Create a contacts table and add a check constraint to the p_number column to enforce the following format mask to ensure that phone numbers are entered into the database in the following standard format: (XXX) XXX-XXXX. The table should have the following columns:

- l_name varchar2(30)
- p_number varchar2(30)

Practice 7-1: Regular Expression Support (continued)

5. Run the SQL script `lab_07_05.sql` to insert the following seven phone numbers into the `contacts` table. Which numbers are added?

l_name Column Value	p_number Column Value
NULL	'(650) 555-5555'
NULL	'(215) 555-3427'
NULL	'650 555-5555'
NULL	'650 555 5555'
NULL	'650-555-5555'
NULL	'(650)555-5555'
NULL	' (650) 555-5555'

6. Write a query to find the number of occurrences of the DNA pattern `ctc` in the string `gtctcgtctcgttctgtctgtcgttctg`. Ignore case-sensitivity.

	1	2
SQL	COUNT_DNA	

Practice Solutions 7-1: Regular Expression Support

1. Write a query to search the EMPLOYEES table for all employees whose first names start with “Ki” or “Ko.”

```
SELECT first_name, last_name
FROM employees
WHERE REGEXP_LIKE (last_name, '^K(i|o).');
```

2. Create a query that removes the spaces in the STREET_ADDRESS column of the LOCATIONS table in the display. Use “Street Address” as the column heading.

```
SELECT regexp_replace (street_address, ' ', '') AS "Street
Address"
FROM locations;
```

3. Create a query that displays “St” replaced by “Street” in the STREET_ADDRESS column of the LOCATIONS table. Be careful that you do not affect any rows that already have “Street” in them. Display only those rows, which are affected.

```
SELECT regexp_replace (street_address, 'St$',
'Street')
FROM locations
WHERE regexp_like (street_address, 'St');
```

4. Create a contacts table and add a check constraint to the p_number column to enforce the following format mask to ensure that phone numbers are entered into the database in the following standard format: (XXX) XXX-XXXX. The table should have the following columns:

- l_name varchar2(30)
- p_number varchar2 (30)

```
CREATE TABLE contacts
(
  l_name      VARCHAR2(30),
  p_number    VARCHAR2(30)
  CONSTRAINT p_number_format
  CHECK ( REGEXP_LIKE ( p_number, '^\\(\\d{3}\\) \\d{3}-
\\d{4}$' ) )
);
```

Practice Solutions 7-1: Regular Expression Support (continued)

5. Run the `lab_07_05.sql` SQL script to insert the following seven phone numbers into the `contacts` table. Which numbers are added?
Only the first two `INSERT` statements use a format that conforms to the `c_contacts_pnf` constraint; the remaining statements generate `CHECK` constraint errors.
6. Write a query to find the number of occurrences of the DNA pattern `ctc` in the string
`gtctcgtctcgttctgtctgtcgttctg`. Use the alias `Count_DNA`. Ignore case-sensitivity.

```
SELECT REGEXP_COUNT('gtctcgtctcgttctgtctgtcgttctg','ctc')
AS Count_DNA
FROM dual;
```


B

Table Descriptions

ORACLE

Copyright © 2009, Oracle. All rights reserved.

Schema Description

Overall Description

The Oracle database sample schemas portray a sample company that operates worldwide to fill orders for several different products. The company has three divisions:

- **Human Resources:** Tracks information about the employees and facilities
- **Order Entry:** Tracks product inventories and sales through various channels
- **Sales History:** Tracks business statistics to facilitate business decisions

Each of these divisions is represented by a schema. In this course, you have access to the objects in all the schemas. However, the emphasis of the examples, demonstrations, and practices is on the Human Resources (HR) schema.

All scripts necessary to create the sample schemas reside in the \$ORACLE_HOME/demo/schema/ folder.

Human Resources (HR)

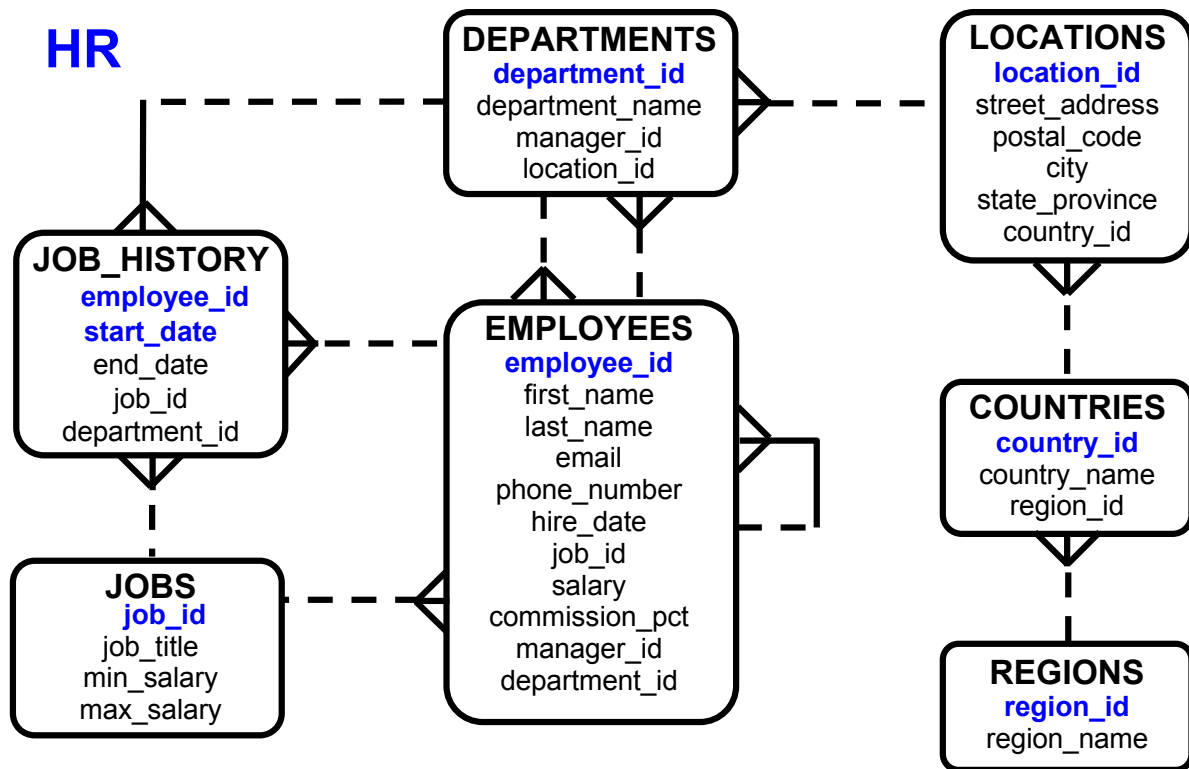
This is the schema that is used in this course. In the Human Resource (HR) records, each employee has an identification number, email address, job identification code, salary, and manager. Some employees earn commissions in addition to their salary.

The company also tracks information about jobs within the organization. Each job has an identification code, job title, and a minimum and maximum salary range. Some employees have been with the company for a long time and have held different positions within the company. When an employee resigns, the duration the employee was working for, the job identification number, and the department are recorded.

The sample company is regionally diverse, so it tracks the locations of its warehouses and departments. Each employee is assigned to a department, and each department is identified either by a unique department number or a short name. Each department is associated with one location, and each location has a full address that includes the street name, postal code, city, state or province, and the country code.

In places where the departments and warehouses are located, the company records details such as the country name, currency symbol, currency name, and the region where the country is located geographically.

The HR Entity Relationship Diagram



The Human Resources (HR) Table Descriptions

DESCRIBE countries

Name	Null	Type

COUNTRY_ID	NOT NULL	CHAR(2)
COUNTRY_NAME		VARCHAR2(40)
REGION_ID		NUMBER

SELECT * FROM countries;





	COUNTRY_ID	COUNTRY_NAME	REGION_ID
1	CA	Canada	2
2	DE	Germany	1
3	UK	United Kingdom	1
4	US	United States of America	2

The Human Resources (HR) Table Descriptions (continued)

DESCRIBE departments

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

SELECT * FROM departments;

	 DEPARTMENT_ID	 DEPARTMENT_NAME	 MANAGER_ID	 LOCATION_ID
1	10	Administration	200	1700
2	20	Marketing	201	1800
3	50	Shipping	124	1500
4	60	IT	103	1400
5	80	Sales	149	2500
6	90	Executive	100	1700
7	110	Accounting	205	1700
8	190	Contracting	(null)	1700

The Human Resources (HR) Table Descriptions (continued)

DESCRIBE employees

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

SELECT * FROM employees;

	EMPLOYEE_ID	FIRST_N...	LAST_N...	EMAIL	PHONE_NUMBER	HIRE_DATE	JOB_ID	SALARY	COMMI...	MANAGER_ID	DEPARTMENT_ID
1	100	Steven	King	SKING	515.123.4567	17-JUN-87	AD_PRES	24000	(null)	(null)	90
2	101	Neena	Kochhar	NKOCHHAR	515.123.4568	21-SEP-89	AD_VP	17000	(null)	100	90
3	102	Lex	De Haan	LDEHAAN	515.123.4569	13-JAN-93	AD_VP	17000	(null)	100	90
4	103	Alexander	Hunold	AHUNOLD	590.423.4567	03-JAN-90	IT_PROG	9000	(null)	102	60
5	104	Bruce	Ernst	BERNST	590.423.4568	21-MAY-91	IT_PROG	6000	(null)	103	60
6	107	Diana	Lorentz	DLORENTZ	590.423.5567	07-FEB-99	IT_PROG	4200	(null)	103	60
7	124	Kevin	Mourgos	KMOURGOS	650.123.5234	16-NOV-99	ST_MAN	5800	(null)	100	50
8	141	Trenna	Rajs	TRAJS	650.121.8009	17-OCT-95	ST_CLERK	3500	(null)	124	50
9	142	Curtis	Davies	CDAVIES	650.121.2994	29-JAN-97	ST_CLERK	3100	(null)	124	50
10	143	Randall	Matos	RMATOS	650.121.2874	15-MAR-98	ST_CLERK	2600	(null)	124	50
11	144	Peter	Vargas	PVARGAS	650.121.2004	09-JUL-98	ST_CLERK	2500	(null)	124	50
12	149	Eleni	Zlotkey	EZLOTKEY	011.44.1344.429018	29-JAN-00	SA_MAN	10500	0.2	100	80
13	174	Ellen	Abel	EABEL	011.44.1644.429267	11-MAY-96	SA_REP	11000	0.3	149	80
14	176	Jonathon	Taylor	JTAYLOR	011.44.1644.429265	24-MAR-98	SA_REP	8600	0.2	149	80
15	178	Kimberely	Grant	KGRANT	011.44.1644.429263	24-MAY-99	SA_REP	7000	0.15	149	(null)
16	200	Jennifer	Whalen	JWHALEN	515.123.4444	17-SEP-87	AD_ASST	4400	(null)	101	10
17	201	Michael	Hartstein	MHARTSTE	515.123.5555	17-FEB-96	MK_MAN	13000	(null)	100	20
18	202	Pat	Fay	PFAY	603.123.6666	17-AUG-97	MK_REP	6000	(null)	201	20
19	205	Shelley	Higgins	SHIGGINS	515.123.8080	07-JUN-94	AC_MGR	12000	(null)	101	110
20	206	William	Gietz	WGIEZT	515.123.8181	07-JUN-94	AC_ACC...	8300	(null)	205	110

The Human Resources (HR) Table Descriptions (continued)

DESCRIBE job_history

DESCRIBE job_history		
Name	Null	Type

EMPLOYEE_ID	NOT NULL	NUMBER(6)
START_DATE	NOT NULL	DATE
END_DATE	NOT NULL	DATE
JOB_ID	NOT NULL	VARCHAR2(10)
DEPARTMENT_ID		NUMBER(4)

SELECT * FROM job_history





	EMPLOYEE_ID	START_DATE	END_DATE	JOB_ID	DEPARTMENT_ID
1	102	13-JAN-93	24-JUL-98	IT_PROG	60
2	101	21-SEP-89	27-OCT-93	AC_ACCOUNT	110
3	101	28-OCT-93	15-MAR-97	AC_MGR	110
4	201	17-FEB-96	19-DEC-99	MK_REP	20
5	114	24-MAR-98	31-DEC-99	ST_CLERK	50
6	122	01-JAN-99	31-DEC-99	ST_CLERK	50
7	200	17-SEP-87	17-JUN-93	AD_ASST	90
8	176	24-MAR-98	31-DEC-98	SA_REP	80
9	176	01-JAN-99	31-DEC-99	SA_MAN	80
10	200	01-JUL-94	31-DEC-98	AC_ACCOUNT	90

The Human Resources (HR) Table Descriptions (continued)

DESCRIBE jobs

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

SELECT * FROM jobs






	 JOB_ID	 JOB_TITLE	 MIN_SALARY	 MAX_SALARY
1	AD_PRES	President	20000	40000
2	AD_VP	Administration Vice President	15000	30000
3	AD_ASST	Administration Assistant	3000	6000
4	AC_MGR	Accounting Manager	8200	16000
5	AC_ACCOUNT	Public Accountant	4200	9000
6	SA_MAN	Sales Manager	10000	20000
7	SA_REP	Sales Representative	6000	12000
8	ST_MAN	Stock Manager	5500	8500
9	ST_CLERK	Stock Clerk	2000	5000
10	IT_PROG	Programmer	4000	10000
11	MK_MAN	Marketing Manager	9000	15000
12	MK_REP	Marketing Representative	4000	9000

The Human Resources (HR) Table Descriptions (continued)

DESCRIBE locations

Name	Null	Type
LOCATION_ID	NOT NULL	NUMBER(4)
STREET_ADDRESS		VARCHAR2(40)
POSTAL_CODE		VARCHAR2(12)
CITY	NOT NULL	VARCHAR2(30)
STATE_PROVINCE		VARCHAR2(25)
COUNTRY_ID		CHAR(2)

SELECT * FROM locations



	 LOCATION_ID	 STREET_ADDRESS	 POSTAL_CODE	 CITY	 STATE_PROVINCE	COUNTRY_ID
1	1400	2014 Jabberwocky Rd	26192	Southlake	Texas	US
2	1500	2011 Interiors Blvd	99236	South San Francisco	California	US
3	1700	2004 Charade Rd	98199	Seattle	Washington	US
4	1800	460 Bloor St. W.	ON M5S 1X8	Toronto	Ontario	CA
5	2500	Magdalen Centre, The Oxford Science Park	OX9 9ZB	Oxford	Oxford	UK

The Human Resources (HR) Table Descriptions (continued)

DESCRIBE regions

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

SELECT * FROM regions

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

Using SQL Developer

ORACLE

Copyright © 2009, Oracle. All rights reserved.

Objectives

After completing this appendix, you should be able to do the following:

- List the key features of Oracle SQL Developer
- Identify menu items of Oracle SQL Developer
- Create a database connection
- Manage database objects
- Use SQL Worksheet
- Save and run SQL scripts
- Create and save reports

ORACLE

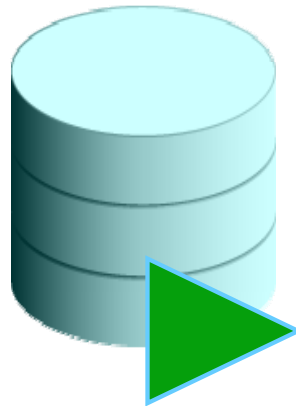
Copyright © 2009, Oracle. All rights reserved.

Objectives

In this appendix, you are introduced to the graphical tool called SQL Developer. You learn how to use SQL Developer for your database development tasks. You learn how to use SQL Worksheet to execute SQL statements and SQL scripts.

What Is Oracle SQL Developer?

- Oracle SQL Developer is a graphical tool that enhances productivity and simplifies database development tasks.
- You can connect to any target Oracle database schema by using standard Oracle database authentication.



SQL Developer

ORACLE

Copyright © 2009, Oracle. All rights reserved.

What Is Oracle SQL Developer?

Oracle SQL Developer is a free graphical tool designed to improve your productivity and simplify the development of everyday database tasks. With just a few clicks, you can easily create and debug stored procedures, test SQL statements, and view optimizer plans.

SQL Developer, the visual tool for database development, simplifies the following tasks:

- Browsing and managing database objects
- Executing SQL statements and scripts
- Editing and debugging PL/SQL statements
- Creating reports

You can connect to any target Oracle database schema by using standard Oracle database authentication. When connected, you can perform operations on objects in the database.

The SQL Developer 1.2 release tightly integrates with *Developer Migration Workbench* that provides users with a single point to browse database objects and data in third-party databases, and to migrate from these databases to Oracle. You can also connect to schemas for selected third-party (non-Oracle) databases such as MySQL, Microsoft SQL Server, and Microsoft Access, and you can view metadata and data in these databases.

Additionally, SQL Developer includes support for Oracle Application Express 3.0.1 (Oracle APEX).

Specifications of SQL Developer

- Shipped along with Oracle Database 11g Release 2
- Developed in Java
- Supports Windows, Linux, and Mac OS X platforms
- Default connectivity by using the Java Database Connectivity (JDBC) thin driver
- Connects to Oracle Database version 9.2.0.1 and later
- Freely downloadable from the following link:
 - http://www.oracle.com/technology/products/database/sql_developer/index.html

ORACLE

Copyright © 2009, Oracle. All rights reserved.

Specifications of SQL Developer

Oracle SQL Developer 1.5 is shipped along with Oracle Database 11g Release 2. SQL Developer is developed in Java leveraging the Oracle JDeveloper integrated development environment (IDE). Therefore, it is a cross-platform tool. The tool runs on Windows, Linux, and Mac operating system (OS) X platforms.

Default connectivity to the database is through the JDBC thin driver, and therefore, no Oracle Home is required. SQL Developer does not require an installer and you need to simply unzip the downloaded file. With SQL Developer, users can connect to Oracle Databases 9.2.0.1 and later, and all Oracle database editions including Express Edition.

Note

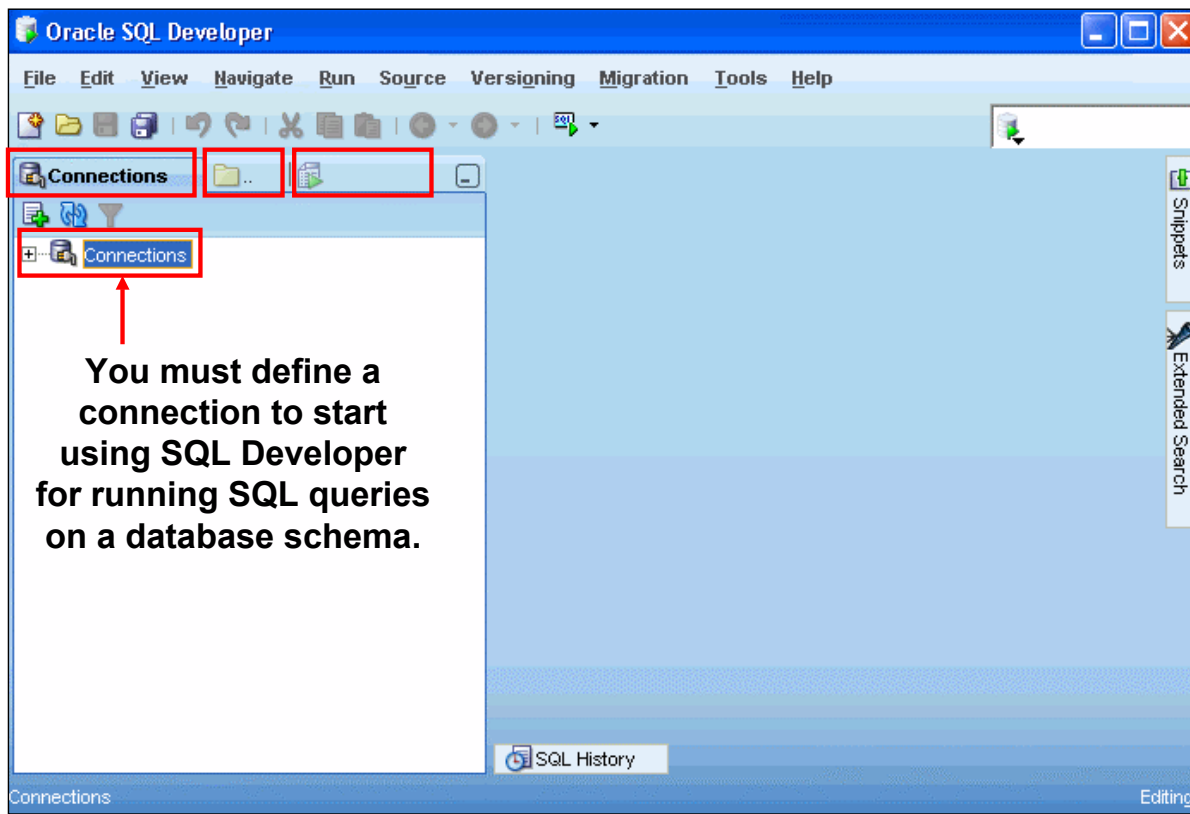
For Oracle Database versions earlier than Oracle Database 11g Release 2, you will have to download and install SQL Developer. SQL Developer 1.5 is freely downloadable from the following link:

http://www.oracle.com/technology/products/database/sql_developer/index.html.

For instructions on how to install SQL Developer, you can visit the following link:

http://download.oracle.com/docs/cd/E12151_01/index.htm

SQL Developer 1.5 Interface



ORACLE

Copyright © 2009, Oracle. All rights reserved.

SQL Developer 1.5 Interface

The SQL Developer 1.5 interface contains three main navigation tabs, from left to right:

- **Connections tab:** By using this tab, you can browse database objects and users to which you have access.
- **Files tab:** Identified by the Files folder icon, this tab enables you to access files from your local machine without having to use the File > Open menu.
- **Reports tab:** Identified by the Reports icon, this tab enables you to run predefined reports or create and add your own reports.

General Navigation and Use

SQL Developer uses the left side for navigation to find and select objects, and the right side to display information about selected objects. You can customize many aspects of the appearance and behavior of SQL Developer by setting preferences.

Note: You need to define at least one connection to be able to connect to a database schema and issue SQL queries or run procedures/functions.

SQL Developer 1.5 Interface (continued)

Menus

The following menus contain standard entries, plus entries for features specific to SQL Developer:

- **View:** Contains options that affect what is displayed in the SQL Developer interface
- **Navigate:** Contains options for navigating to various panes and for executing subprograms
- **Run:** Contains the Run File and Execution Profile options that are relevant when a function or procedure is selected, and also debugging options
- **Source:** Contains options for use when you edit functions and procedures
- **Versioning:** Provides integrated support for the following versioning and source control systems: Concurrent Versions System (CVS) and Subversion
- **Migration:** Contains options related to migrating third-party databases to Oracle
- **Tools:** Invokes SQL Developer tools such as SQL*Plus, Preferences, and SQL Worksheet

Note: The Run menu also contains options that are relevant when a function or procedure is selected for debugging. These are the same options that are found in the Debug menu in version 1.2.

Creating a Database Connection

- You must have at least one database connection to use SQL Developer.
- You can create and test connections for multiple:
 - Databases
 - Schemas
- SQL Developer automatically imports any connections defined in the `tnsnames.ora` file on your system.
- You can export connections to an Extensible Markup Language (XML) file.
- Each additional database connection created is listed in the Connections Navigator hierarchy.

ORACLE

Copyright © 2009, Oracle. All rights reserved.

Creating a Database Connection

A connection is a SQL Developer object that specifies the necessary information for connecting to a specific database as a specific user of that database. To use SQL Developer, you must have at least one database connection, which may be existing, created, or imported.

You can create and test connections for multiple databases and for multiple schemas.

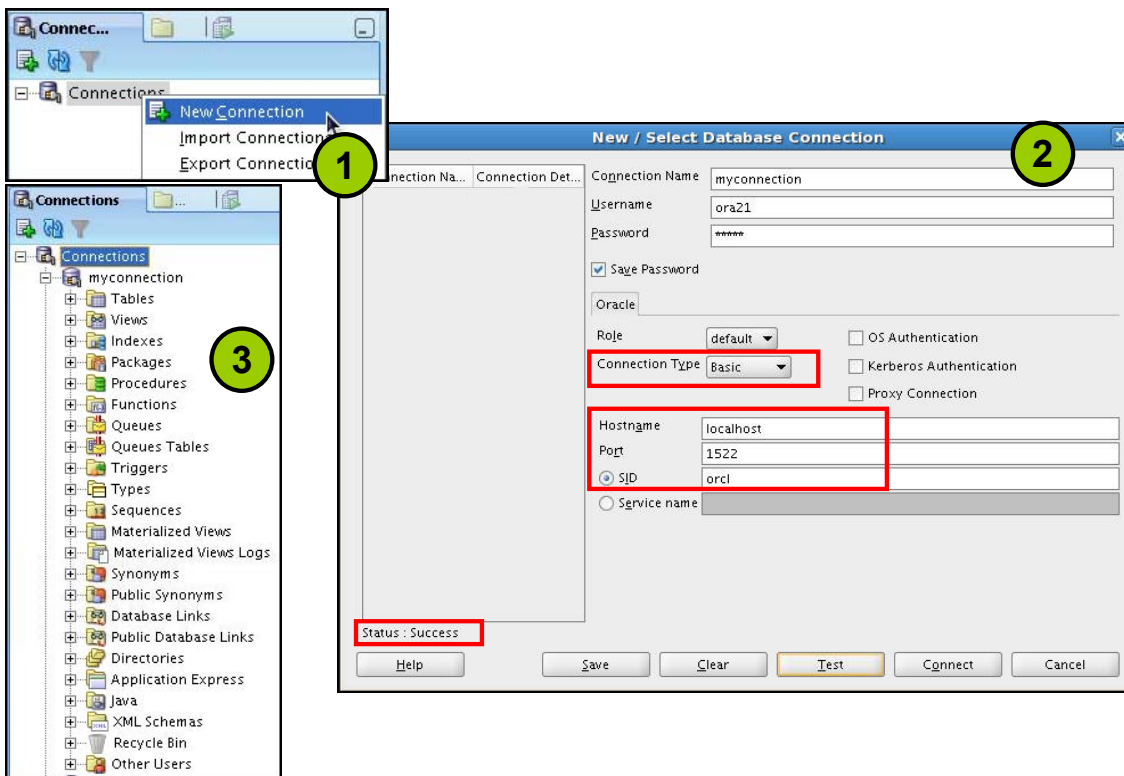
By default, the `tnsnames.ora` file is located in the `$ORACLE_HOME/network/admin` directory, but it can also be in the directory specified by the `TNS_ADMIN` environment variable or registry value. When you start SQL Developer and display the Database Connections dialog box, SQL Developer automatically imports any connections defined in the `tnsnames.ora` file on your system.

Note: On Windows, if the `tnsnames.ora` file exists but its connections are not being used by SQL Developer, define `TNS_ADMIN` as a system environment variable.

You can export connections to an XML file so that you can reuse it later.

You can create additional connections as different users to the same database or to connect to the different databases.

Creating a Database Connection



ORACLE

Copyright © 2009, Oracle. All rights reserved.

Creating a Database Connection (continued)

To create a database connection, perform the following steps:

1. On the Connections tabbed page, right-click **Connections** and select **New Connection**.
2. In the New/Select Database Connection window, enter the connection name. Enter the username and password of the schema that you want to connect to.
 - a) From the Role drop-down box, you can select either default or SYSDBA (you choose SYSDBA for the sys user or any user with database administrator privileges).
 - b) You can select the connection type as:
 - **Basic:** In this type, enter hostname and SID for the database you want to connect to. Port is already set to 1521. Or you can also choose to enter the Service name directly if you use a remote database connection.
 - **TNS:** You can select any one of the database aliases imported from the `tnsnames.ora` file.
 - **LDAP:** You can look up database services in Oracle Internet Directory which is a component of Oracle Identity Management.
 - **Advanced:** You can define a custom JDBC URL to connect to the database.
 - c) Click Test to ensure that the connection has been set correctly.
 - d) Click Connect.

Creating a Database Connection (continued)

If you select the Save Password check box, the password is saved to an XML file. So, after you close the SQL Developer connection and open it again, you are not prompted for the password.

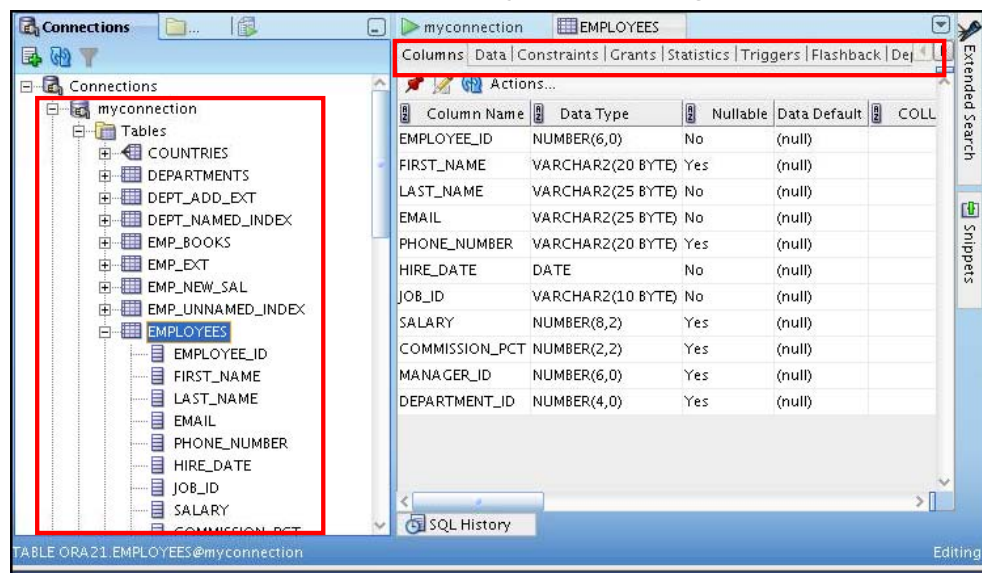
3. The connection gets added in the Connections Navigator. You can expand the connection to view the database objects and view object definitions, for example, dependencies, details, statistics, and so on.

Note: From the same New/Select Database Connection window, you can define connections to non-Oracle data sources using the Access, MySQL, and SQL Server tabs. However, these connections are read-only connections that enable you to browse objects and data in that data source.

Browsing Database Objects

Use the Connections Navigator to to:

- Browse through many objects in a database schema
- Review the definitions of objects at a glance



ORACLE

Copyright © 2009, Oracle. All rights reserved.

Browsing Database Objects

After you create a database connection, you can use the Connections Navigator to browse through many objects in a database schema including Tables, Views, Indexes, Packages, Procedures, Triggers, and Types.

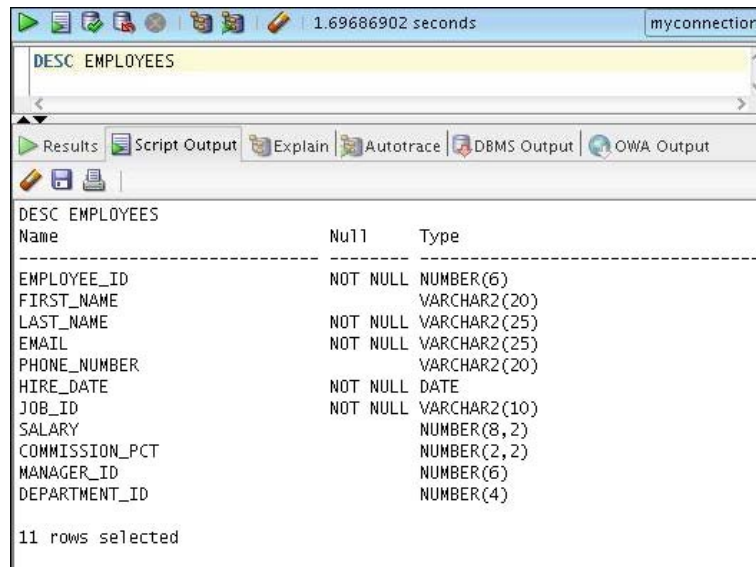
You can see the definition of the objects broken into tabs of information that is pulled out of the data dictionary. For example, if you select a table in the Navigator, the details about columns, constraints, grants, statistics, triggers, and so on are displayed on an easy-to-read tabbed page.

If you want to see the definition of the EMPLOYEES table as shown in the slide, perform the following steps:

1. Expand the Connections node in the Connections Navigator.
2. Expand Tables.
3. Click EMPLOYEES. By default, the Columns tab is selected. It shows the column description of the table. Using the Data tab, you can view the table data and also enter new rows, update data, and commit these changes to the database.

Displaying the Table Structure

Use the DESCRIBE command to display the structure of a table:



The screenshot shows the SQL Developer interface with the command 'DESC EMPLOYEES' entered in the SQL window. The Results tab is active, displaying the table structure. The table has 11 rows selected. The columns and their data types are as follows:

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

ORACLE

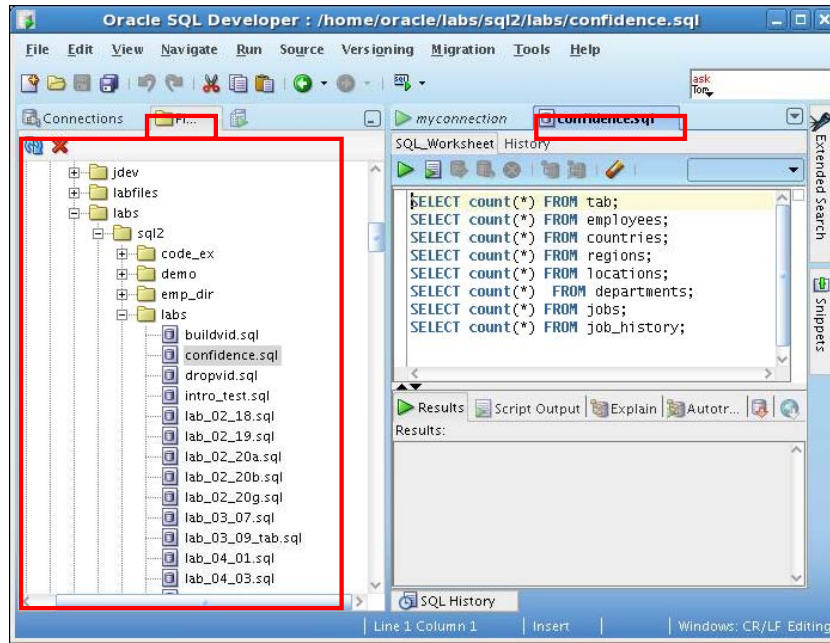
Copyright © 2009, Oracle. All rights reserved.

Displaying the Table Structure

In SQL Developer, you can also display the structure of a table using the DESCRIBE command. The result of the command is a display of column names and data types as well as an indication if a column must contain data.

Browsing Files

Use the File Navigator to explore the file system and open system files.



ORACLE

Copyright © 2009, Oracle. All rights reserved.

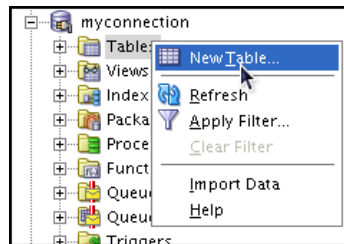
Browsing Database Objects

You can use the File Navigator to browse and open system files.

- To view the files navigator, click the Files tab, or click View > Files.
- To view the contents of a file, double-click a file name to display its contents in the SQL worksheet area.

Creating a Schema Object

- SQL Developer supports the creation of any schema object by:
 - Executing a SQL statement in SQL Worksheet
 - Using the context menu
- Edit the objects by using an edit dialog box or one of the many context-sensitive menus.
- View the data definition language (DDL) for adjustments such as creating a new object or editing an existing schema object.



ORACLE

Copyright © 2009, Oracle. All rights reserved.

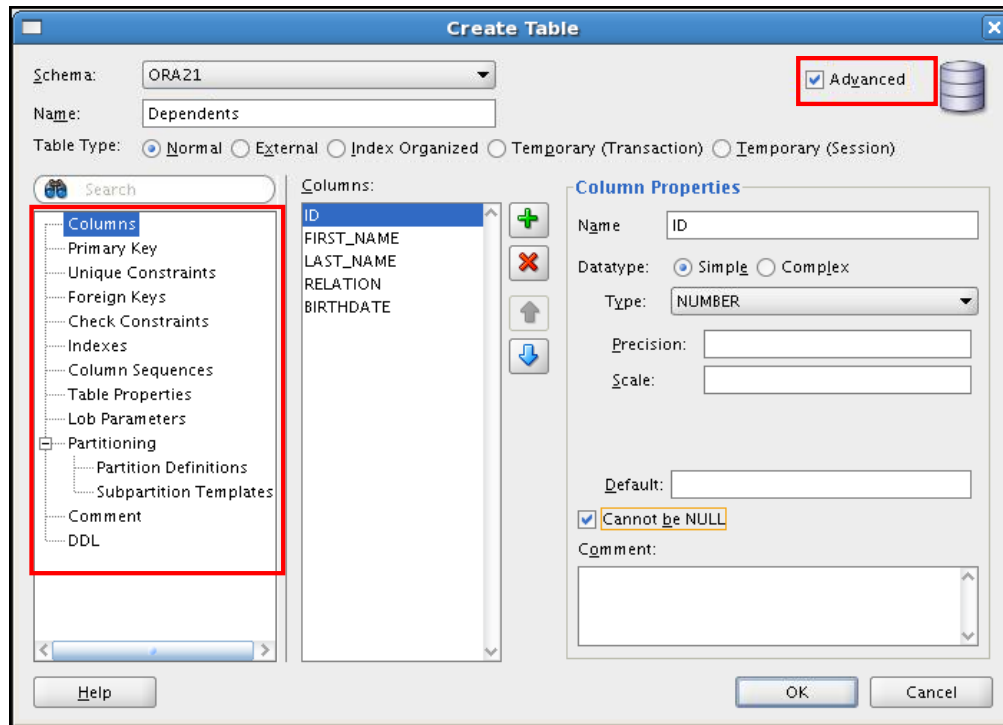
Creating a Schema Object

SQL Developer supports the creation of any schema object by executing a SQL statement in SQL Worksheet. Alternatively, you can create objects using the context menus. When created, you can edit the objects using an edit dialog box or one of the many context-sensitive menus.

As new objects are created or existing objects are edited, the DDL for those adjustments is available for review. An Export DDL option is available if you want to create the full DDL for one or more objects in the schema.

The slide shows how to create a table using the context menu. To open a dialog box for creating a new table, right-click Tables and select New Table. The dialog boxes to create and edit database objects have multiple tabs, each reflecting a logical grouping of properties for that type of object.

Creating a New Table: Example



ORACLE

Copyright © 2009, Oracle. All rights reserved.

Creating a New Table: Example

In the Create Table dialog box, if you do not select the Advanced check box, you can create a table quickly by specifying columns and some frequently used features.

If you select the Advanced check box, the Create Table dialog box changes to one with multiple options, in which you can specify an extended set of features while you create the table.

The example in the slide shows how to create the DEPENDENTS table by selecting the Advanced check box.

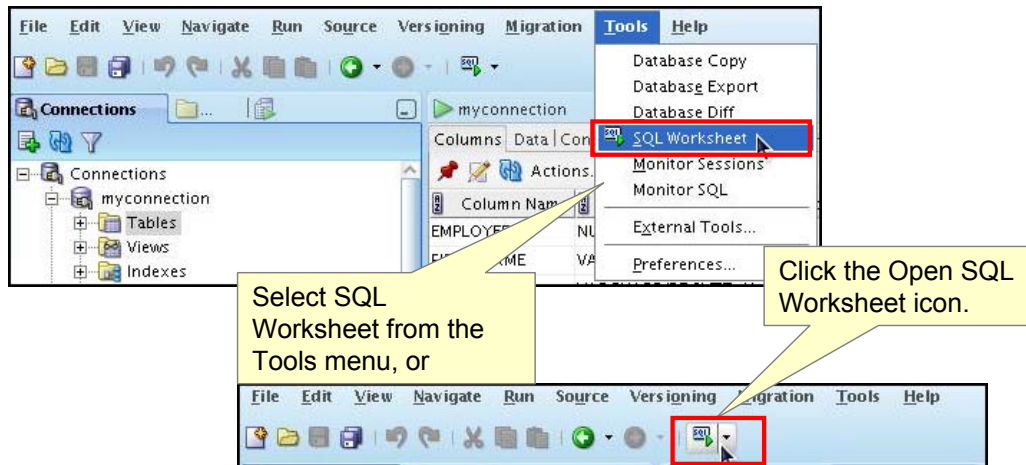
To create a new table, perform the following steps:

1. In the Connections Navigator, right-click Tables.
2. Select Create TABLE.
3. In the Create Table dialog box, select Advanced.
4. Specify column information.
5. Click OK.

Although it is not required, you should also specify a primary key by using the Primary Key tab in the dialog box. Sometimes, you may want to edit the table that you have created; to do so, right-click the table in the Connections Navigator and select Edit.

Using the SQL Worksheet

- Use the SQL Worksheet to enter and execute SQL, PL/SQL, and SQL *Plus statements.
- Specify any actions that can be processed by the database connection associated with the worksheet.



ORACLE

Copyright © 2009, Oracle. All rights reserved.

Using the SQL Worksheet

When you connect to a database, a SQL Worksheet window for that connection automatically opens. You can use the SQL Worksheet to enter and execute SQL, PL/SQL, and SQL*Plus statements. The SQL Worksheet supports SQL*Plus statements to a certain extent. SQL*Plus statements that are not supported by the SQL Worksheet are ignored and not passed to the database.

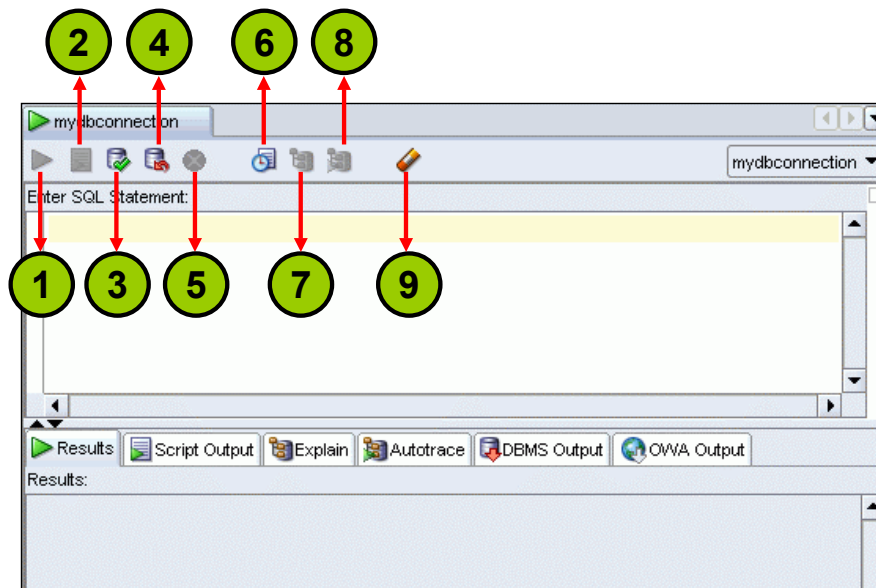
You can specify actions that can be processed by the database connection associated with the worksheet, such as:

- Creating a table
- Inserting data
- Creating and editing a trigger
- Selecting data from a table
- Saving the selected data to a file

You can display a SQL Worksheet by using one of the following:

- Select Tools > SQL Worksheet.
- Click the Open SQL Worksheet icon.

Using the SQL Worksheet



ORACLE

Copyright © 2009, Oracle. All rights reserved.

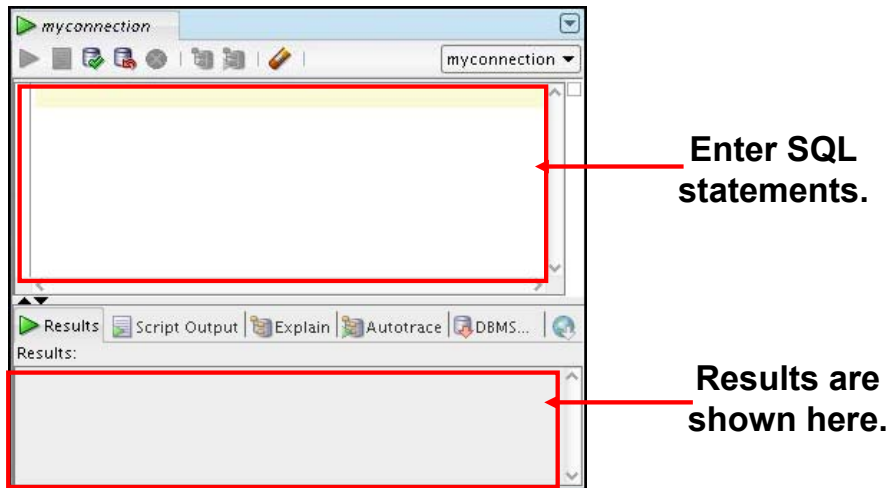
Using the SQL Worksheet (continued)

You may want to use the shortcut keys or icons to perform certain tasks such as executing a SQL statement, running a script, and viewing the history of SQL statements that you have executed. You can use the SQL Worksheet toolbar that contains icons to perform the following tasks:

1. **Execute Statement:** Executes the statement where the cursor is located in the Enter SQL Statement box. You can use bind variables in the SQL statements, but not substitution variables.
2. **Run Script:** Executes all statements in the Enter SQL Statement box by using the Script Runner. You can use substitution variables in the SQL statements, but not bind variables.
3. **Commit:** Writes any changes to the database and ends the transaction
4. **Rollback:** Discards any changes to the database, without writing them to the database, and ends the transaction
5. **Cancel:** Stops the execution of any statements currently being executed
6. **SQL History:** Displays a dialog box with information about SQL statements that you have executed
7. **Execute Explain Plan:** Generates the execution plan, which you can see by clicking the Explain tab
8. **Autotrace:** Generates trace information for the statement
9. **Clear:** Erases the statement or statements in the Enter SQL Statement box

Using the SQL Worksheet

- Use the SQL Worksheet to enter and execute SQL, PL/SQL, and SQL*Plus statements.
- Specify any actions that can be processed by the database connection associated with the worksheet.



ORACLE

Copyright © 2009, Oracle. All rights reserved.

Using the SQL Worksheet (continued)

When you connect to a database, a SQL Worksheet window for that connection automatically opens. You can use the SQL Worksheet to enter and execute SQL, PL/SQL, and SQL*Plus statements. All SQL and PL/SQL commands are supported as they are passed directly from the SQL Worksheet to the Oracle database. SQL*Plus commands used in the SQL Developer have to be interpreted by the SQL Worksheet before being passed to the database.

The SQL Worksheet currently supports a number of SQL*Plus commands. Commands not supported by the SQL Worksheet are ignored and are not sent to the Oracle database. Through the SQL Worksheet, you can execute SQL statements and some of the SQL*Plus commands.

You can display a SQL Worksheet by using any of the following two options:

- Select Tools > SQL Worksheet.
- Click the Open SQL Worksheet icon.

Executing SQL Statements

Use the Enter SQL Statement box to enter single or multiple SQL statements.

The screenshot illustrates the Oracle SQL Developer interface. The top window, titled 'myconnection', shows the 'Enter SQL Statement' box with the query: `SELECT employee_id, last_name FROM employees;`. A red box highlights the 'Execute Statement' button (F9) and the 'Run Script' button (F5). The bottom-left window shows the 'Results' tab with a table of 5 rows. The bottom-right window shows the 'Script Output' tab with a text-based output of 10 rows.

EMPLOYEE_ID	LAST_NAME
1	100 King
2	101 Kochhar
3	102 De Haan
4	103 Hunold
5	104 Ernst

EMPLOYEE_ID	LAST_NAME
100	King
101	Kochhar
102	De Haan
103	Hunold
104	Ernst
105	Austin

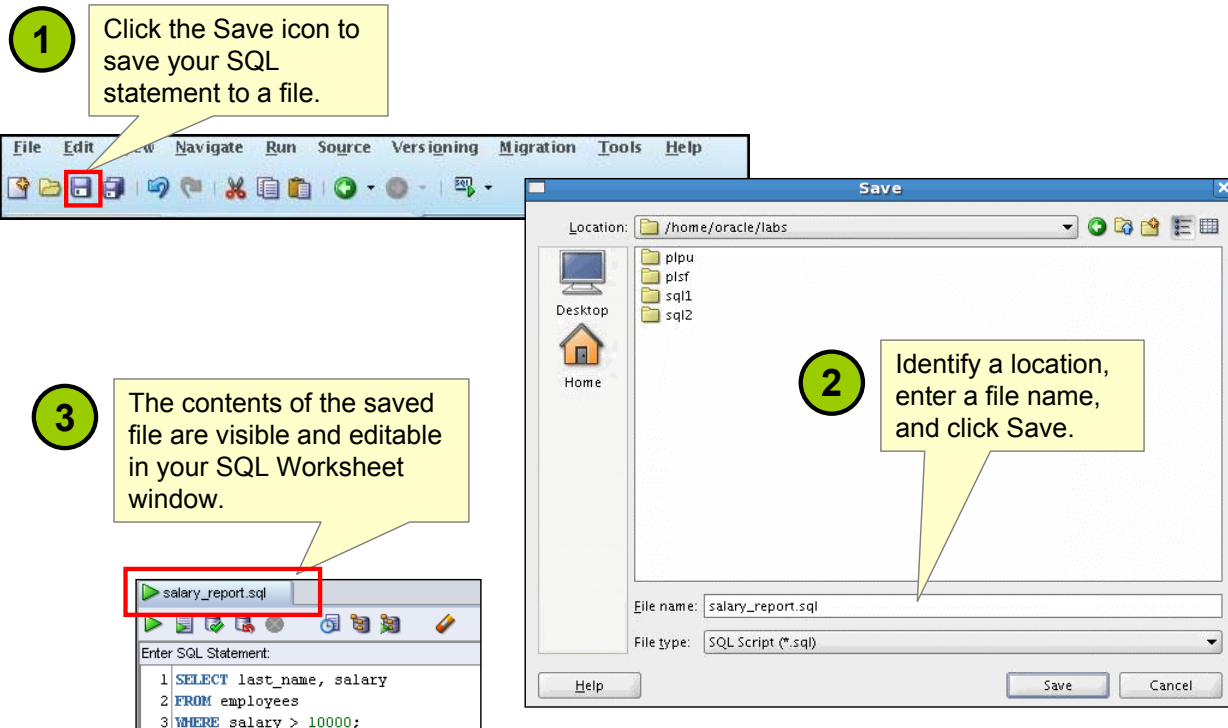
ORACLE

Copyright © 2009, Oracle. All rights reserved.

Executing SQL Statements

The example in the slide shows the difference in output for the same query when the F9 key or Execute Statement is used versus the output when F5 or Run Script is used.

Saving SQL Scripts



ORACLE

Copyright © 2009, Oracle. All rights reserved.

Saving SQL Scripts

You can save your SQL statements from the SQL Worksheet into a text file. To save the contents of the Enter SQL Statement box, follow these steps:

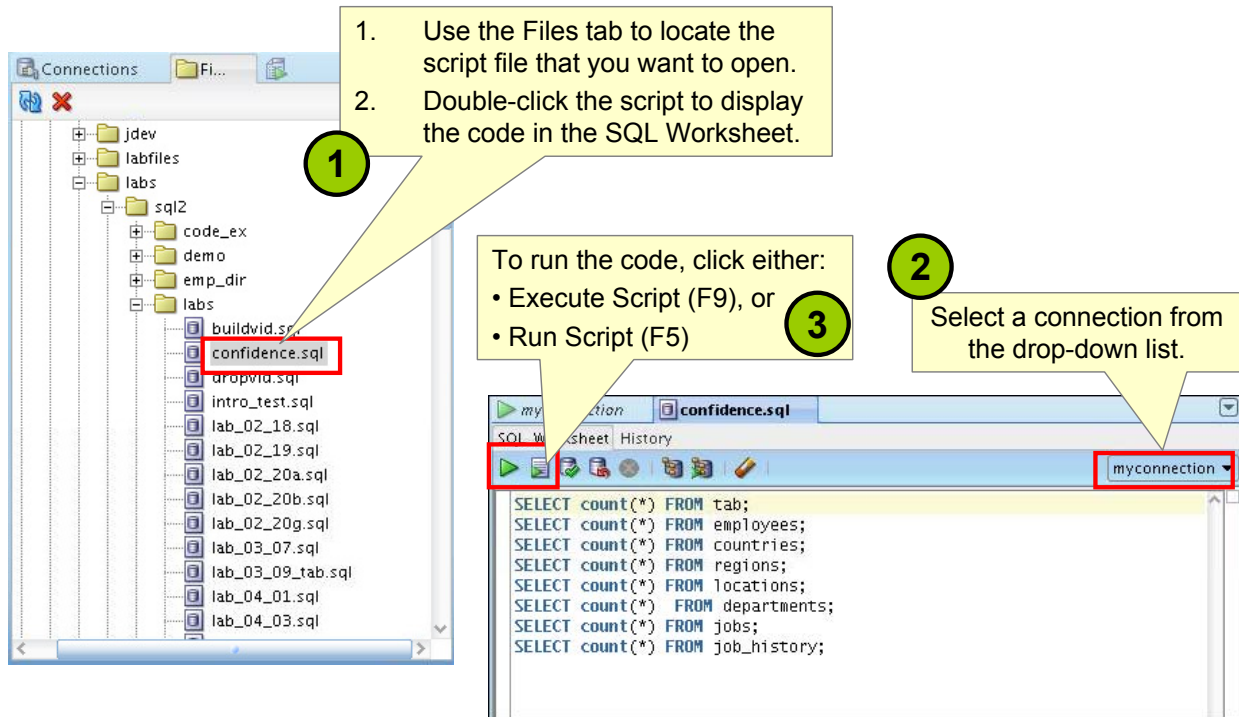
1. Click the Save icon or use the File > Save menu item.
2. In the Windows Save dialog box, enter a file name and the location where you want the file saved.
3. Click Save.

After you save the contents to a file, the Enter SQL Statement window displays a tabbed page of your file contents. You can have multiple files open at the same time. Each file displays as a tabbed page.

Script Pathing

You can select a default path to look for scripts and to save scripts. Under Tools > Preferences > Database > Worksheet Parameters, enter a value in the “Select default path to look for scripts” field.

Executing Saved Script Files: Method 1



ORACLE

Copyright © 2009, Oracle. All rights reserved.

Executing Saved Script Files: Method 1

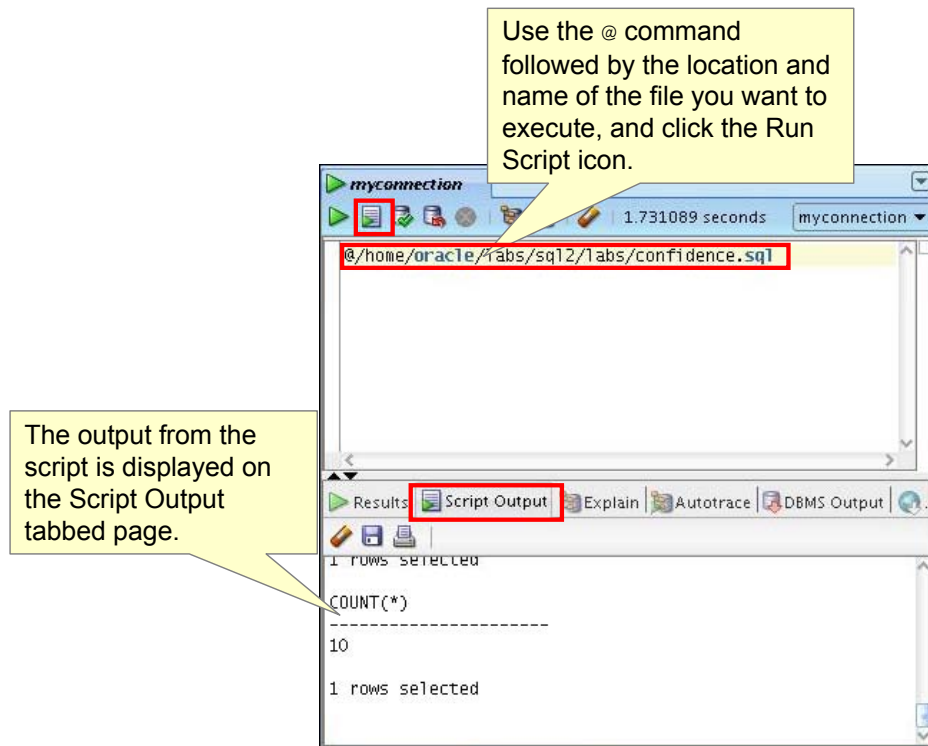
To open a script file and display the code in the SQL Worksheet area, perform the following:

1. In the files navigator select (or navigate to) the script file that you want to open.
2. Double-click to open. The code of the script file is displayed in the SQL Worksheet area.
3. Select a connection from the connection drop-down list.
4. To run the code, click the Run Script (F5) icon on the SQL Worksheet toolbar. If you have not selected a connection from the connection drop-down list, a connection dialog box will appear. Select the connection you want to use for the script execution.

Alternatively, you can also:

1. Select File > Open. The Open dialog box is displayed.
2. In the Open dialog box, select (or navigate to) the script file that you want to open.
3. Click Open. The code of the script file is displayed in the SQL Worksheet area.
4. Select a connection from the connection drop-down list.
5. To run the code, click the Run Script (F5) icon on the SQL Worksheet toolbar. If you have not selected a connection from the connection drop-down list, a connection dialog box will appear. Select the connection you want to use for the script execution.

Executing Saved Script Files: Method 2



ORACLE

Copyright © 2009, Oracle. All rights reserved.

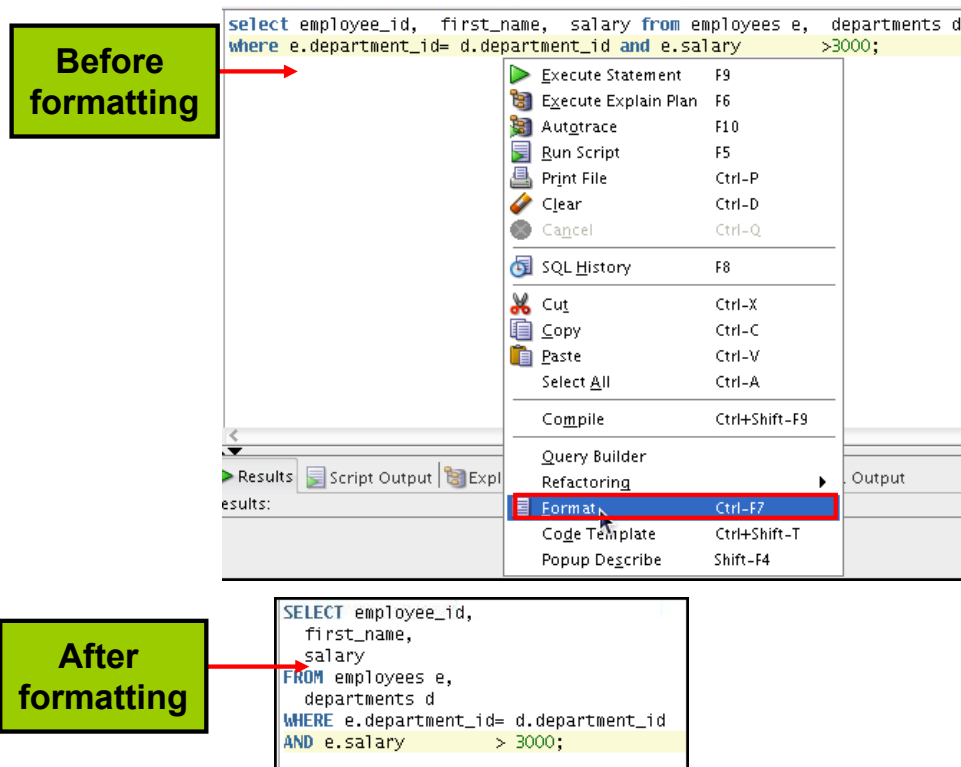
Executing Saved Script Files: Method 2

To run a saved SQL script, perform the following:

1. Use the @ command, followed by the location, and name of the file you want to run, in the Enter SQL Statement window.
2. Click the Run Script icon.

The results from running the file are displayed on the Script Output tabbed page. You can also save the script output by clicking the Save icon on the Script Output tabbed page. The Windows Save dialog box appears and you can identify a name and location for your file.

Formatting the SQL Code



ORACLE

Copyright © 2009, Oracle. All rights reserved.

Formatting the SQL Code

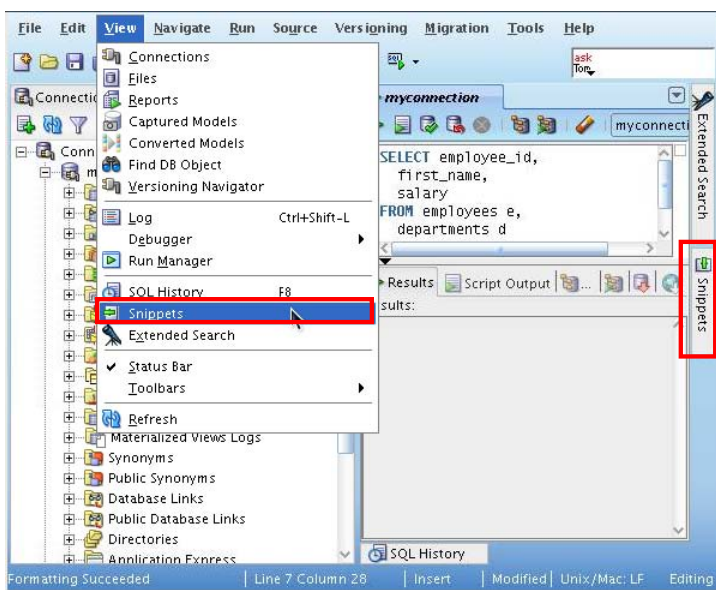
You may want to beautify the indentation, spacing, capitalization, and line separation of the SQL code. SQL Developer has a feature for formatting SQL code.

To format the SQL code, right-click in the statement area and select Format SQL.

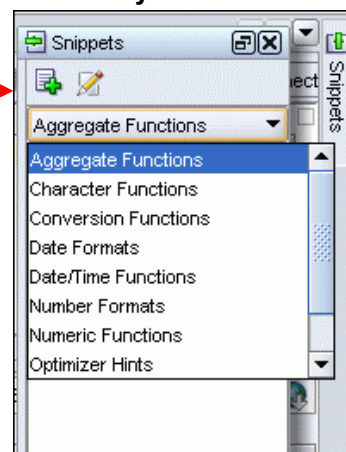
In the example in the slide, before formatting, the SQL code has the keywords not capitalized and the statement not properly indented. After formatting, the SQL code is beautified with the keywords capitalized and the statement properly indented.

Using Snippets

Snippets are code fragments that may be just syntax or examples.



When you place your cursor here, it shows the Snippets window. From the drop-down list, you can select the functions category that you want.



ORACLE

Copyright © 2009, Oracle. All rights reserved.

Using Snippets

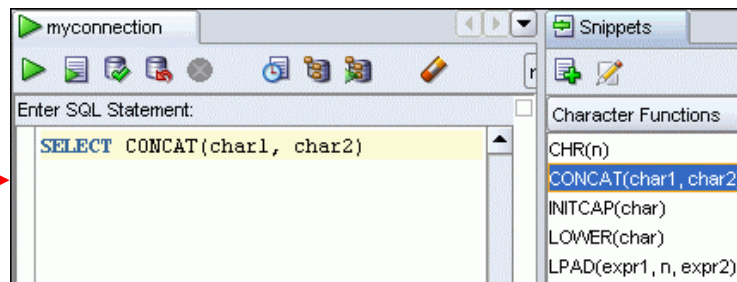
You may want to use certain code fragments when you use the SQL Worksheet or create or edit a PL/SQL function or procedure. SQL Developer has the feature called Snippets. Snippets are code fragments such as SQL functions, Optimizer hints, and miscellaneous PL/SQL programming techniques. You can drag snippets into the Editor window.

To display Snippets, select View > Snippets.

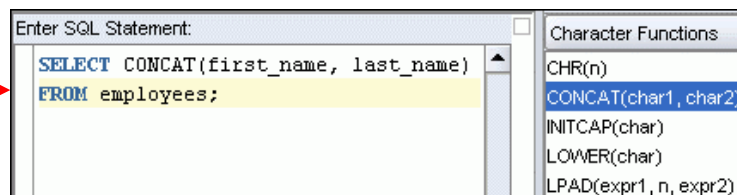
The Snippets window is displayed at the right side. You can use the drop-down list to select a group. A Snippets button is placed in the right window margin, so that you can display the Snippets window if it becomes hidden.

Using Snippets: Example

Inserting a snippet



Editing the snippet



ORACLE

Copyright © 2009, Oracle. All rights reserved.

Using Snippets: Example

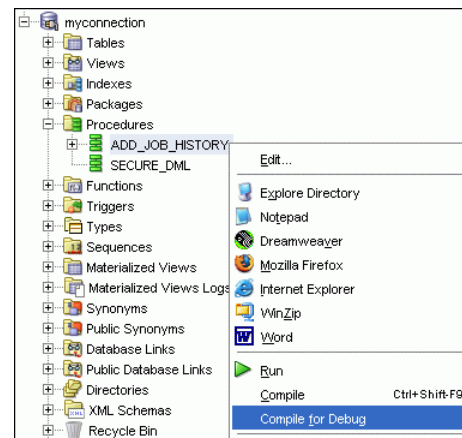
To insert a Snippet into your code in a SQL Worksheet or in a PL/SQL function or procedure, drag the snippet from the Snippets window into the desired place in your code. Then you can edit the syntax so that the SQL function is valid in the current context. To see a brief description of a SQL function in a tool tip, place the cursor over the function name.

The example in the slide shows that `CONCAT(char1, char2)` is dragged from the Character Functions group in the Snippets window. Then the `CONCAT` function syntax is edited and the rest of the statement is added as in the following:

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


Debugging Procedures and Functions

- Use SQL Developer to debug PL/SQL functions and procedures.
- Use the “Compile for Debug” option to perform a PL/SQL compilation so that the procedure can be debugged.
- Use Debug menu options to set breakpoints, and to perform step into and step over tasks.



ORACLE

Copyright © 2009, Oracle. All rights reserved.

Debugging Procedures and Functions

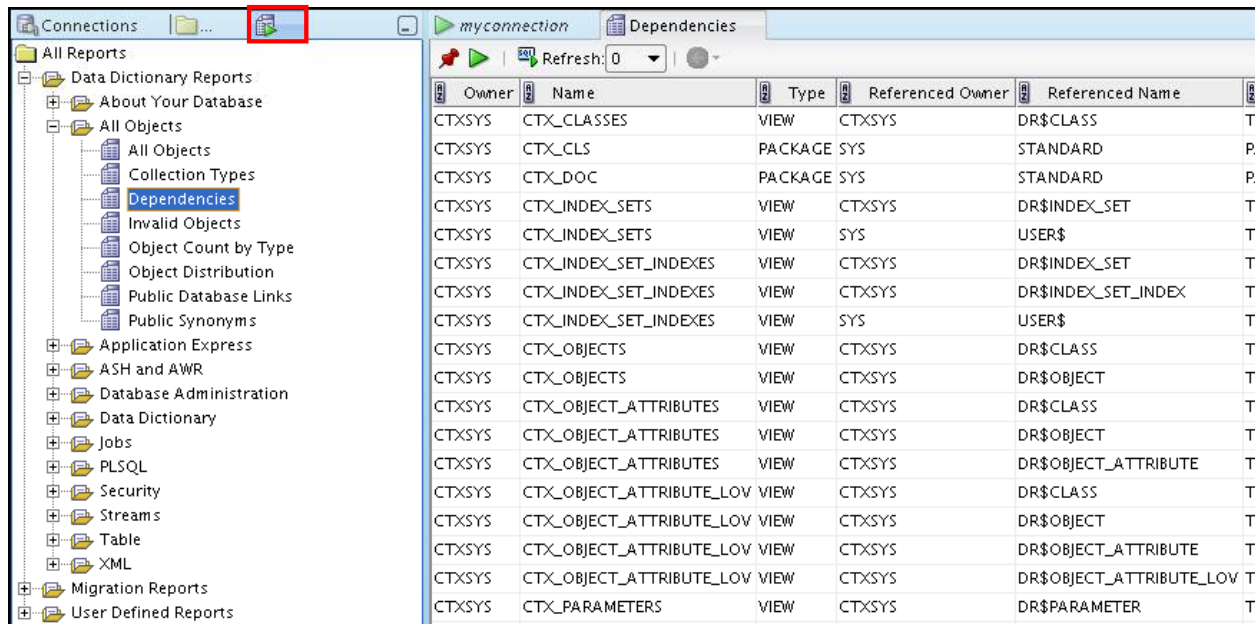
In SQL Developer, you can debug PL/SQL procedures and functions. Using the Debug menu options, you can perform the following debugging tasks:

- **Find Execution Point** goes to the next execution point.
- **Resume** continues execution.
- **Step Over** bypasses the next method and goes to the next statement after the method.
- **Step Into** goes to the first statement in the next method.
- **Step Out** leaves the current method and goes to the next statement.
- **Step to End of Method** goes to the last statement of the current method.
- **Pause** halts execution but does not exit, thus allowing you to resume execution.
- **Terminate** halts and exits the execution. You cannot resume execution from this point; instead, to start running or debugging from the beginning of the function or procedure, click the Run or Debug icon on the Source tab toolbar.
- **Garbage Collection** removes invalid objects from the cache in favor of more frequently accessed and more valid objects.

These options are also available as icons on the debugging toolbar.

Database Reporting

SQL Developer provides a number of predefined reports about the database and its objects.



Owner	Name	Type	Referenced Owner	Referenced Name
CTXSYS	CTX_CLASSES	VIEW	CTXSYS	DR\$CLASS
CTXSYS	CTX_CLS	PACKAGE	SYS	STANDARD
CTXSYS	CTX_DOC	PACKAGE	SYS	STANDARD
CTXSYS	CTX_INDEX_SETS	VIEW	CTXSYS	DR\$INDEX_SET
CTXSYS	CTX_INDEX_SETS	VIEW	SYS	USER\$
CTXSYS	CTX_INDEX_SET_INDEXES	VIEW	CTXSYS	DR\$INDEX_SET
CTXSYS	CTX_INDEX_SET_INDEXES	VIEW	CTXSYS	DR\$INDEX_SET_INDEX
CTXSYS	CTX_INDEX_SET_INDEXES	VIEW	SYS	USER\$
CTXSYS	CTX_OBJECTS	VIEW	CTXSYS	DR\$CLASS
CTXSYS	CTX_OBJECTS	VIEW	CTXSYS	DR\$OBJECT
CTXSYS	CTX_OBJECT_ATTRIBUTES	VIEW	CTXSYS	DR\$CLASS
CTXSYS	CTX_OBJECT_ATTRIBUTES	VIEW	CTXSYS	DR\$OBJECT
CTXSYS	CTX_OBJECT_ATTRIBUTES	VIEW	CTXSYS	DR\$OBJECT_ATTRIBUTE
CTXSYS	CTX_OBJECT_ATTRIBUTE_LOV	VIEW	CTXSYS	DR\$CLASS
CTXSYS	CTX_OBJECT_ATTRIBUTE_LOV	VIEW	CTXSYS	DR\$OBJECT
CTXSYS	CTX_OBJECT_ATTRIBUTE_LOV	VIEW	CTXSYS	DR\$OBJECT_ATTRIBUTE
CTXSYS	CTX_OBJECT_ATTRIBUTE_LOV	VIEW	CTXSYS	DR\$OBJECT_ATTRIBUTE_LOV
CTXSYS	CTX_PARAMETERS	VIEW	CTXSYS	DR\$PARAMETER

ORACLE

Copyright © 2009, Oracle. All rights reserved.

Database Reporting

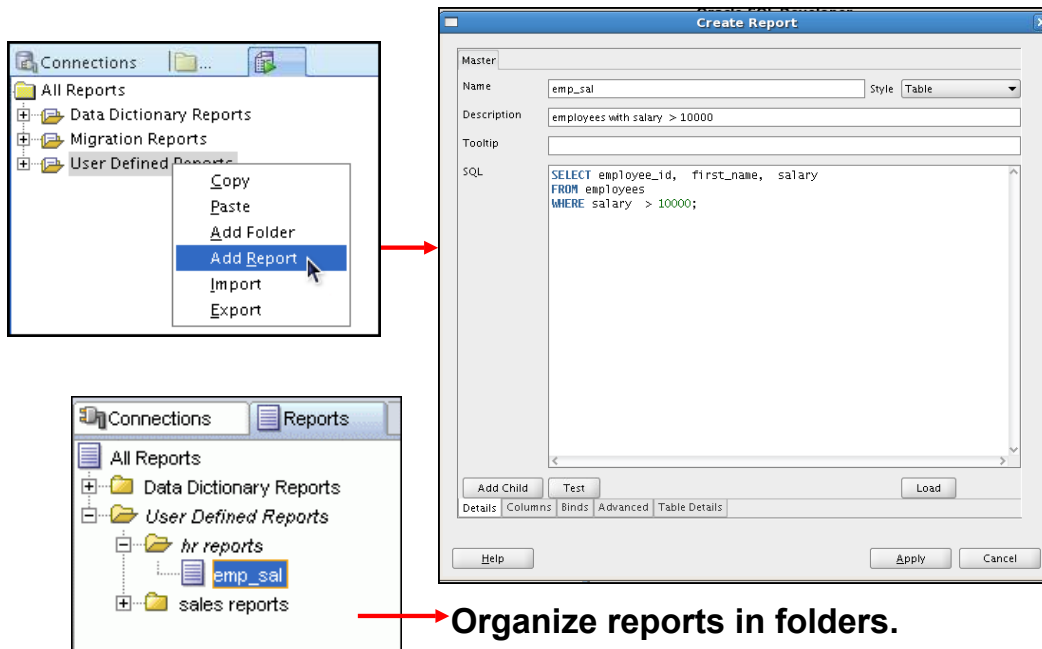
SQL Developer provides many reports about the database and its objects. These reports can be grouped into the following categories:

- About Your Database reports
- Database Administration reports
- Table reports
- PL/SQL reports
- Security reports
- XML reports
- Jobs reports
- Streams reports
- All Objects reports
- Data Dictionary reports
- User-Defined reports

To display reports, click the Reports tab at the left side of the window. Individual reports are displayed in tabbed panes at the right side of the window; and for each report, you can select (using a drop-down list) the database connection for which to display the report. For reports about objects, the objects shown are only those visible to the database user associated with the selected database connection, and the rows are usually ordered by Owner. You can also create your own user-defined reports.

Creating a User-Defined Report

Create and save user-defined reports for repeated use.



Organize reports in folders.

ORACLE

Copyright © 2009, Oracle. All rights reserved.

Creating a User-Defined Report

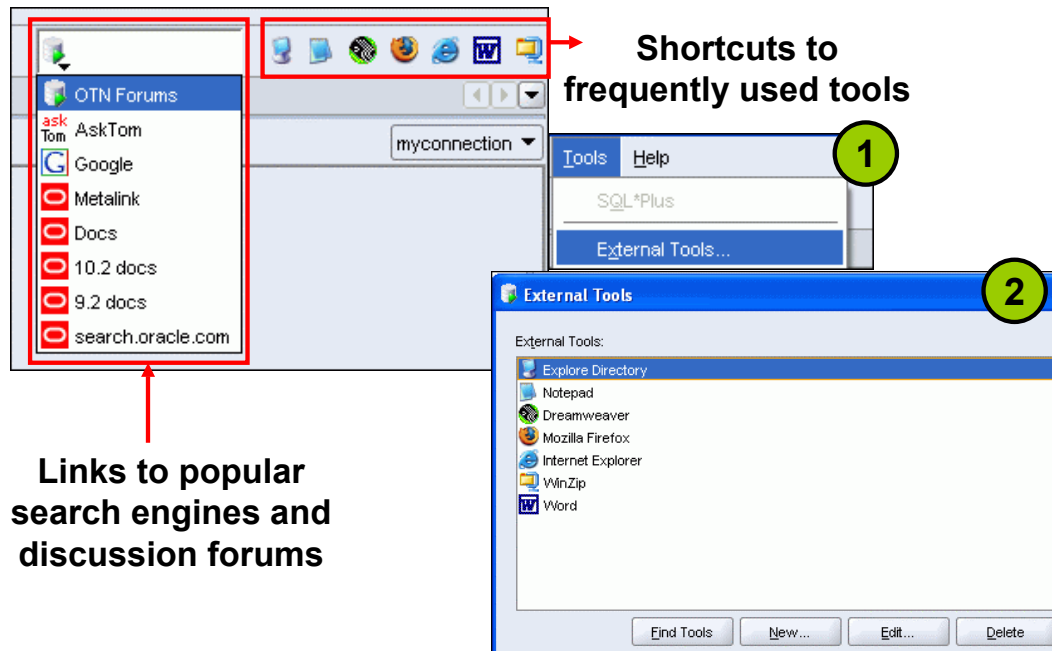
User-defined reports are reports created by SQL Developer users. To create a user-defined report, perform the following steps:

1. Right-click the User Defined Reports node under Reports, and select Add Report.
2. In the Create Report Dialog box, specify the report name and the SQL query to retrieve information for the report. Then, click Apply.

In the example in the slide, the report name is specified as `emp_sal`. An optional description is provided indicating that the report contains details of employees with `salary >= 10000`. The complete SQL statement for retrieving the information to be displayed in the user-defined report is specified in the SQL box. You can also include an optional tool tip to be displayed when the cursor stays briefly over the report name in the Reports navigator display.

You can organize user-defined reports in folders, and you can create a hierarchy of folders and subfolders. To create a folder for user-defined reports, right-click the User Defined Reports node or any folder name under that node and select Add Folder. Information about user-defined reports, including any folders for these reports, is stored in a file named `UserReports.xml` under the directory for user-specific information.

Search Engines and External Tools



Copyright © 2009, Oracle. All rights reserved.

ORACLE

Search Engines and External Tools

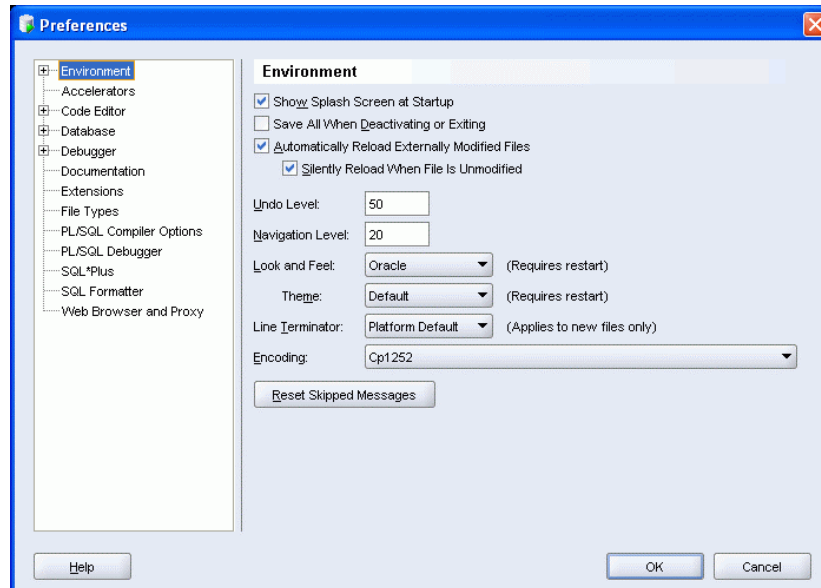
To enhance productivity of the SQL developers, SQL Developer has added quick links to popular search engines and discussion forums such as AskTom, Google, and so on. Also, you have shortcut icons to some of the frequently used tools such as Notepad, Microsoft Word, and Dreamweaver, available to you.

You can add external tools to the existing list or even delete shortcuts to tools that you do not use frequently. To do so, perform the following:

1. From the Tools menu, select External Tools.
2. In the External Tools dialog box, select New to add new tools. Select Delete to remove any tool from the list.

Setting Preferences

- Customize the SQL Developer interface and environment.
- In the Tools menu, select Preferences.



ORACLE

Copyright © 2009, Oracle. All rights reserved.

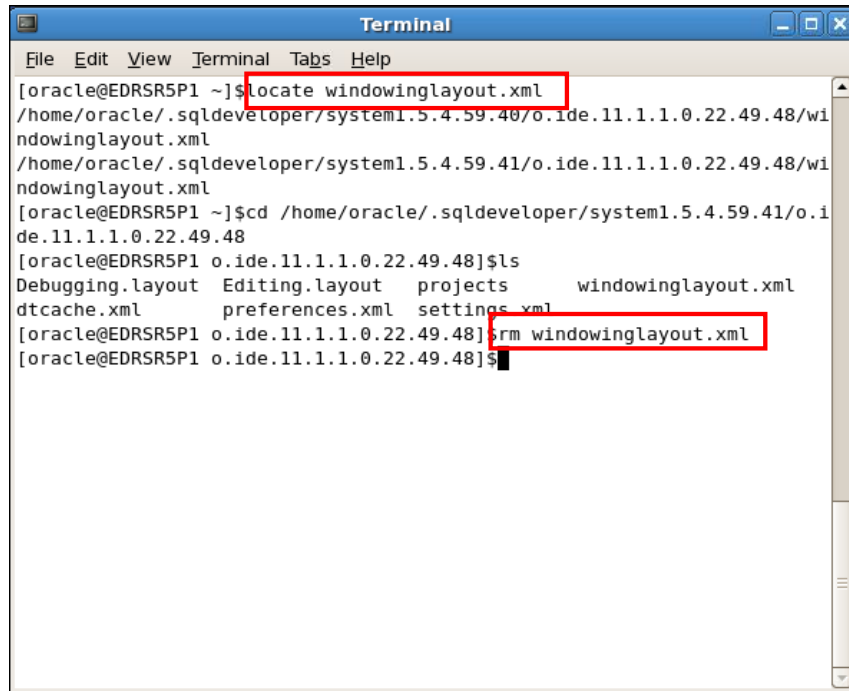
Setting Preferences

You can customize many aspects of the SQL Developer interface and environment by modifying SQL Developer preferences according to your preferences and needs. To modify SQL Developer preferences, select Tools, then Preferences.

The preferences are grouped into the following categories:

- Environment
- Accelerators (keyboard shortcuts)
- Code Editors
- Database
- Debugger
- Documentation
- Extensions
- File Types
- Migration
- PL/SQL Compilers
- PL/SQL Debugger, and so on

Resetting the SQL Developer Layout



```
Terminal
File Edit View Terminal Tabs Help
[oracle@EDRSR5P1 ~]$ locate windowinglayout.xml
/home/oracle/.sqldeveloper/system1.5.4.59.40/o.ide.11.1.1.0.22.49.48/wi
ndowinglayout.xml
/home/oracle/.sqldeveloper/system1.5.4.59.41/o.ide.11.1.1.0.22.49.48/wi
ndowinglayout.xml
[oracle@EDRSR5P1 ~]$ cd /home/oracle/.sqldeveloper/system1.5.4.59.41/o.i
de.11.1.1.0.22.49.48
[oracle@EDRSR5P1 o.ide.11.1.1.0.22.49.48]$ ls
Debugging.layout  Editing.layout  projects        windowinglayout.xml
dtcache.xml      preferences.xml settings.xml
[oracle@EDRSR5P1 o.ide.11.1.1.0.22.49.48]$ rm windowinglayout.xml
[oracle@EDRSR5P1 o.ide.11.1.1.0.22.49.48]$
```

ORACLE

Copyright © 2009, Oracle. All rights reserved.

Resetting the SQL Developer Layout

While working with SQL Developer, if the Connections Navigator disappears or if you cannot dock the Log window in its original place, perform the following steps to fix the problem:

1. Exit from SQL Developer.
2. Open a terminal window and use the `locate` command to find the location of `windowinglayout.xml`.
3. Go to the directory which has `windowinglayout.xml` and delete it.
4. Restart SQL Developer.

Summary

In this appendix, you should have learned how to use SQL Developer to do the following:

- Browse, create, and edit database objects
- Execute SQL statements and scripts in SQL Worksheet
- Create and save custom reports

ORACLE

Copyright © 2009, Oracle. All rights reserved.

Summary

SQL Developer is a free graphical tool to simplify database development tasks. Using SQL Developer, you can browse, create, and edit database objects. You can use SQL Worksheet to run SQL statements and scripts. SQL Developer enables you to create and save your own special set of reports for repeated use.



Using SQL*Plus

ORACLE

Copyright © 2009, Oracle. All rights reserved.

Objectives

After completing this appendix, you should be able to do the following:

- Log in to SQL*Plus
- Edit SQL commands
- Format the output using SQL*Plus commands
- Interact with script files

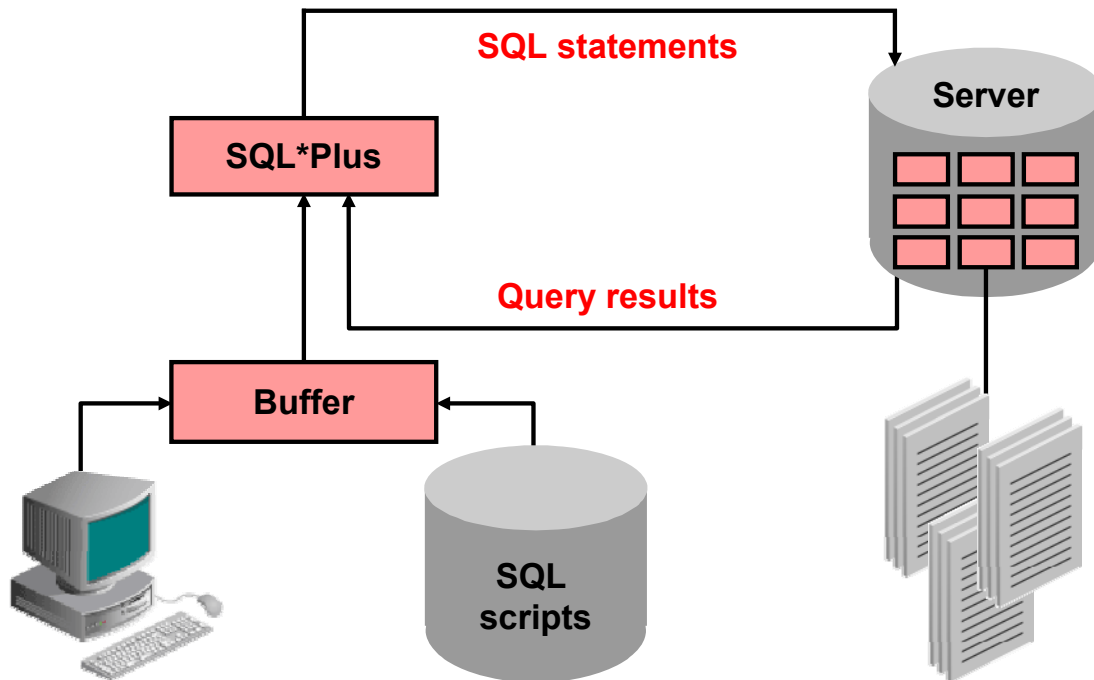
ORACLE

Copyright © 2009, Oracle. All rights reserved.

Objectives

You might want to create `SELECT` statements that can be used again and again. This appendix also covers the use of SQL*Plus commands to execute SQL statements. You learn how to format output using SQL*Plus commands, edit SQL commands, and save scripts in SQL*Plus.

SQL and SQL*Plus Interaction



ORACLE

Copyright © 2009, Oracle. All rights reserved.

SQL and SQL*Plus

SQL is a command language used for communication with the Oracle server from any tool or application. Oracle SQL contains many extensions. When you enter a SQL statement, it is stored in a part of memory called the *SQL buffer* and remains there until you enter a new SQL statement. SQL*Plus is an Oracle tool that recognizes and submits SQL statements to the Oracle9i Server for execution. It contains its own command language.

Features of SQL

- Can be used by a range of users, including those with little or no programming experience
- Is a nonprocedural language
- Reduces the amount of time required for creating and maintaining systems
- Is an English-like language

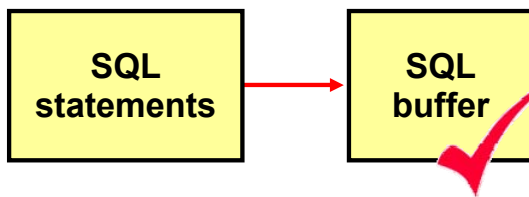
Features of SQL*Plus

- Accepts ad hoc entry of statements
- Accepts SQL input from files
- Provides a line editor for modifying SQL statements
- Controls environmental settings
- Formats query results into basic reports
- Accesses local and remote databases

SQL Statements Versus SQL*Plus Commands

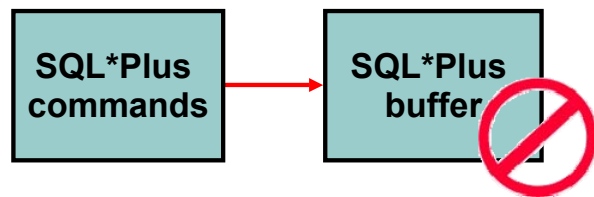
SQL

- A language
- ANSI-standard
- Keywords cannot be abbreviated.
- Statements manipulate data and table definitions in the database.



SQL*Plus

- An environment
- Oracle-proprietary
- Keywords can be abbreviated.
- Commands do not allow manipulation of values in the database.



ORACLE

Copyright © 2009, Oracle. All rights reserved.

SQL and SQL*Plus (continued)

The following table compares SQL and SQL*Plus:

SQL	SQL*Plus
Is a language for communicating with the Oracle server to access data	Recognizes SQL statements and sends them to the server
Is based on American National Standards Institute (ANSI)–standard SQL	Is the Oracle-proprietary interface for executing SQL statements
Manipulates data and table definitions in the database	Does not allow manipulation of values in the database
Is entered into the SQL buffer on one or more lines	Is entered one line at a time, not stored in the SQL buffer
Does not have a continuation character	Uses a dash (–) as a continuation character if the command is longer than one line
Cannot be abbreviated	Can be abbreviated
Uses a termination character to execute commands immediately	Does not require termination characters; executes commands immediately
Uses functions to perform some formatting	Uses commands to format data

Overview of SQL*Plus

- Log in to SQL*Plus.
- Describe the table structure.
- Edit your SQL statement.
- Execute SQL from SQL*Plus.
- Save SQL statements to files and append SQL statements to files.
- Execute saved files.
- Load commands from the file to buffer to edit.

ORACLE

Copyright © 2009, Oracle. All rights reserved.

SQL*Plus

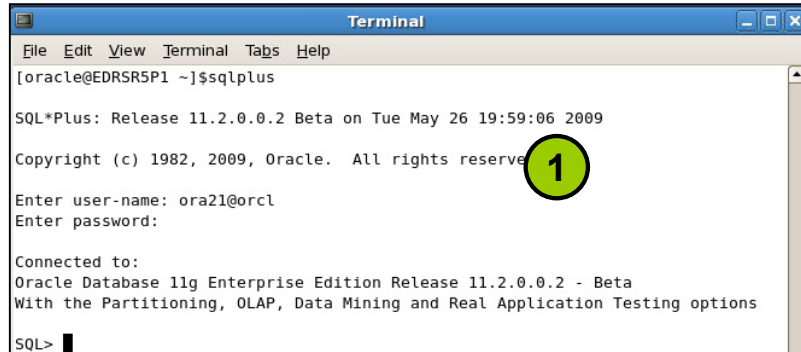
SQL*Plus is an environment in which you can:

- Execute SQL statements to retrieve, modify, add, and remove data from the database
- Format, perform calculations on, store, and print query results in the form of reports
- Create script files to store SQL statements for repeated use in the future

SQL*Plus commands can be divided into the following main categories:

Category	Purpose
Environment	Affect the general behavior of SQL statements for the session.
Format	Format query results.
File manipulation	Save, load, and run script files.
Execution	Send SQL statements from the SQL buffer to the Oracle server.
Edit	Modify SQL statements in the buffer.
Interaction	Create and pass variables to SQL statements, print variable values, and print messages to the screen.
Miscellaneous	Connect to the database, manipulate the SQL*Plus environment, and display column definitions.

Logging In to SQL*Plus



```
Terminal
File Edit View Terminal Tabs Help
[oracle@EDRSR5P1 ~]$sqlplus

SQL*Plus: Release 11.2.0.0.2 Beta on Tue May 26 19:59:06 2009

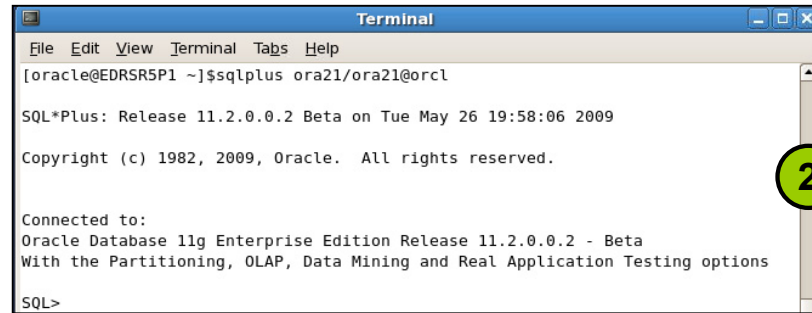
Copyright (c) 1982, 2009, Oracle. All rights reserved.

Enter user-name: ora21@orcl
Enter password:

Connected to:
Oracle Database 11g Enterprise Edition Release 11.2.0.0.2 - Beta
With the Partitioning, OLAP, Data Mining and Real Application Testing options

SQL>
```

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



```
Terminal
File Edit View Terminal Tabs Help
[oracle@EDRSR5P1 ~]$sqlplus ora21/oracle

SQL*Plus: Release 11.2.0.0.2 Beta on Tue May 26 19:58:06 2009

Copyright (c) 1982, 2009, Oracle. All rights reserved.

Connected to:
Oracle Database 11g Enterprise Edition Release 11.2.0.0.2 - Beta
With the Partitioning, OLAP, Data Mining and Real Application Testing options

SQL>
```

ORACLE

Copyright © 2009, Oracle. All rights reserved.

Logging In to SQL*Plus

How you invoke SQL*Plus depends on which type of operating system you are running Oracle Database.

To log in from a Linux environment:

1. Right-click your Linux desktop and select terminal.
2. Enter the `sqlplus` command shown in the slide.
3. Enter the username, password, and database name.

In the syntax:

username Your database username
password Your database password (Your password is visible if you enter it here.)
@database The database connect string

Note: To ensure the integrity of your password, do not enter it at the operating system prompt. Instead, enter only your username. Enter your password at the password prompt.

Displaying the Table Structure

Use the SQL*Plus DESCRIBE command to display the structure of a table:

```
DESC[RIBE] tablename
```

ORACLE

Copyright © 2009, Oracle. All rights reserved.

Displaying the Table Structure

In SQL*Plus, you can display the structure of a table using the DESCRIBE command. The result of the command is a display of column names and data types as well as an indication if a column must contain data.

In the syntax:

tablename The name of any existing table, view, or synonym that is accessible to the user

To describe the DEPARTMENTS table, use this command:

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

Displaying the Table Structure

```
DESCRIBE departments
```

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

ORACLE

Copyright © 2009, Oracle. All rights reserved.

Displaying the Table Structure (continued)

The example in the slide displays the information about the structure of the DEPARTMENTS table. In the result:

Null?: Specifies whether a column must contain data (NOT NULL indicates that a column must contain data.)

Type: Displays the data type for a column

SQL*Plus Editing Commands

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

ORACLE

Copyright © 2009, Oracle. All rights reserved.

SQL*Plus Editing Commands

SQL*Plus commands are entered one line at a time and are not stored in the SQL buffer.

Command	Description
<code>A[PPEND] text</code>	Adds text to the end of the current line
<code>C[HANGE] / old / new</code>	Changes <i>old</i> text to <i>new</i> in the current line
<code>C[HANGE] / text /</code>	Deletes <i>text</i> from the current line
<code>CL[EAR] BUFF[ER]</code>	Deletes all lines from the SQL buffer
<code>DEL</code>	Deletes current line
<code>DEL n</code>	Deletes line <i>n</i>
<code>DEL m n</code>	Deletes lines <i>m</i> to <i>n</i> inclusive

Guidelines

- If you press Enter before completing a command, SQL*Plus prompts you with a line number.
- You terminate the SQL buffer either by entering one of the terminator characters (semicolon or slash) or by pressing Enter twice. The SQL prompt then appears.

SQL*Plus Editing Commands

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

ORACLE

Copyright © 2009, Oracle. All rights reserved.

SQL*Plus Editing Commands (continued)

Command	Description
I [NPUT]	Inserts an indefinite number of lines
I [NPUT] <i>text</i>	Inserts a line consisting of <i>text</i>
L [IST]	Lists all lines in the SQL buffer
L [IST] <i>n</i>	Lists one line (specified by <i>n</i>)
L [IST] <i>m n</i>	Lists a range of lines (<i>m</i> to <i>n</i>) inclusive
R [UN]	Displays and runs the current SQL statement in the buffer
<i>n</i>	Specifies the line to make the current line
<i>n text</i>	Replaces line <i>n</i> with <i>text</i>
0 <i>text</i>	Inserts a line before line 1

Note: You can enter only one SQL*Plus command for each SQL prompt. SQL*Plus commands are not stored in the buffer. To continue a SQL*Plus command on the next line, end the first line with a hyphen (-).

Using LIST, n, and APPEND

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

```
1
 1*  SELECT last_name
```

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

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

ORACLE

Copyright © 2009, Oracle. All rights reserved.

Oracle University and NETEC, S.A. de C.V. use only.

Using LIST, n, and APPEND

- Use the L[IST] command to display the contents of the SQL buffer. The asterisk (*) beside line 2 in the buffer indicates that line 2 is the current line. Any edits that you made apply to the current line.
- Change the number of the current line by entering the number (n) of the line that you want to edit. The new current line is displayed.
- Use the A[PPEND] command to add text to the current line. The newly edited line is displayed. Verify the new contents of the buffer by using the LIST command.

Note: Many SQL*Plus commands, including LIST and APPEND, can be abbreviated to just their first letter. LIST can be abbreviated to L; APPEND can be abbreviated to A.

Using the CHANGE Command

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

```
c/employees/departments
1* SELECT * from departments
```

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

ORACLE

Copyright © 2009, Oracle. All rights reserved.

Using the CHANGE Command

- Use L [IST] to display the contents of the buffer.
- Use the C [HANGE] command to alter the contents of the current line in the SQL buffer. In this case, replace the employees table with the departments table. The new current line is displayed.
- Use the L [IST] command to verify the new contents of the buffer.

SQL*Plus File Commands

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

ORACLE

Copyright © 2009, Oracle. All rights reserved.

SQL*Plus File Commands

SQL statements communicate with the Oracle server. SQL*Plus commands control the environment, format query results, and manage files. You can use the commands described in the following table:

Command	Description
<code>SAV[E] filename [.ext]</code> <code>[REP[LACE] APP[END]]</code>	Saves current contents of SQL buffer to a file. Use APPEND to add to an existing file; use REPLACE to overwrite an existing file. The default extension is .sql.
<code>GET filename [.ext]</code>	Writes the contents of a previously saved file to the SQL buffer. The default extension for the file name is .sql.
<code>STA[RT] filename [.ext]</code>	Runs a previously saved command file
<code>@ filename</code>	Runs a previously saved command file (same as START)
<code>ED[IT]</code>	Invokes the editor and saves the buffer contents to a file named <code>afiedt.buf</code>
<code>ED[IT] [filename [.ext]]</code>	Invokes the editor to edit the contents of a saved file
<code>SPO[OL] [filename [.ext]]</code> <code>OFF OUT]</code>	Stores query results in a file. OFF closes the spool file. OUT closes the spool file and sends the file results to the printer.
<code>EXIT</code>	Quits SQL*Plus

Using the SAVE and START Commands

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

```
SAVE my_query
Created file my_query
```

```
START my_query

LAST_NAME                MANAGER_ID DEPARTMENT_ID
-----
King                                90
Kochhar                        100      90
...
107 rows selected.
```

ORACLE

Copyright © 2009, Oracle. All rights reserved.

Using the SAVE and START Commands

SAVE

Use the SAVE command to store the current contents of the buffer in a file. In this way, you can store frequently used scripts for use in the future.

START

Use the START command to run a script in SQL*Plus. You can also, alternatively, use the symbol @ to run a script.

```
@my_query
```

SERVEROUTPUT Command

- Use the `SET SERVEROUT [PUT]` command to control whether to display the output of stored procedures or PL/SQL blocks in SQL*Plus.
- The `DBMS_OUTPUT` line length limit is increased from 255 bytes to 32767 bytes.
- The default size is now unlimited.
- Resources are not preallocated when `SERVEROUTPUT` is set.
- Because there is no performance penalty, use `UNLIMITED` unless you want to conserve physical memory.

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

ORACLE

Copyright © 2009, Oracle. All rights reserved.

SERVEROUTPUT Command

Most of the PL/SQL programs perform input and output through SQL statements to store data in database tables or query those tables. All other PL/SQL input/output is done through APIs that interact with other programs. For example, the `DBMS_OUTPUT` package has procedures such as `PUT_LINE`. To see the result outside of PL/SQL you require another program, such as SQL*Plus, to read and display the data passed to `DBMS_OUTPUT`.

SQL*Plus does not display `DBMS_OUTPUT` data unless you first issue the SQL*Plus command `SET SERVEROUTPUT ON` as follows:

```
SET SERVEROUTPUT ON
```

Note

- `SIZE` sets the number of bytes of the output that can be buffered within the Oracle Database server. The default is `UNLIMITED`. `n` cannot be less than 2000 or greater than 1,000,000.
- For additional information about `SERVEROUTPUT`, see the *Oracle Database PL/SQL User's Guide and Reference 11g*.

Using the SQL*Plus SPOOL Command

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

Option	Description
file_name[.ext]	Spools output to the specified file name
CRE[ATE]	Creates a new file with the name specified
REP[LACE]	Replaces the contents of an existing file. If the file does not exist, REPLACE creates the file.
APP[END]	Adds the contents of the buffer to the end of the file you specify
OFF	Stops spooling
OUT	Stops spooling and sends the file to your computer's standard (default) printer

ORACLE

Copyright © 2009, Oracle. All rights reserved.

Using the SQL*Plus SPOOL Command

The SPOOL command stores query results in a file or optionally sends the file to a printer. The SPOOL command has been enhanced. You can now append to, or replace an existing file, where previously you could only use SPOOL to create (and replace) a file. REPLACE is the default.

To spool output generated by commands in a script without displaying the output on the screen, use SET TERMOUT OFF. SET TERMOUT OFF does not affect output from commands that run interactively.

You must use quotation marks around file names containing white space. To create a valid HTML file using SPOOL APPEND commands, you must use PROMPT or a similar command to create the HTML page header and footer. The SPOOL APPEND command does not parse HTML tags. Set SQLPLUSCOMPAT[IBILITY] to 9.2 or earlier to disable the CREATE, APPEND, and SAVE parameters.

Using the AUTOTRACE Command

- It displays a report after the successful execution of SQL data manipulation statements (DML) statements such as SELECT, INSERT, UPDATE, or DELETE.
- The report can now include execution statistics and the query execution path.

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

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

ORACLE

Copyright © 2009, Oracle. All rights reserved.

Using the AUTOTRACE Command

EXPLAIN shows the query execution path by performing an EXPLAIN PLAN. STATISTICS displays SQL statement statistics. The formatting of your AUTOTRACE report may vary depending on the version of the server to which you are connected and the configuration of the server. The DBMS_XPLAN package provides an easy way to display the output of the EXPLAIN PLAN command in several predefined formats.

Note

- For additional information about the package and subprograms, see the *Oracle Database PL/SQL Packages and Types Reference 11g* guide.
- For additional information about the EXPLAIN PLAN, see *Oracle Database SQL Reference 11g*.
- For additional information about Execution Plans and the statistics, see the *Oracle Database Performance Tuning Guide 11g*.

Summary

In this appendix, you should have learned how to use SQL*Plus as an environment to do the following:

- Execute SQL statements
- Edit SQL statements
- Format the output
- Interact with script files

ORACLE

Copyright © 2009, Oracle. All rights reserved.

Summary

SQL*Plus is an execution environment that you can use to send SQL commands to the database server and to edit and save SQL commands. You can execute commands from the SQL prompt or from a script file.

E

Using JDeveloper

ORACLE

Copyright © 2009, Oracle. All rights reserved.

Objectives

After completing this appendix, you should be able to do the following:

- List the key features of Oracle JDeveloper
- Create a database connection in JDeveloper
- Manage database objects in JDeveloper
- Use JDeveloper to execute SQL Commands
- Create and run PL/SQL Program Units

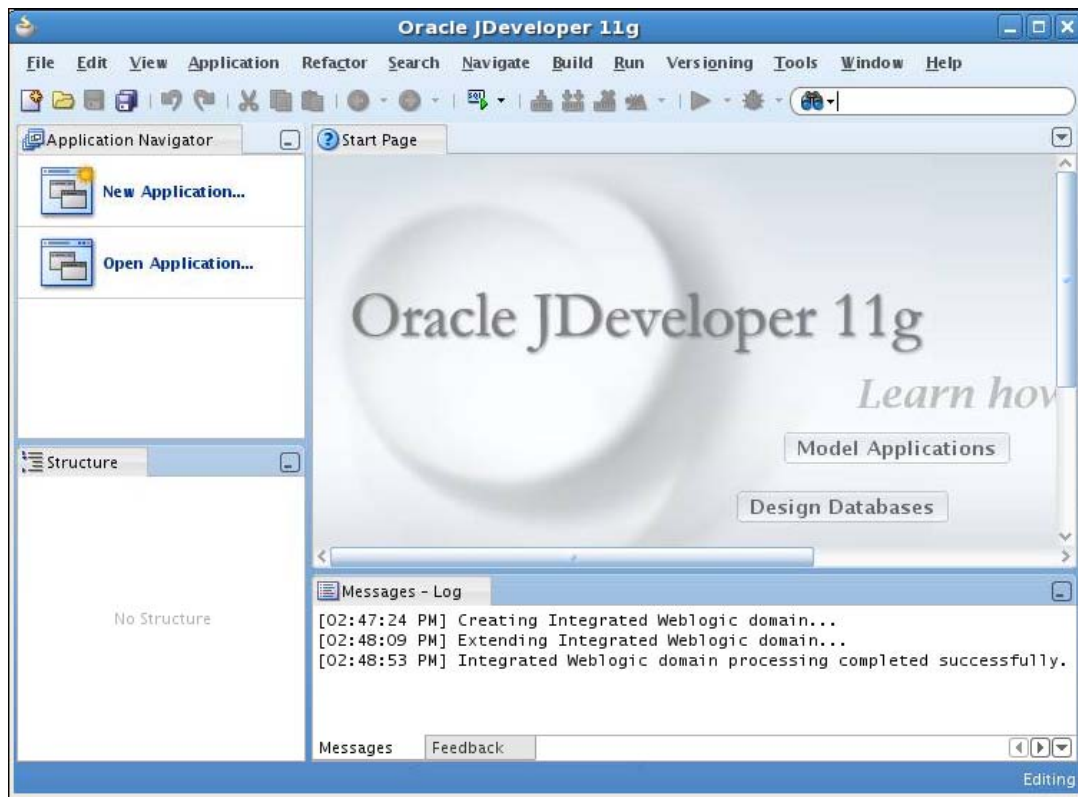
ORACLE

Copyright © 2009, Oracle. All rights reserved.

Objectives

In this appendix, you are introduced to the tool JDeveloper. You learn how to use JDeveloper for your database development tasks.

Oracle JDeveloper



ORACLE

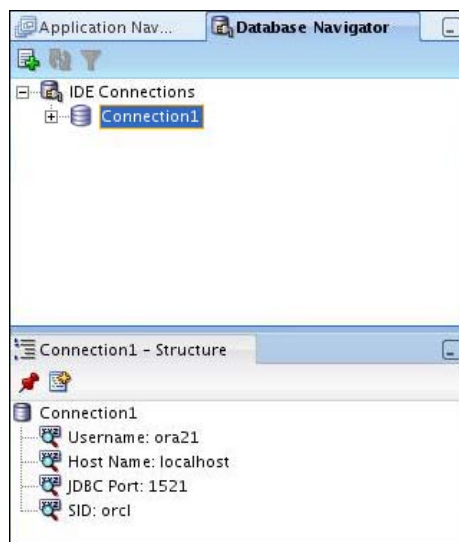
Copyright © 2009, Oracle. All rights reserved.

Oracle JDeveloper

Oracle JDeveloper is an integrated development environment (IDE) for developing and deploying Java applications and Web services. It supports every stage of the software development life cycle (SDLC) from modeling to deploying. It has the features to use the latest industry standards for Java, XML, and SQL while developing an application.

Oracle JDeveloper 11g initiates a new approach to J2EE development with features that enable visual and declarative development. This innovative approach makes J2EE development simple and efficient.

Database Navigator



ORACLE

Copyright © 2009, Oracle. All rights reserved.

Database Navigator

Using Oracle JDeveloper, you can store the information necessary to connect to a database in an object called “connection.” A connection is stored as part of the IDE settings, and can be exported and imported for easy sharing among groups of users. A connection serves several purposes from browsing the database and building applications, all the way through to deployment.

Creating a Connection

1 Click the New Connection icon in the Database Navigator.

2 In the Create Database Connection window, enter the username, password and the SID.

3 Test the connection.

4 Click OK.

ORACLE

Copyright © 2009, Oracle. All rights reserved.

Creating a Connection

A connection is an object that specifies the necessary information for connecting to a specific database as a specific user of that database. You can create and test connections for multiple databases and for multiple schemas.

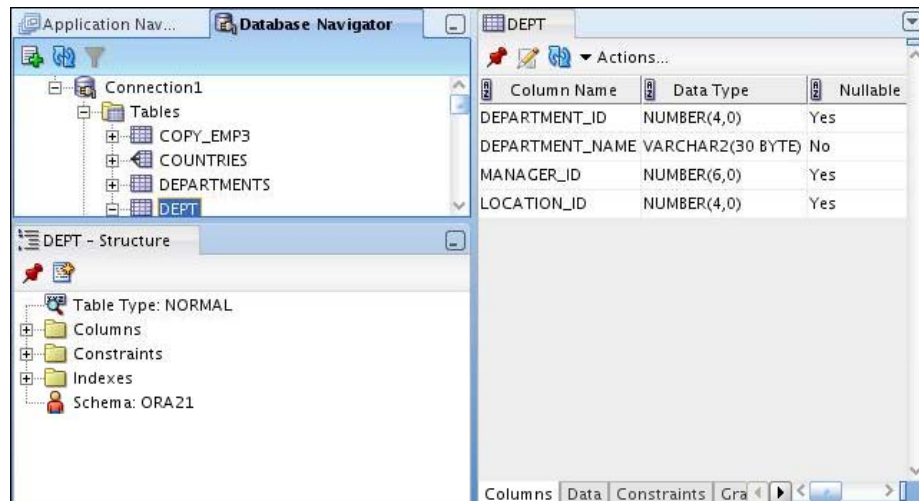
To create a database connection, perform the following steps:

1. Click the New Connection icon in the Database Navigator.
2. In the Create Database Connection window, enter the connection name. Enter the username and password of the schema that you want to connect to. Enter the SID of the database that you want to connect to.
3. Click Test to ensure that the connection has been set correctly.
4. Click OK.

Browsing Database Objects

Use the Database Navigator to:

- Browse through many objects in a database schema
- Review the definitions of objects at a glance



ORACLE

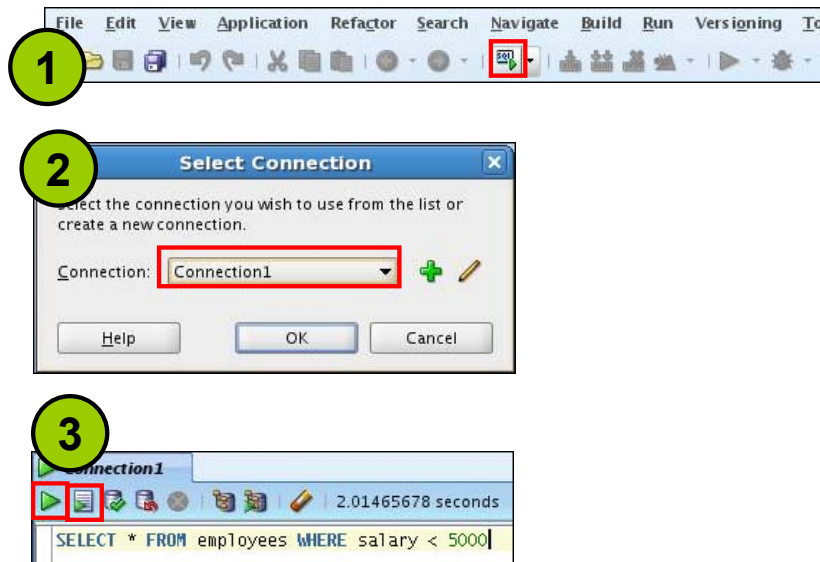
Copyright © 2009, Oracle. All rights reserved.

Browsing Database Objects

After you create a database connection, you can use the Database Navigator to browse through many objects in a database schema including tables, views, indexes, packages, procedures, triggers, and types.

You can object definitions broken into tabs of information that is pulled out of the data dictionary. For example, if you select a table in the Navigator, details about columns, constraints, grants, statistics, triggers, and so on are displayed on an easy-to-read tabbed page.

Executing SQL Statements



ORACLE

Copyright © 2009, Oracle. All rights reserved.

Executing SQL Statements

To execute a SQL statement, perform the following steps:

1. Click the Open SQL Worksheet icon.
2. Select the connection.
3. Execute the SQL command by clicking:
 - The **Execute statement** button or by pressing F9. The output is as follows:

The screenshot shows the 'Results' window with the following data:

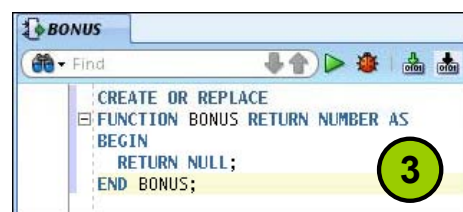
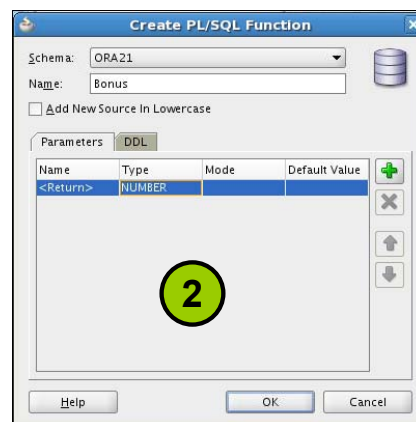
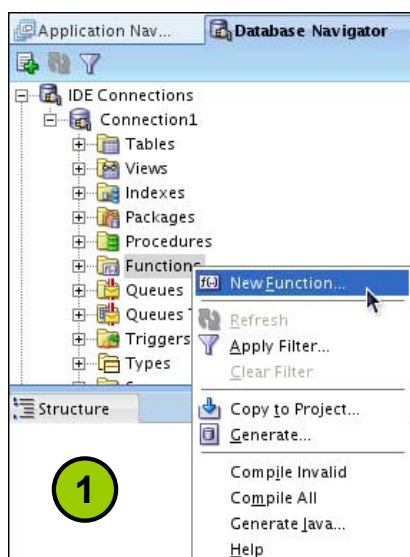
	EMPLOYEE_ID	FIRST_NAME	LAST_NAME	
1	100	Steven	King	SK
2	101	Neena	Kochhar	NK

- The **Run Script** button or by pressing F5. The output is as follows:

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

EMPLOYEE_ID	FIRST_NAME	LAST_NAME
100	Steven	King

Creating Program Units



Skeleton of the function

ORACLE

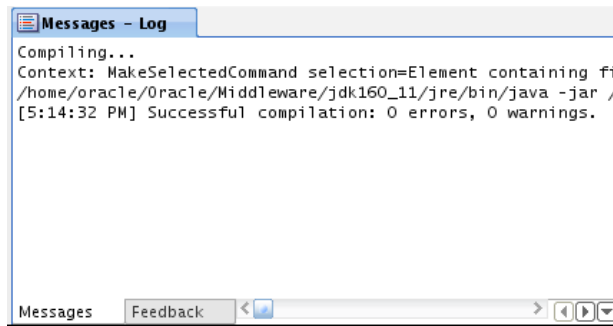
Copyright © 2009, Oracle. All rights reserved.

Creating Program Units

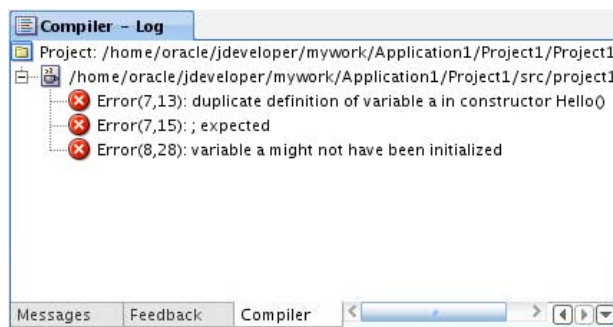
To create a PL/SQL program unit:

1. Select View > Database Navigator. Select and expand a database connection. Right-click a folder corresponding to the object type (Procedures, Packages, Functions). Select “New [Procedures|Packages|Functions]”.
2. Enter a valid name for the function, package, or procedure, and click OK.
3. A skeleton definition is created and opened in the Code Editor. You can then edit the subprogram to suit your need.

Compiling



Compilation with errors



Compilation without errors

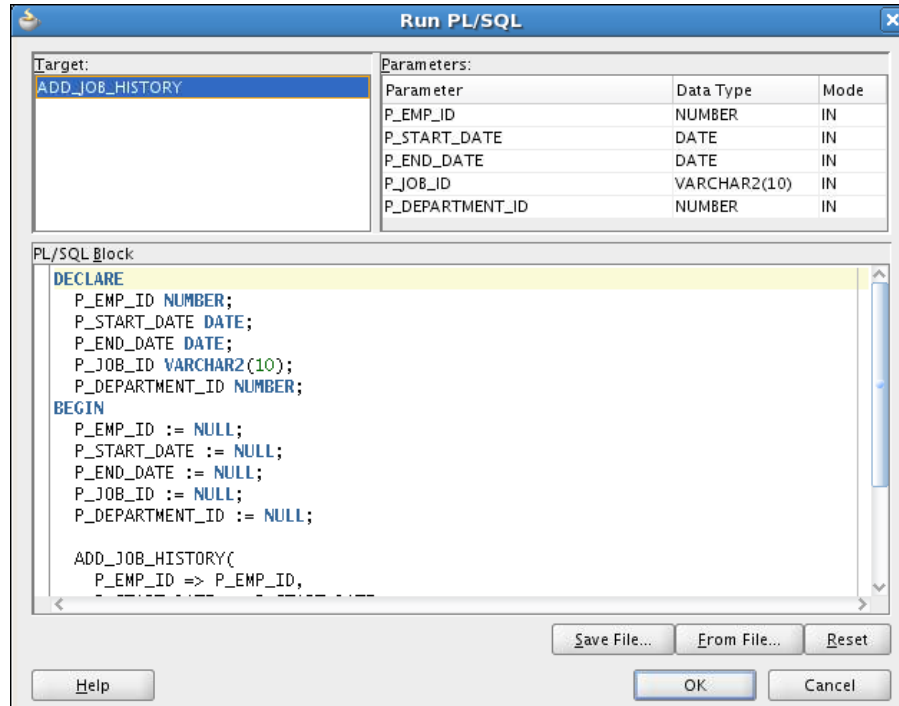
ORACLE

Copyright © 2009, Oracle. All rights reserved.

Compiling

After editing the skeleton definition, you need to compile the program unit. Right-click the PL/SQL object that you need to compile in the Connection Navigator, and then select Compile. Alternatively, you can press CTRL + SHIFT + F9 to compile.

Running a Program Unit



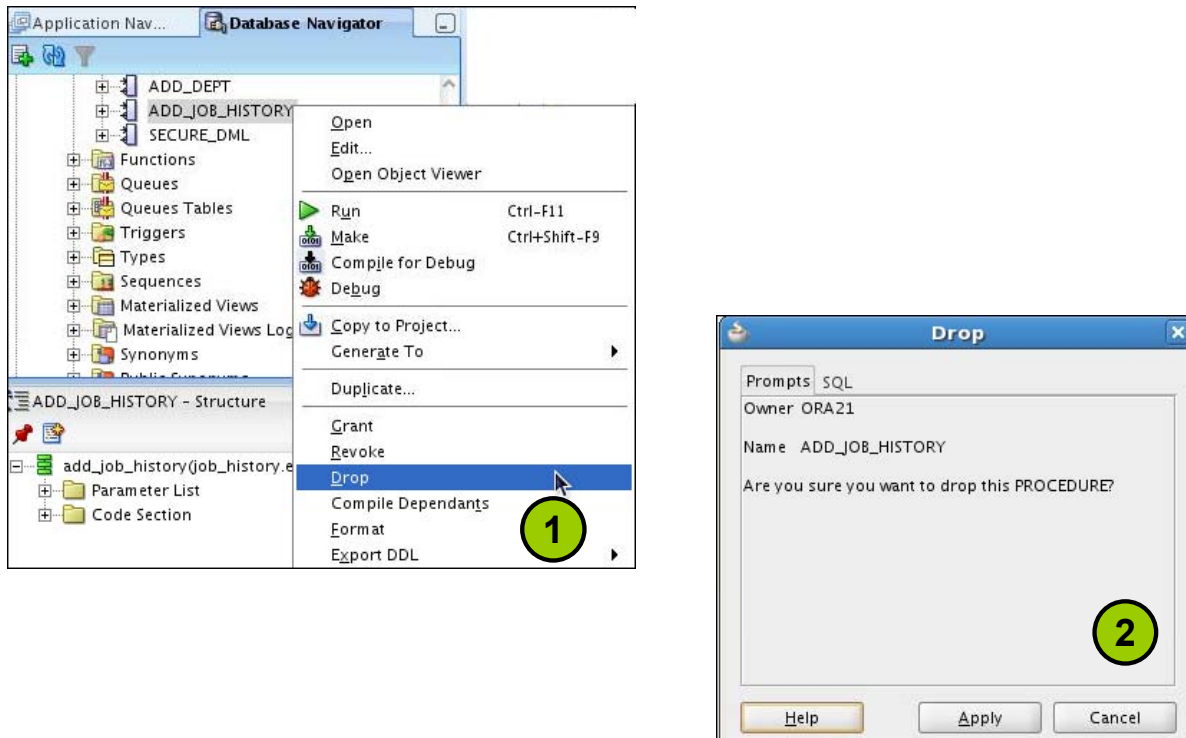
ORACLE

Copyright © 2009, Oracle. All rights reserved.

Running a Program Unit

To execute the program unit, right-click the object and select Run. The Run PL/SQL dialog box appears. You may need to change the NULL values with reasonable values that are passed into the program unit. After you change the values, click OK. The output is displayed in the Message-Log window.

Dropping a Program Unit



ORACLE

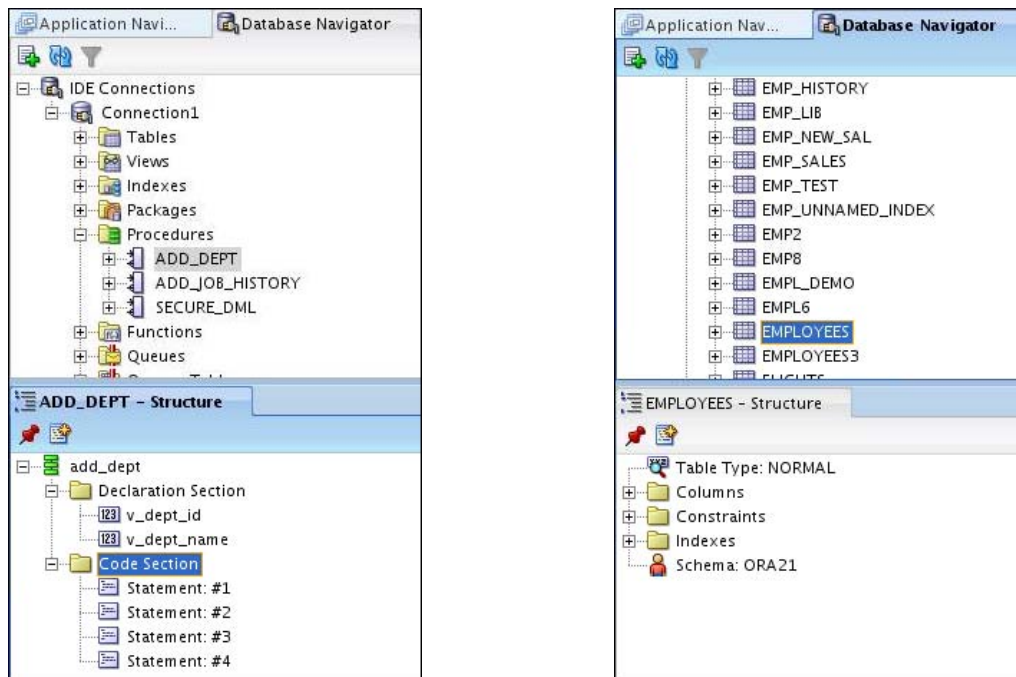
Copyright © 2009, Oracle. All rights reserved.

Dropping a Program Unit

To drop a program unit:

1. Right-click the object and select Drop.
The Drop Confirmation dialog box appears.
2. Click Apply.
The object is dropped from the database.

Structure Window



ORACLE

Copyright © 2009, Oracle. All rights reserved.

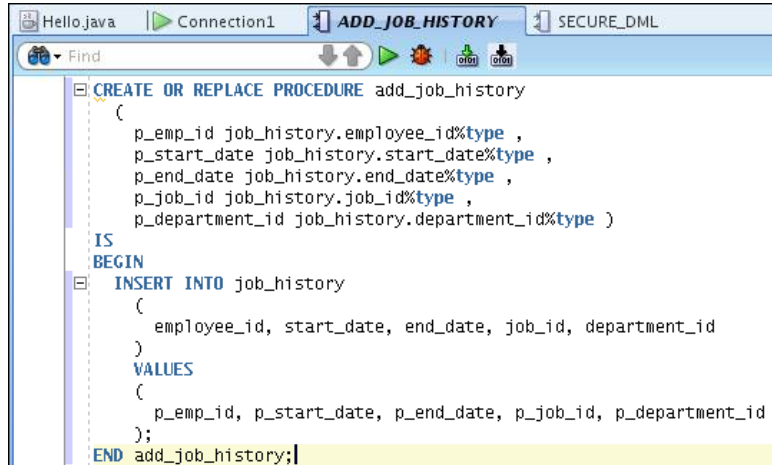
Structure Window

The Structure window offers a structural view of the data in the document that is currently selected in the active window of those windows that participate in providing structure: the navigators, the editors and viewers, and the Property Inspector.

In the Structure window, you can view the document data in a variety of ways. The structures that are available for display are based on document type. For a Java file, you can view code structure, UI structure, or UI model data. For an XML file, you can view XML structure, design structure, or UI model data.

The Structure window is dynamic, tracking always the current selection of the active window (unless you freeze the window's contents on a particular view), as is pertinent to the currently active editor. When the current selection is a node in the navigator, the default editor is assumed. To change the view on the structure for the current selection, select a different structure tab.

Editor Window



```

Hello.java | Connection1 | ADD_JOB_HISTORY | SECURE_DML
Find
CREATE OR REPLACE PROCEDURE add_job_history
(
  p_emp_id job_history.employee_id%type ,
  p_start_date job_history.start_date%type ,
  p_end_date job_history.end_date%type ,
  p_job_id job_history.job_id%type ,
  p_department_id job_history.department_id%type )
IS
BEGIN
  INSERT INTO job_history
  (
    employee_id, start_date, end_date, job_id, department_id
  )
  VALUES
  (
    p_emp_id, p_start_date, p_end_date, p_job_id, p_department_id
  );
END add_job_history;
```

ORACLE

Copyright © 2009, Oracle. All rights reserved.

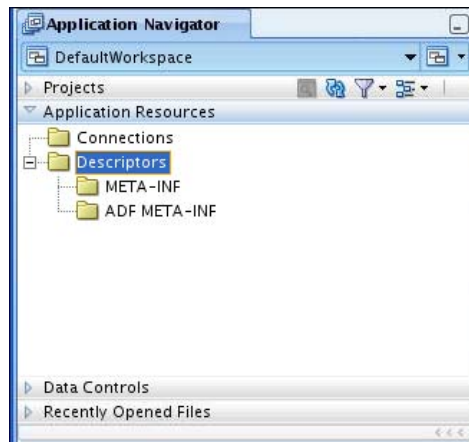
Editor Window

You can view your project files all in one single editor window, you can open multiple views of the same file, or you can open multiple views of different files.

The tabs at the top of the editor window are the document tabs. Selecting a document tab gives that file focus, bringing it to the foreground of the window in the current editor.

The tabs at the bottom of the editor window for a given file are the editor tabs. Selecting an editor tab opens the file in that editor.

Application Navigator



ORACLE

Copyright © 2009, Oracle. All rights reserved.

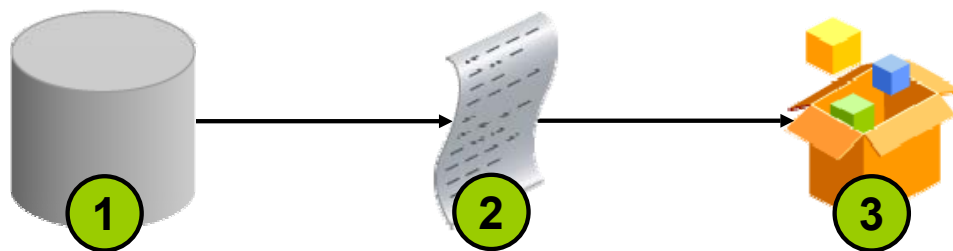
Application Navigator

Application Navigator gives you a logical view of your application and the data that it contains. Application Navigator provides an infrastructure that the different extensions can plug in to and use to organize their data and menus in a consistent, abstract manner. While Application Navigator can contain individual files (such as Java source files), it is designed to consolidate complex data. Complex data types such as entity objects, Unified Modeling Language (UML) diagrams, Enterprise JavaBeans (EJB), or Web services appear in this navigator as single nodes. The raw files that make up these abstract nodes appear in the Structure window.

Deploying Java Stored Procedures

Before deploying Java stored procedures, perform the following steps:

1. Create a database connection.
2. Create a deployment profile.
3. Deploy the objects.



ORACLE

Copyright © 2009, Oracle. All rights reserved.

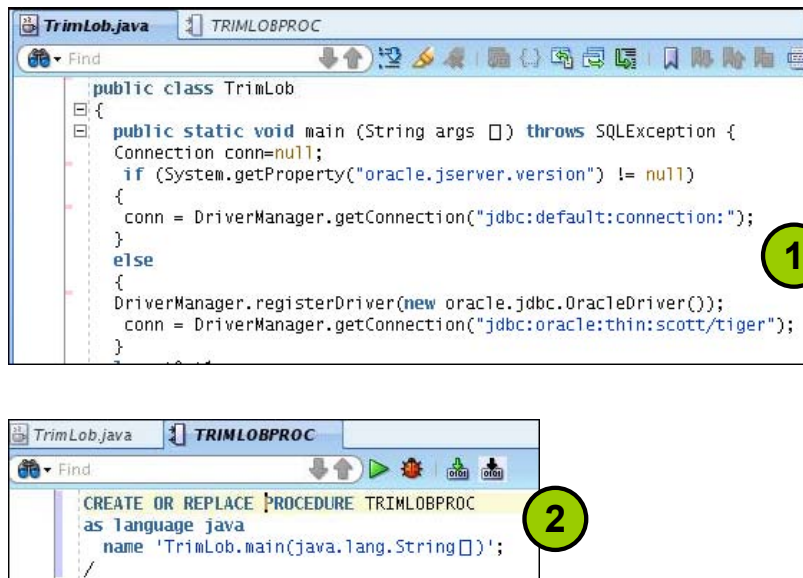
Deploying Java Stored Procedures

Create a deployment profile for Java stored procedures, and then deploy the classes and, optionally, any public static methods in JDeveloper using the settings in the profile.

Deploying to the database uses the information provided in the Deployment Profile Wizard and two Oracle Database utilities:

- `loadjava` loads the Java class containing the stored procedures to an Oracle database.
- `publish` generates the PL/SQL call-specific wrappers for the loaded public static methods. Publishing enables the Java methods to be called as PL/SQL functions or procedures.

Publishing Java to PL/SQL



ORACLE

Copyright © 2009, Oracle. All rights reserved.

Publishing Java to PL/SQL

The slide shows the Java code and illustrates how to publish the Java code in a PL/SQL procedure.

How Can I Learn More About JDeveloper 11g ?

Topic	Web site
Oracle JDeveloper Product Page	http://www.oracle.com/technology/products/jdev/index.html
Oracle JDeveloper 11g Tutorials	http://www.oracle.com/technology/obe/obe11jdev/11/index.html
Oracle JDeveloper 11g Product Documentation	http://www.oracle.com/technology/documentation/jdev.html
Oracle JDeveloper 11g Discussion Forum	http://forums.oracle.com/forums/forum.jspa?forumID=83

ORACLE

Copyright © 2009, Oracle. All rights reserved.

Summary

In this appendix, you should have learned how to use JDeveloper to do the following:

- List the key features of Oracle JDeveloper
- Create a database connection in JDeveloper
- Manage database objects in JDeveloper
- Use JDeveloper to execute SQL Commands
- Create and run PL/SQL Program Units

ORACLE

Copyright © 2009, Oracle. All rights reserved.

Objectives

In this appendix, you are introduced to the tool JDeveloper. You learn how to use JDeveloper for your database development tasks.

Generating Reports by Grouping Related Data

ORACLE

Copyright © 2009, Oracle. All rights reserved.

Objectives

After completing this appendix, you should be able to use the:

- ROLLUP operation to produce subtotal values
- CUBE operation to produce cross-tabulation values
- GROUPING function to identify the row values created by ROLLUP or CUBE
- GROUPING SETS to produce a single result set

ORACLE

Copyright © 2009, Oracle. All rights reserved.

Objectives

In this appendix, you learn how to:

- Group data to obtain the subtotal values by using the ROLLUP operator
- Group data to obtain the cross-tabulation values by using the CUBE operator
- Use the GROUPING function to identify the level of aggregation in the result set produced by a ROLLUP or CUBE operator
- Use GROUPING SETS to produce a single result set that is equivalent to a UNION ALL approach

Review of Group Functions

- Group functions operate on sets of rows to give one result per group.

```
SELECT      [column,] group_function(column) . . .
FROM        table
[WHERE      condition]
[GROUP BY  group_by_expression]
[ORDER BY  column];
```

- Example:

```
SELECT AVG(salary), STDDEV(salary),
       COUNT(commission_pct), MAX(hire_date)
FROM   employees
WHERE  job_id LIKE 'SA%';
```

ORACLE

Copyright © 2009, Oracle. All rights reserved.

Group Functions

You can use the GROUP BY clause to divide the rows in a table into groups. You can then use group functions to return summary information for each group. Group functions can appear in select lists and in ORDER BY and HAVING clauses. The Oracle server applies the group functions to each group of rows and returns a single result row for each group.

Types of group functions: Each of the group functions—AVG, SUM, MAX, MIN, COUNT, STDDEV, and VARIANCE—accepts one argument. The AVG, SUM, STDDEV, and VARIANCE functions operate only on numeric values. MAX and MIN can operate on numeric, character, or date data values. COUNT returns the number of non-NULL rows for the given expression. The example in the slide calculates the average salary, standard deviation on the salary, number of employees earning a commission, and the maximum hire date for those employees whose JOB_ID begins with SA.

Guidelines for Using Group Functions

- The data types for the arguments can be CHAR, VARCHAR2, NUMBER, or DATE.
- All group functions except COUNT (*) ignore null values. To substitute a value for null values, use the NVL function. COUNT returns either a number or zero.
- The Oracle server implicitly sorts the result set in ascending order of the grouping columns specified, when you use a GROUP BY clause. To override this default ordering, you can use DESC in an ORDER BY clause.

Review of the GROUP BY Clause

- Syntax:

```
SELECT      [column,] group_function(column). . .
FROM        table
[WHERE      condition]
[GROUP BY   group_by_expression]
[ORDER BY   column];
```

- Example:

```
SELECT  department_id, job_id, SUM(salary),
        COUNT(employee_id)
FROM    employees
GROUP BY department_id, job_id ;
```

ORACLE

Copyright © 2009, Oracle. All rights reserved.

Review of the GROUP BY Clause

The example illustrated in the slide is evaluated by the Oracle server as follows:

- The SELECT clause specifies that the following columns be retrieved:
 - Department ID and job ID columns from the EMPLOYEES table
 - The sum of all the salaries and the number of employees in each group that you have specified in the GROUP BY clause
- The GROUP BY clause specifies how the rows should be grouped in the table. The total salary and the number of employees are calculated for each job ID within each department. The rows are grouped by department ID and then grouped by job within each department.

Review of the HAVING Clause

- Use the HAVING clause to specify which groups are to be displayed.
- You further restrict the groups on the basis of a limiting condition.

```
SELECT      [column,] group_function(column)...  
FROM        table  
[WHERE      condition]  
[GROUP BY   group_by_expression]  
[HAVING     having_expression]  
[ORDER BY   column];
```

ORACLE

Copyright © 2009, Oracle. All rights reserved.

HAVING Clause

Groups are formed and group functions are calculated before the HAVING clause is applied to the groups. The HAVING clause can precede the GROUP BY clause, but it is recommended that you place the GROUP BY clause first because it is more logical.

The Oracle server performs the following steps when you use the HAVING clause:

1. It groups rows.
2. It applies the group functions to the groups and displays the groups that match the criteria in the HAVING clause.

GROUP BY with ROLLUP and CUBE Operators

- Use ROLLUP or CUBE with GROUP BY to produce superaggregate rows by cross-referencing columns.
- ROLLUP grouping produces a result set containing the regular grouped rows and the subtotal values.
- CUBE grouping produces a result set containing the rows from ROLLUP and cross-tabulation rows.

ORACLE

Copyright © 2009, Oracle. All rights reserved.

GROUP BY with the ROLLUP and CUBE Operators

You specify ROLLUP and CUBE operators in the GROUP BY clause of a query. ROLLUP grouping produces a result set containing the regular grouped rows and subtotal rows. The ROLLUP operator also calculates a grand total. The CUBE operation in the GROUP BY clause groups the selected rows based on the values of all possible combinations of expressions in the specification and returns a single row of summary information for each group. You can use the CUBE operator to produce cross-tabulation rows.

Note: When working with ROLLUP and CUBE, make sure that the columns following the GROUP BY clause have meaningful, real-life relationships with each other; otherwise, the operators return irrelevant information.

ROLLUP Operator

- ROLLUP is an extension to the GROUP BY clause.
- Use the ROLLUP operation to produce cumulative aggregates, such as subtotals.

```
SELECT      [column,] group_function(column). . .  
FROM        table  
[WHERE      condition]  
[GROUP BY   [ROLLUP] group_by_expression]  
[HAVING     having_expression];  
[ORDER BY   column];
```

ORACLE

Copyright © 2009, Oracle. All rights reserved.

ROLLUP Operator

The ROLLUP operator delivers aggregates and superaggregates for expressions within a GROUP BY statement. The ROLLUP operator can be used by report writers to extract statistics and summary information from result sets. The cumulative aggregates can be used in reports, charts, and graphs.

The ROLLUP operator creates groupings by moving in one direction, from right to left, along the list of columns specified in the GROUP BY clause. It then applies the aggregate function to these groupings.

Note

- To produce subtotals in n dimensions (that is, n columns in the GROUP BY clause) without a ROLLUP operator, $n+1$ SELECT statements must be linked with UNION ALL. This makes the query execution inefficient because each of the SELECT statements causes table access. The ROLLUP operator gathers its results with just one table access. The ROLLUP operator is useful when there are many columns involved in producing the subtotals.
- Subtotals and totals are produced with ROLLUP. CUBE produces totals as well but effectively rolls up in each possible direction, producing cross-tabular data.

ROLLUP Operator: Example

```
SELECT  department_id, job_id, SUM(salary)
FROM    employees
WHERE   department_id < 60
GROUP BY ROLLUP (department_id, job_id);
```

	DEPARTMENT_ID	JOB_ID	SUM(SALARY)
1	10	AD_ASST	4400
2	10	(null)	4400
3	20	MK_MAN	13000
4	20	MK_REP	6000
5	20	(null)	19000
6	30	PU_MAN	11000
7	30	PU_CLERK	13900
8	30	(null)	24900
9	40	HR_REP	6500
10	40	(null)	6500
11	50	ST_MAN	36400
12	50	SH_CLERK	64300
13	50	ST_CLERK	55700
14	50	(null)	156400
15	(null)	(null)	211200

1

2

3

ORACLE

Copyright © 2009, Oracle. All rights reserved.

Example of a ROLLUP Operator

In the example in the slide:

- Total salaries for every job ID within a department for those departments whose department ID is less than 60 are displayed by the GROUP BY clause
- The ROLLUP operator displays:
 - The total salary for each department whose department ID is less than 60
 - The total salary for all departments whose department ID is less than 60, irrespective of the job IDs

In this example, 1 indicates a group totaled by both DEPARTMENT_ID and JOB_ID, 2 indicates a group totaled only by DEPARTMENT_ID, and 3 indicates the grand total.

The ROLLUP operator creates subtotals that roll up from the most detailed level to a grand total, following the grouping list specified in the GROUP BY clause. First, it calculates the standard aggregate values for the groups specified in the GROUP BY clause (in the example, the sum of salaries grouped on each job within a department). Then it creates progressively higher-level subtotals, moving from right to left through the list of grouping columns. (In the example, the sum of salaries for each department is calculated, followed by the sum of salaries for all departments.)

- Given n expressions in the ROLLUP operator of the GROUP BY clause, the operation results in $n + 1$ (in this case, $2 + 1 = 3$) groupings.
- Rows based on the values of the first n expressions are called rows or regular rows, and the others are called superaggregate rows.

CUBE Operator

- CUBE is an extension to the GROUP BY clause.
- You can use the CUBE operator to produce cross-tabulation values with a single SELECT statement.

```
SELECT      [column,] group_function(column)...  
FROM        table  
[WHERE      condition]  
[GROUP BY   [CUBE] group_by_expression]  
[HAVING     having_expression]  
[ORDER BY   column];
```

ORACLE

Copyright © 2009, Oracle. All rights reserved.

CUBE Operator

The CUBE operator is an additional switch in the GROUP BY clause in a SELECT statement. The CUBE operator can be applied to all aggregate functions, including AVG, SUM, MAX, MIN, and COUNT. It is used to produce result sets that are typically used for cross-tabular reports.

ROLLUP produces only a fraction of possible subtotal combinations, whereas CUBE produces subtotals for all possible combinations of groupings specified in the GROUP BY clause, and a grand total.

The CUBE operator is used with an aggregate function to generate additional rows in a result set. Columns included in the GROUP BY clause are cross-referenced to produce a superset of groups. The aggregate function specified in the select list is applied to these groups to produce summary values for the additional superaggregate rows. The number of extra groups in the result set is determined by the number of columns included in the GROUP BY clause.

In fact, every possible combination of the columns or expressions in the GROUP BY clause is used to produce superaggregates. If you have n columns or expressions in the GROUP BY clause, there will be 2^n possible superaggregate combinations. Mathematically, these combinations form an n -dimensional cube, which is how the operator got its name.

By using application or programming tools, these superaggregate values can then be fed into charts and graphs that convey results and relationships visually and effectively.

CUBE Operator: Example

```
SELECT  department_id, job_id, SUM(salary)
FROM    employees
WHERE   department_id < 60
GROUP BY CUBE (department_id, job_id) ;
```

	DEPARTMENT_ID	JOB_ID	SUM(SALARY)	
1	(null)	(null)	211200	1
2	(null)	HR_REP	6500	
3	(null)	MK_MAN	13000	
4	(null)	MK_REP	6000	
5	(null)	PU_MAN	11000	
6	(null)	ST_MAN	36400	2
7	(null)	AD_ASST	4400	
8	(null)	PU_CLERK	13900	
9	(null)	SH_CLERK	64300	
10	(null)	ST_CLERK	55700	
11	10	(null)	4400	3
12	10	AD_ASST	4400	
13	20	(null)	19000	
14	20	MK_MAN	13000	
15	20	MK_REP	6000	
16	30	(null)	24900	4

ORACLE

Copyright © 2009, Oracle. All rights reserved.

Example of a CUBE Operator

The output of the SELECT statement in the example can be interpreted as follows:

- The total salary for every job within a department (for those departments whose department ID is less than 60)
- The total salary for each department whose department ID is less than 60
- The total salary for each job irrespective of the department
- The total salary for those departments whose department ID is less than 60, irrespective of the job titles

In this example, 1 indicates the grand total, 2 indicates the rows totaled by JOB_ID alone, 3 indicates some of the rows totaled by DEPARTMENT_ID and JOB_ID, and 4 indicates some of the rows totaled by DEPARTMENT_ID alone.

The CUBE operator has also performed the ROLLUP operation to display the subtotals for those departments whose department ID is less than 60 and the total salary for those departments whose department ID is less than 60, irrespective of the job titles. Further, the CUBE operator displays the total salary for every job irrespective of the department.

Note: Similar to the ROLLUP operator, producing subtotals in n dimensions (that is, n columns in the GROUP BY clause) without a CUBE operator requires that 2^n SELECT statements be linked with UNION ALL. Thus, a report with three dimensions requires $2^3 = 8$ SELECT statements to be linked with UNION ALL.

GROUPING Function

The GROUPING function:

- Is used with either the CUBE or ROLLUP operator
- Is used to find the groups forming the subtotal in a row
- Is used to differentiate stored NULL values from NULL values created by ROLLUP or CUBE
- Returns 0 or 1

```
SELECT    [column,] group_function(column) .. ,  
          GROUPING(expr)  
FROM      table  
[WHERE    condition]  
[GROUP BY [ROLLUP] [CUBE] group_by_expression]  
[HAVING   having_expression]  
[ORDER BY column];
```

ORACLE

Copyright © 2009, Oracle. All rights reserved.

GROUPING Function

The GROUPING function can be used with either the CUBE or ROLLUP operator to help you understand how a summary value has been obtained.

The GROUPING function uses a single column as its argument. The *expr* in the GROUPING function must match one of the expressions in the GROUP BY clause. The function returns a value of 0 or 1.

The values returned by the GROUPING function are useful to:

- Determine the level of aggregation of a given subtotal (that is, the group or groups on which the subtotal is based)
- Identify whether a NULL value in the expression column of a row of the result set indicates:
 - A NULL value from the base table (stored NULL value)
 - A NULL value created by ROLLUP or CUBE (as a result of a group function on that expression)

A value of 0 returned by the GROUPING function based on an expression indicates one of the following:

- The expression has been used to calculate the aggregate value.
- The NULL value in the expression column is a stored NULL value.

A value of 1 returned by the GROUPING function based on an expression indicates one of the following:

- The expression has not been used to calculate the aggregate value.
- The NULL value in the expression column is created by ROLLUP or CUBE as a result of grouping.

GROUPING Function: Example

```
SELECT  department_id DEPTID, job_id JOB,
        SUM(salary),
        GROUPING(department_id) GRP_DEPT,
        GROUPING(job_id) GRP_JOB
FROM    employees
WHERE   department_id < 50
GROUP BY ROLLUP(department_id, job_id);
```

	DEPTID	JOB	SUM(SALARY)	GRP_DEPT	GRP_JOB
1	10	AD_ASST	4400	0	0
2	10	(null)	4400	0	1
3	20	MK_MAN	13000	0	0
4	20	MK_REP	6000	0	0
5	20	(null)	19000	0	1
6	30	PU_MAN	11000	0	0
7	30	PU_CLERK	13900	0	0
8	30	(null)	24900	0	1
9	40	HR_REP	6500	0	0
10	40	(null)	6500	0	1
11	(null)	(null)	54800	1	1

ORACLE

Copyright © 2009, Oracle. All rights reserved.

Example of a GROUPING Function

In the example in the slide, consider the summary value 4400 in the first row (labeled 1). This summary value is the total salary for the job ID of AD_ASST within department 10. To calculate this summary value, both the DEPARTMENT_ID and JOB_ID columns have been taken into account. Thus, a value of 0 is returned for both the GROUPING(department_id) and GROUPING(job_id) expressions.

Consider the summary value 4400 in the second row (labeled 2). This value is the total salary for department 10 and has been calculated by taking into account the DEPARTMENT_ID column; thus, a value of 0 has been returned by GROUPING(department_id). Because the JOB_ID column has not been taken into account to calculate this value, a value of 1 has been returned for GROUPING(job_id). You can observe similar output in the fifth row.

In the last row, consider the summary value 54800 (labeled 3). This is the total salary for those departments whose department ID is less than 50 and all job titles. To calculate this summary value, neither of the DEPARTMENT_ID and JOB_ID columns have been taken into account. Thus, a value of 1 is returned for both the GROUPING(department_id) and GROUPING(job_id) expressions.

GROUPING SETS

- The GROUPING SETS syntax is used to define multiple groupings in the same query.
- All groupings specified in the GROUPING SETS clause are computed and the results of individual groupings are combined with a UNION ALL operation.
- Grouping set efficiency:
 - Only one pass over the base table is required.
 - There is no need to write complex UNION statements.
 - The more elements GROUPING SETS has, the greater is the performance benefit.

ORACLE

Copyright © 2009, Oracle. All rights reserved.

GROUPING SETS

GROUPING SETS is a further extension of the GROUP BY clause that you can use to specify multiple groupings of data. Doing so facilitates efficient aggregation and, therefore, facilitates analysis of data across multiple dimensions.

A single SELECT statement can now be written using GROUPING SETS to specify various groupings (which can also include ROLLUP or CUBE operators), rather than multiple SELECT statements combined by UNION ALL operators. For example:

```
SELECT department_id, job_id, manager_id, AVG(salary)
FROM employees
GROUP BY
GROUPING SETS
((department_id, job_id, manager_id),
 (department_id, manager_id), (job_id, manager_id));
```

This statement calculates aggregates over three groupings:

```
((department_id, job_id, manager_id), (department_id,
manager_id) and (job_id, manager_id)
```

Without this feature, multiple queries combined together with UNION ALL are required to obtain the output of the preceding SELECT statement. A multiquery approach is inefficient because it requires multiple scans of the same data.

GROUPING SETS (continued)

Compare the previous example with the following alternative:

```
SELECT department_id, job_id, manager_id, AVG(salary)
FROM employees
GROUP BY CUBE(department_id, job_id, manager_id);
```

This statement computes all the 8 (2 *2 *2) groupings, though only the (department_id, job_id, manager_id), (department_id, manager_id), and (job_id, manager_id) groups are of interest to you.

Another alternative is the following statement:

```
SELECT department_id, job_id, manager_id, AVG(salary)
FROM employees
GROUP BY department_id, job_id, manager_id
UNION ALL
SELECT department_id, NULL, manager_id, AVG(salary)
FROM employees
GROUP BY department_id, manager_id
UNION ALL
SELECT NULL, job_id, manager_id, AVG(salary)
FROM employees
GROUP BY job_id, manager_id;
```

This statement requires three scans of the base table, which makes it inefficient. CUBE and ROLLUP can be thought of as grouping sets with very specific semantics and results. The following equivalencies show this fact:

CUBE (a, b, c) is equivalent to	GROUPING SETS ((a, b, c), (a, b), (a, c), (b, c), (a), (b), (c), ())
ROLLUP (a, b,c) is equivalent to	GROUPING SETS ((a, b, c), (a, b), (a), ())

GROUPING SETS: Example

```
SELECT  department_id, job_id,
        manager_id, AVG(salary)
FROM    employees
GROUP BY GROUPING SETS
        ((department_id, job_id), (job_id, manager_id));
```

	DEPARTMENT_ID	JOB_ID	MANAGER_ID	AVG(SALARY)
1	(null)	SH_CLERK	122	3200
2	(null)	AC_MGR	101	12000
3	(null)	ST_MAN	100	7280
4	...	(null)	121	2675

1

	DEPARTMENT_ID	JOB_ID	MANAGER_ID	AVG(SALARY)
39	110	AC_MGR	(null)	12000
40	90	AD_PRES	(null)	24000
41	60	IT_PROG	(null)	5760
42	100	FI_MGR	(null)	12000

2

...

ORACLE

Copyright © 2009, Oracle. All rights reserved.

GROUPING SETS: Example

The query in the slide calculates aggregates over two groupings. The table is divided into the following groups:

- Department ID, Job ID
- Job ID, Manager ID

The average salaries for each of these groups are calculated. The result set displays the average salary for each of the two groups.

In the output, the group marked as 1 can be interpreted as the following:

- The average salary of all employees with the SH_CLERK job ID under manager 122 is 3,200.
- The average salary of all employees with the AC_MGR job ID under manager 101 is 12,000, and so on.

The group marked as 2 in the output is interpreted as the following:

- The average salary of all employees with the AC_MGR job ID in department 110 is 12,000.
- The average salary of all employees with the AD_PRES job ID in department 90 is 24,000, and so on.

GROUPING SETS: Example (continued)

The example in the slide can also be written as:

```
SELECT department_id, job_id, NULL as manager_id,  
       AVG(salary) as AVGSAL  
FROM employees  
GROUP BY department_id, job_id  
UNION ALL  
SELECT NULL, job_id, manager_id, avg(salary) as AVGSAL  
FROM employees  
GROUP BY job_id, manager_id;
```

In the absence of an optimizer that looks across query blocks to generate the execution plan, the preceding query would need two scans of the base table, EMPLOYEES. This could be very inefficient. Therefore, the usage of the GROUPING SETS statement is recommended.

Composite Columns

- A composite column is a collection of columns that are treated as a unit.
`ROLLUP (a, (b, c), d)`
- Use parentheses within the `GROUP BY` clause to group columns, so that they are treated as a unit while computing `ROLLUP` or `CUBE` operations.
- When used with `ROLLUP` or `CUBE`, composite columns would require skipping aggregation across certain levels.

ORACLE

Copyright © 2009, Oracle. All rights reserved.

Composite Columns

A composite column is a collection of columns that are treated as a unit during the computation of groupings. You specify the columns in parentheses as in the following statement: `ROLLUP (a, (b, c), d)`

Here, `(b, c)` forms a composite column and is treated as a unit. In general, composite columns are useful in `ROLLUP`, `CUBE`, and `GROUPING SETS`. For example, in `CUBE` or `ROLLUP`, composite columns would require skipping aggregation across certain levels.

That is, `GROUP BY ROLLUP (a, (b, c))` is equivalent to:

```
GROUP BY a, b, c UNION ALL
GROUP BY a UNION ALL
GROUP BY ()
```

Here, `(b, c)` is treated as a unit and `ROLLUP` is not applied across `(b, c)`. It is as though you have an alias—for example, `z` as an alias for `(b, c)`, and the `GROUP BY` expression reduces to: `GROUP BY ROLLUP (a, z)`.

Note: `GROUP BY ()` is typically a `SELECT` statement with `NULL` values for the columns `a` and `b` and only the aggregate function. It is generally used for generating grand totals.

```
SELECT NULL, NULL, aggregate_col
FROM <table_name>
GROUP BY ( );
```

Composite Columns (continued)

Compare this with the normal ROLLUP as in:

```
GROUP BY ROLLUP (a, b, c)
```

This would be:

```
GROUP BY a, b, c UNION ALL
GROUP BY a, b UNION ALL
GROUP BY a UNION ALL
GROUP BY ()
```

Similarly:

```
GROUP BY CUBE ((a, b), c)
```

This would be equivalent to:

```
GROUP BY a, b, c UNION ALL
GROUP BY a, b UNION ALL
GROUP BY c UNION ALL
GROUP BY ()
```

The following table shows the GROUPING SETS specification and the equivalent GROUP BY specification.

GROUPING SETS Statements	Equivalent GROUP BY Statements
GROUP BY GROUPING SETS (a, b, c)	GROUP BY a UNION ALL GROUP BY b UNION ALL GROUP BY c
GROUP BY GROUPING SETS (a, b, (b, c)) (The GROUPING SETS expression has a composite column.)	GROUP BY a UNION ALL GROUP BY b UNION ALL GROUP BY b, c
GROUP BY GROUPING SETS ((a, b, c))	GROUP BY a, b, c
GROUP BY GROUPING SETS (a, (b), ())	GROUP BY a UNION ALL GROUP BY b UNION ALL GROUP BY ()
GROUP BY GROUPING SETS (a, ROLLUP (b, c)) (The GROUPING SETS expression has a composite column.)	GROUP BY a UNION ALL GROUP BY ROLLUP (b, c)

Composite Columns: Example

```
SELECT  department_id, job_id, manager_id,
        SUM(salary)
FROM    employees
GROUP BY ROLLUP( department_id, (job_id, manager_id));
```

1	2	3	4	5
DEPARTMENT_ID	JOB_ID	MANAGER_ID	SUM(SALARY)	
1	(null) SA_REP	149	7000	
2	(null) (null)	(null)	7000	
3	10 AD_ASST	101	4400	
4	10 (null)	(null)	4400	
5	20 MK_MAN	100	13000	2
6	20 MK_REP	201	6000	
7	20 (null)	(null)	19000	
...				
DEPARTMENT_ID	JOB_ID	MANAGER_ID	SUM(SALARY)	
40	100 FL_MGR	101	12000	
41	100 FL_ACCOUNT	108	39600	
42	100 (null)	(null)	51600	3
43	110 AC_MGR	101	12000	
44	110 AC_ACCOUNT	205	8300	
45	110 (null)	(null)	20300	
46	(null) (null)	(null)	691400	4

ORACLE

Copyright © 2009, Oracle. All rights reserved.

Composite Columns: Example

Consider the example:

```
SELECT department_id, job_id, manager_id, SUM(salary)
FROM    employees
GROUP BY ROLLUP( department_id, job_id, manager_id);
```

This query results in the Oracle server computing the following groupings:

- (job_id, manager_id)
- (department_id, job_id, manager_id)
- (department_id)
- Grand total

If you are interested only in specific groups, you cannot limit the calculation to those groupings without using composite columns. With composite columns, this is possible by treating JOB_ID and MANAGER_ID columns as a single unit while rolling up. Columns enclosed in parentheses are treated as a unit while computing ROLLUP and CUBE. This is illustrated in the example in the slide. By enclosing the JOB_ID and MANAGER_ID columns in parentheses, you indicate to the Oracle server to treat JOB_ID and MANAGER_ID as a single unit—that is, a composite column.

Composite Columns: Example (continued)

The example in the slide computes the following groupings:

- (department_id, job_id, manager_id)
- (department_id)
- ()

The example in the slide displays the following:

- Total salary for every job and manager (labeled 1)
- Total salary for every department, job, and manager (labeled 2)
- Total salary for every department (labeled 3)
- Grand total (labeled 4)

The example in the slide can also be written as:

```
SELECT department_id, job_id, manager_id, SUM(salary)
FROM   employees
GROUP  BY department_id, job_id, manager_id
UNION  ALL
SELECT  department_id, TO_CHAR(NULL), TO_NUMBER(NULL),
        SUM(salary)
FROM    employees
GROUP BY department_id
UNION ALL
SELECT  TO_NUMBER(NULL), TO_CHAR(NULL), TO_NUMBER(NULL),
        SUM(salary)
FROM    employees
GROUP BY ();
```

In the absence of an optimizer that looks across query blocks to generate the execution plan, the preceding query would need three scans of the base table, EMPLOYEES. This could be very inefficient. Therefore, the use of composite columns is recommended.

Concatenated Groupings

- Concatenated groupings offer a concise way to generate useful combinations of groupings.
- To specify concatenated grouping sets, you separate multiple grouping sets, ROLLUP and CUBE operations with commas so that the Oracle server combines them into a single GROUP BY clause.
- The result is a cross-product of groupings from each GROUPING SET.

```
GROUP BY GROUPING SETS(a, b), GROUPING SETS(c, d)
```

ORACLE

Copyright © 2009, Oracle. All rights reserved.

Concatenated Groupings

Concatenated groupings offer a concise way to generate useful combinations of groupings. The concatenated groupings are specified by listing multiple grouping sets, CUBEs, and ROLLUPs, and separating them with commas. The following is an example of concatenated grouping sets:

```
GROUP BY GROUPING SETS(a, b), GROUPING SETS(c, d)
```

This SQL example defines the following groupings:

```
(a, c), (a, d), (b, c), (b, d)
```

Concatenation of grouping sets is very helpful for these reasons:

- **Ease of query development:** You need not manually enumerate all groupings.
- **Use by applications:** SQL generated by online analytical processing (OLAP) applications often involves concatenation of grouping sets, with each GROUPING SET defining groupings needed for a dimension.

Concatenated Groupings: Example

```
SELECT  department_id, job_id, manager_id,
        SUM(salary)
FROM    employees
GROUP BY department_id,
        ROLLUP (job_id),
        CUBE (manager_id);
```

	DEPARTMENT_ID	JOB_ID	MANAGER_ID	SUM(SALARY)
1	(null)	SA_REP	149	7000
2		10 AD_ASST	101	4400
3		20 MK_MAN	100	13000
4		20 MK_REP	201	6000
...				
1		90 AD_VP	100	34000
		90 AD PRES	(null)	24000
...				
	(null)	SA_REP	(null)	7000
	10	AD_ASST	(null)	4400
...				
2		110 (null)	101	12000
		110 (null)	205	8300
		110 (null)	(null)	20300

ORACLE

Copyright © 2009, Oracle. All rights reserved.

Concatenated Groupings: Example

The example in the slide results in the following groupings:

- (department_id,job_id,) (1)
- (department_id,manager_id) (2)
- (department_id) (3)

The total salary for each of these groups is calculated.

The following is another example of a concatenated grouping.

```
SELECT department_id, job_id, manager_id, SUM(salary) totalsal
FROM employees
WHERE department_id<60
GROUP BY GROUPING SETS(department_id),
GROUPING SETS (job_id, manager_id);
```

Summary

In this appendix, you should have learned how to use the:

- ROLLUP operation to produce subtotal values
- CUBE operation to produce cross-tabulation values
- GROUPING function to identify the row values created by ROLLUP or CUBE
- GROUPING SETS syntax to define multiple groupings in the same query
- GROUP BY clause to combine expressions in various ways:
 - Composite columns
 - Concatenated grouping sets

ORACLE

Copyright © 2009, Oracle. All rights reserved.

Summary

- ROLLUP and CUBE are extensions of the GROUP BY clause.
- ROLLUP is used to display subtotal and grand total values.
- CUBE is used to display cross-tabulation values.
- The GROUPING function enables you to determine whether a row is an aggregate produced by a CUBE or ROLLUP operator.
- With the GROUPING SETS syntax, you can define multiple groupings in the same query. GROUP BY computes all the groupings specified and combines them with UNION ALL.
- Within the GROUP BY clause, you can combine expressions in various ways:
 - To specify composite columns, you group columns within parentheses so that the Oracle server treats them as a unit while computing ROLLUP or CUBE operations.
 - To specify concatenated grouping sets, you separate multiple grouping sets, ROLLUP, and CUBE operations with commas so that the Oracle server combines them into a single GROUP BY clause. The result is a cross-product of groupings from each grouping set.

Hierarchical Retrieval

ORACLE

Copyright © 2009, Oracle. All rights reserved.

Objectives

After completing this appendix, you should be able to do the following:

- Interpret the concept of a hierarchical query
- Create a tree-structured report
- Format hierarchical data
- Exclude branches from the tree structure

ORACLE

Copyright © 2009, Oracle. All rights reserved.

Objectives

In this appendix, you learn how to use hierarchical queries to create tree-structured reports.

Sample Data from the EMPLOYEES Table

EMPLOYEE_ID	LAST_NAME	JOB_ID	MANAGER_ID
1	100 King	AD_PRES	(null)
2	101 Kochhar	AD_VP	100
3	102 De Haan	AD_VP	100
4	103 Hunold	IT_PROG	102
5	104 Ernst	IT_PROG	103
6	107 Lorentz	IT_PROG	103

...

16	200 Whalen	AD_ASST	101
17	201 Hartstein	MK_MAN	100
18	202 Fay	MK_REP	201
19	205 Higgins	AC_MGR	101
20	206 Gietz	AC_ACCOUNT	205

ORACLE

Copyright © 2009, Oracle. All rights reserved.

Sample Data from the EMPLOYEES Table

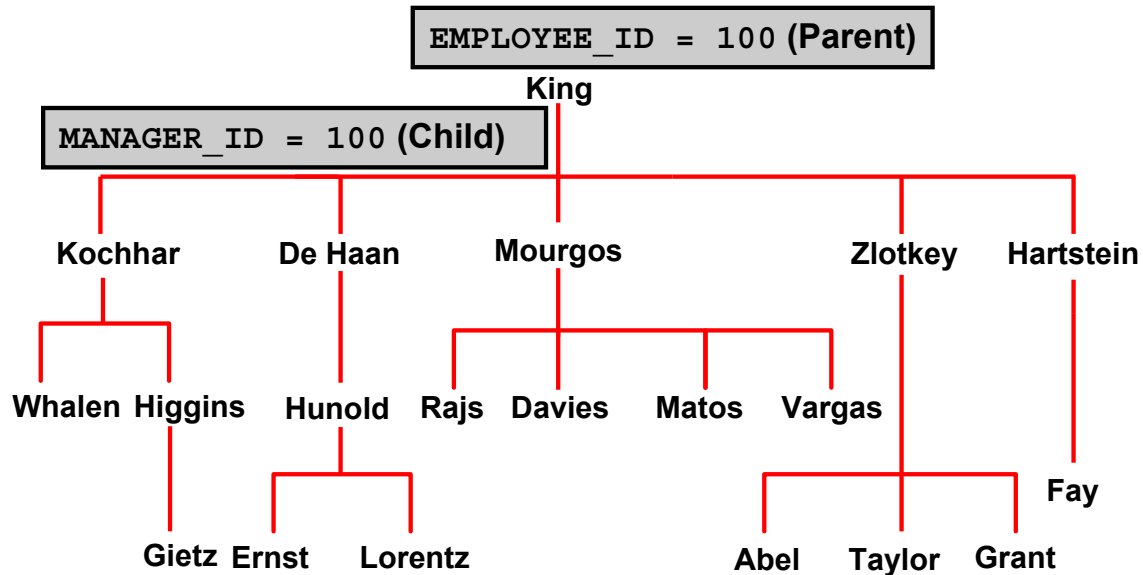
Using hierarchical queries, you can retrieve data based on a natural hierarchical relationship between the rows in a table. A relational database does not store records in a hierarchical way. However, where a hierarchical relationship exists between the rows of a single table, a process called *tree walking* enables the hierarchy to be constructed. A hierarchical query is a method of reporting, with the branches of a tree in a specific order.

Imagine a family tree with the eldest members of the family found close to the base or trunk of the tree and the youngest members representing branches of the tree. Branches can have their own branches, and so on.

A hierarchical query is possible when a relationship exists between rows in a table. For example, in the slide, you see that Kochhar, De Haan, and Hartstein report to `MANAGER_ID` 100, which is King's `EMPLOYEE_ID`.

Note: Hierarchical trees are used in various fields such as human genealogy (family trees), livestock (breeding purposes), corporate management (management hierarchies), manufacturing (product assembly), evolutionary research (species development), and scientific research.

Natural Tree Structure



ORACLE

Copyright © 2009, Oracle. All rights reserved.

Natural Tree Structure

The EMPLOYEES table has a tree structure representing the management reporting line. The hierarchy can be created by looking at the relationship between equivalent values in the EMPLOYEE_ID and MANAGER_ID columns. This relationship can be exploited by joining the table to itself. The MANAGER_ID column contains the employee number of the employee's manager.

The parent-child relationship of a tree structure enables you to control:

- The direction in which the hierarchy is walked
- The starting point inside the hierarchy

Note: The slide displays an inverted tree structure of the management hierarchy of the employees in the EMPLOYEES table.

Hierarchical Queries

```
SELECT [LEVEL], column, expr...  
FROM table  
[WHERE condition(s)]  
[START WITH condition(s)]  
[CONNECT BY PRIOR condition(s)] ;
```

condition:

```
expr comparison_operator expr
```

ORACLE

Copyright © 2009, Oracle. All rights reserved.

Keywords and Clauses

Hierarchical queries can be identified by the presence of the CONNECT BY and START WITH clauses.

In the syntax:

SELECT	Is the standard SELECT clause
LEVEL	For each row returned by a hierarchical query, the LEVEL pseudocolumn returns 1 for a root row, 2 for a child of a root, and so on.
FROM <i>table</i>	Specifies the table, view, or snapshot containing the columns. You can select from only one table.
WHERE	Restricts the rows returned by the query without affecting other rows of the hierarchy
<i>condition</i>	Is a comparison with expressions
START WITH	Specifies the root rows of the hierarchy (where to start). This clause is required for a true hierarchical query.
CONNECT BY	Specifies the columns in which the relationship between parent and child PRIOR rows exist. This clause is required for a hierarchical query.

Walking the Tree

Starting Point

- Specifies the condition that must be met
- Accepts any valid condition

```
START WITH column1 = value
```

Using the EMPLOYEES table, start with the employee whose last name is Kochhar.

```
...START WITH last_name = 'Kochhar'
```

ORACLE

Copyright © 2009, Oracle. All rights reserved.

Oracle University and NETEC, S.A. de C.V. use only.

Walking the Tree

The row or rows to be used as the root of the tree are determined by the START WITH clause. The START WITH clause can contain any valid condition.

Examples

Using the EMPLOYEES table, start with King, the president of the company.

```
... START WITH manager_id IS NULL
```

Using the EMPLOYEES table, start with employee Kochhar. A START WITH condition can contain a subquery.

```
... START WITH employee_id = (SELECT employee_id
                               FROM   employees
                               WHERE  last_name = 'Kochhar')
```

If the START WITH clause is omitted, the tree walk is started with all the rows in the table as root rows.

Note: The CONNECT BY and START WITH clauses are not American National Standards Institute (ANSI) SQL standard.

Walking the Tree

```
CONNECT BY PRIOR column1 = column2
```

Walk from the top down, using the EMPLOYEES table.

```
... CONNECT BY PRIOR employee_id = manager_id
```

Direction

Top down	→	Column1 = Parent Key Column2 = Child Key
Bottom up	→	Column1 = Child Key Column2 = Parent Key

ORACLE

Copyright © 2009, Oracle. All rights reserved.

Walking the Tree (continued)

The direction of the query is determined by the CONNECT BY PRIOR column placement. For top-down, the PRIOR operator refers to the parent row. For bottom-up, the PRIOR operator refers to the child row. To find the child rows of a parent row, the Oracle server evaluates the PRIOR expression for the parent row and the other expressions for each row in the table. Rows for which the condition is true are the child rows of the parent. The Oracle server always selects child rows by evaluating the CONNECT BY condition with respect to a current parent row.

Examples

Walk from the top down using the EMPLOYEES table. Define a hierarchical relationship in which the EMPLOYEE_ID value of the parent row is equal to the MANAGER_ID value of the child row:

```
... CONNECT BY PRIOR employee_id = manager_id
```

Walk from the bottom up using the EMPLOYEES table:

```
... CONNECT BY PRIOR manager_id = employee_id
```

The PRIOR operator does not necessarily need to be coded immediately following CONNECT BY. Thus, the following CONNECT BY PRIOR clause gives the same result as the one in the preceding example:

```
... CONNECT BY employee_id = PRIOR manager_id
```

Note: The CONNECT BY clause cannot contain a subquery.

Walking the Tree: From the Bottom Up

```
SELECT employee_id, last_name, job_id, manager_id
FROM   employees
START WITH employee_id = 101
CONNECT BY PRIOR manager_id = employee_id ;
```

	EMPLOYEE_ID	LAST_NAME	JOB_ID	MANAGER_ID
1	101	Kochhar	AD_VP	100
2	100	King	AD_PRES	(null)

ORACLE

Copyright © 2009, Oracle. All rights reserved.

Walking the Tree: From the Bottom Up

The example in the slide displays a list of managers starting with the employee whose employee ID is 101.

Walking the Tree: From the Top Down

```
SELECT last_name || ' reports to ' ||  
PRIOR last_name "Walk Top Down"  
FROM employees  
START WITH last_name = 'King'  
CONNECT BY PRIOR employee_id = manager_id ;
```

1	King reports to
2	King reports to
3	Kochhar reports to King
4	Greenberg reports to Kochhar
5	Faviet reports to Greenberg

...

105	Grant reports to Zlotkey
106	Johnson reports to Zlotkey
107	Hartstein reports to King
108	Fay reports to Hartstein

ORACLE

Copyright © 2009, Oracle. All rights reserved.

Walking the Tree: From the Top Down

Walking from the top down, display the names of the employees and their manager. Use employee King as the starting point. Print only one column.

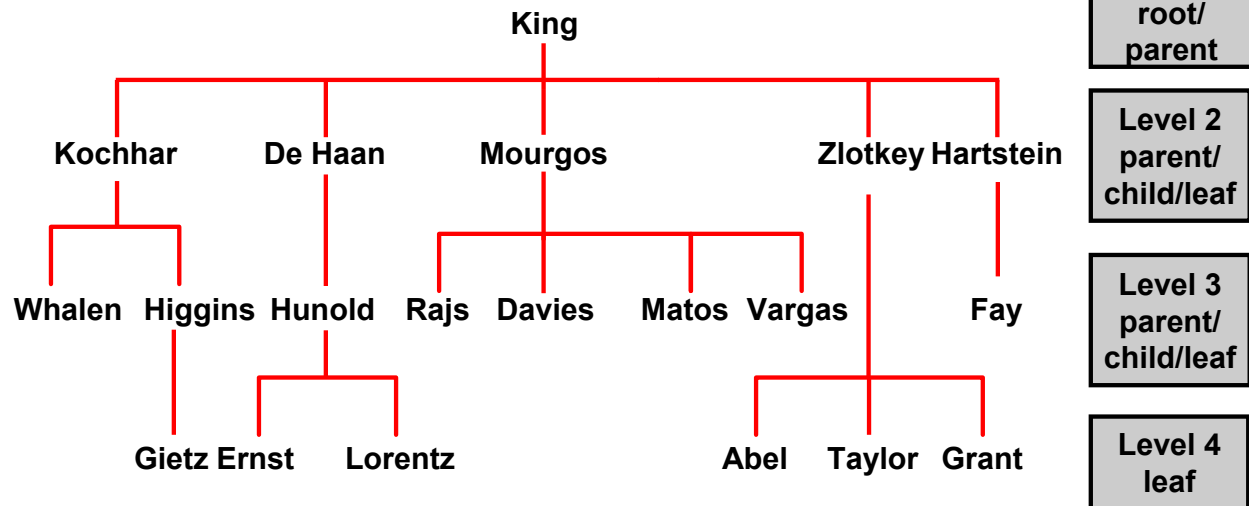
Example

In the following example, EMPLOYEE_ID values are evaluated for the parent row and MANAGER_ID and SALARY values are evaluated for the child rows. The PRIOR operator applies only to the EMPLOYEE_ID value.

```
... CONNECT BY PRIOR employee_id = manager_id  
AND salary > 15000;
```

To qualify as a child row, a row must have a MANAGER_ID value equal to the EMPLOYEE_ID value of the parent row and must have a SALARY value greater than \$15,000.

Ranking Rows with the LEVEL Pseudocolumn



ORACLE

Copyright © 2009, Oracle. All rights reserved.

Ranking Rows with the LEVEL Pseudocolumn

You can explicitly show the rank or level of a row in the hierarchy by using the `LEVEL` pseudocolumn. This will make your report more readable. The forks where one or more branches split away from a larger branch are called nodes, and the very end of a branch is called a leaf or leaf node. The graphic in the slide shows the nodes of the inverted tree with their `LEVEL` values. For example, employee Higgins is a parent and a child, whereas employee Davies is a child and a leaf.

LEVEL Pseudocolumn

Value	Level for Top Down	Level for Bottom up
1	A root node	A root node
2	A child of a root node	The parent of a root node
3	A child of a child, and so on	A parent of a parent, and so on

In the slide, King is the root or parent (`LEVEL = 1`). Kochhar, De Haan, Mourgos, Zlotkey, Hartstein, Higgins, and Hunold are children and also parents (`LEVEL = 2`). Whalen, Rajs, Davies, Matos, Vargas, Gietz, Ernst, Lorentz, Abel, Taylor, Grant, and Fay are children and leaves (`LEVEL = 3` and `LEVEL = 4`).

Note: A *root node* is the highest node within an inverted tree. A *child node* is any nonroot node. A *parent node* is any node that has children. A *leaf node* is any node without children. The number of levels returned by a hierarchical query may be limited by available user memory.

Formatting Hierarchical Reports Using LEVEL and LPAD

Create a report displaying company management levels, beginning with the highest level and indenting each of the following levels.

```
COLUMN org_chart FORMAT A12
SELECT LPAD(last_name, LENGTH(last_name) + (LEVEL*2) - 2, '_')
       AS org_chart
FROM   employees
START WITH first_name='Steven' AND last_name='King'
CONNECT BY PRIOR employee_id=manager_id
```

ORACLE

Copyright © 2009, Oracle. All rights reserved.

Formatting Hierarchical Reports Using LEVEL and LPAD

The nodes in a tree are assigned level numbers from the root. Use the LPAD function in conjunction with the LEVEL pseudocolumn to display a hierarchical report as an indented tree.

In the example in the slide:

- `LPAD(char1, n [, char2])` returns *char1*, left-padded to length *n* with the sequence of characters in *char2*. The argument *n* is the total length of the return value as it is displayed on your terminal screen.
- `LPAD(last_name, LENGTH(last_name) + (LEVEL*2) - 2, '_')` defines the display format
- *char1* is the `LAST_NAME`, *n* the total length of the return value, is length of the `LAST_NAME + (LEVEL*2) - 2`, and *char2* is `'_'`

That is, this tells SQL to take the `LAST_NAME` and left-pad it with the `'_'` character until the length of the resultant string is equal to the value determined by `LENGTH(last_name) + (LEVEL*2) - 2`.

For King, `LEVEL = 1`. Therefore, $(2 * 1) - 2 = 2 - 2 = 0$. So King does not get padded with any `'_'` character and is displayed in column 1.

For Kochhar, `LEVEL = 2`. Therefore, $(2 * 2) - 2 = 4 - 2 = 2$. So Kochhar gets padded with 2 `'_'` characters and is displayed indented.

The rest of the records in the `EMPLOYEES` table are displayed similarly.

Formatting Hierarchical Reports Using LEVEL and LPAD (continued)

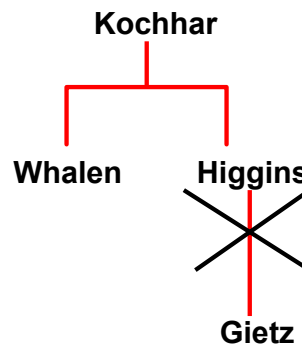
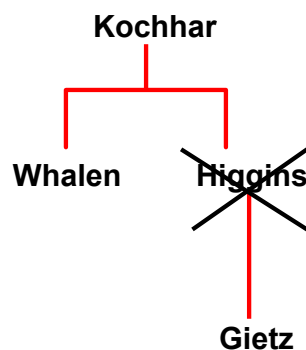
	ORG_CHART
1	King
2	__Kochhar
3	___Greenberg
4	____Faviet
5	____Chen
6	____Sciarra
7	____Urman
8	____Popp
9	___Whalen
10	___Mavris
11	___Baer
12	___Higgins
13	___Gietz
14	__De Haan
15	___Hunold
16	___Ernst
17	___Austin

Pruning Branches

Use the WHERE clause
to eliminate a node.

Use the CONNECT BY clause
to eliminate a branch.

```
WHERE last_name != 'Higgins'
CONNECT BY PRIOR
  employee_id = manager_id
  AND last_name != 'Higgins'
```



ORACLE

Copyright © 2009, Oracle. All rights reserved.

Pruning Branches

You can use the WHERE and CONNECT BY clauses to prune the tree (that is, to control which nodes or rows are displayed). The predicate you use acts as a Boolean condition.

Examples

Starting at the root, walk from the top down, and eliminate employee Higgins in the result, but process the child rows.

```
SELECT department_id, employee_id, last_name, job_id, salary
FROM employees
WHERE last_name != 'Higgins'
START WITH manager_id IS NULL
CONNECT BY PRIOR employee_id = manager_id;
```

Starting at the root, walk from the top down, and eliminate employee Higgins and all child rows.

```
SELECT department_id, employee_id, last_name, job_id, salary
FROM employees
START WITH manager_id IS NULL
CONNECT BY PRIOR employee_id = manager_id
AND last_name != 'Higgins';
```

Summary

In this appendix, you should have learned that you can:

- Use hierarchical queries to view a hierarchical relationship between rows in a table
- Specify the direction and starting point of the query
- Eliminate nodes or branches by pruning

ORACLE

Copyright © 2009, Oracle. All rights reserved.

Summary

You can use hierarchical queries to retrieve data based on a natural hierarchical relationship between rows in a table. The `LEVEL` pseudocolumn counts how far down a hierarchical tree you have traveled. You can specify the direction of the query using the `CONNECT BY PRIOR` clause. You can specify the starting point using the `START WITH` clause. You can use the `WHERE` and `CONNECT BY` clauses to prune the tree branches.

II

Writing Advanced Scripts

ORACLE

Copyright © 2009, Oracle. All rights reserved.

Objectives

After completing this appendix, you should be able to do the following:

- Describe the type of problems that are solved by using SQL to generate SQL
- Write a script that generates a script of DROP TABLE statements
- Write a script that generates a script of INSERT INTO statements

ORACLE

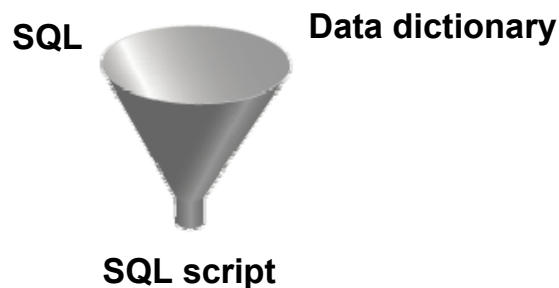
Copyright © 2009, Oracle. All rights reserved.

Objectives

In this appendix, you learn how to write a SQL script to generate a SQL script.

Using SQL to Generate SQL

- SQL can be used to generate scripts in SQL.
- The data dictionary is:
 - A collection of tables and views that contain database information
 - Created and maintained by the Oracle server



ORACLE

Copyright © 2009, Oracle. All rights reserved.

Using SQL to Generate SQL

SQL can be a powerful tool to generate other SQL statements. In most cases, this involves writing a script file. You can use SQL from SQL to:

- Avoid repetitive coding
- Access information from the data dictionary
- Drop or re-create database objects
- Generate dynamic predicates that contain run-time parameters

The examples used in this appendix involve selecting information from the data dictionary. The data dictionary is a collection of tables and views that contain information about the database. This collection is created and maintained by the Oracle server. All data dictionary tables are owned by the SYS user. Information stored in the data dictionary includes names of Oracle server users, privileges granted to users, database object names, table constraints, and audit information. There are four categories of data dictionary views. Each category has a distinct prefix that reflects its intended use.

Prefix	Description
USER_	Contains details of objects owned by the user
ALL_	Contains details of objects to which the user has been granted access rights, in addition to objects owned by the user
DBA_	Contains details of users with DBA privileges to access any object in the database
VS_	Stores information about database server performance and locking; available only to the DBA

Creating a Basic Script

```
SELECT 'CREATE TABLE ' || table_name ||  
      ' _test ' || 'AS SELECT * FROM '  
      || table_name || ' WHERE 1=2;'  
      AS "Create Table Script"  
FROM   user tables;
```

	Create Table Script
1	CREATE TABLE REGIONS_test AS SELECT * FROM REGIONS WHERE 1=2;
2	CREATE TABLE LOCATIONS_test AS SELECT * FROM LOCATIONS WHERE 1=2;
3	CREATE TABLE DEPARTMENTS_test AS SELECT * FROM DEPARTMENTS WHERE 1=2;
4	CREATE TABLE JOBS_test AS SELECT * FROM JOBS WHERE 1=2;
5	CREATE TABLE EMPLOYEES_test AS SELECT * FROM EMPLOYEES WHERE 1=2;
6	CREATE TABLE JOB_HISTORY_test AS SELECT * FROM JOB_HISTORY WHERE 1=2;

ORACLE

Copyright © 2009, Oracle. All rights reserved.

A Basic Script

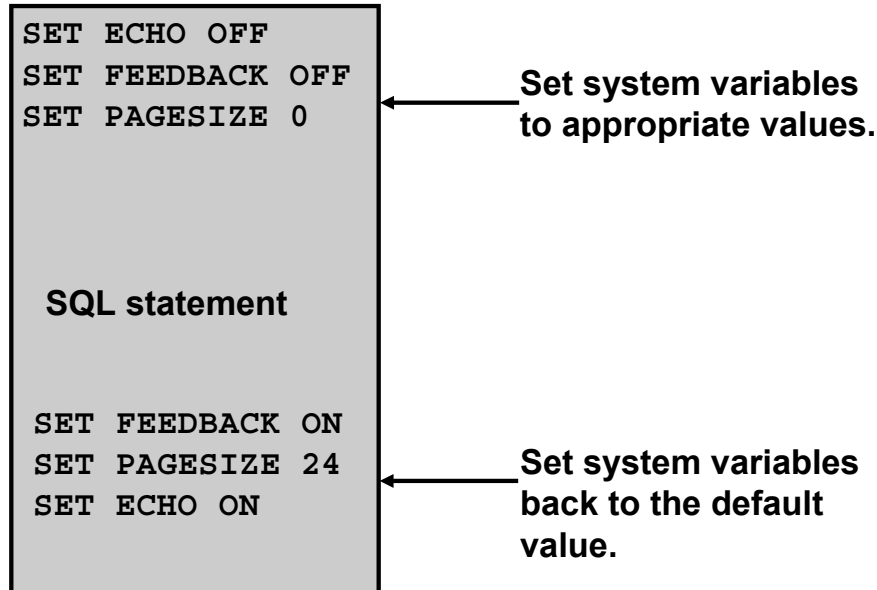
The example in the slide produces a report with CREATE TABLE statements from every table you own. Each CREATE TABLE statement produced in the report includes the syntax to create a table using the table name with a suffix of _test and having only the structure of the corresponding existing table. The old table name is obtained from the TABLE_NAME column of the data dictionary view USER_TABLES.

The next step is to enhance the report to automate the process.

Note: You can query the data dictionary tables to view various database objects that you own. The data dictionary views frequently used include:

- USER_TABLES: Displays description of the user's own tables
- USER_OBJECTS: Displays all the objects owned by the user
- USER_TAB_PRIVS_MADE: Displays all grants on objects owned by the user
- USER_COL_PRIVS_MADE: Displays all grants on columns of objects owned by the user

Controlling the Environment



ORACLE

Copyright © 2009, Oracle. All rights reserved.

Controlling the Environment

To execute the SQL statements that are generated, you must capture them in a file that can then be run. You must also plan to clean up the output that is generated and make sure that you suppress elements such as headings, feedback messages, top titles, and so on. In SQL Developer, you can save these statements to a script. To save the contents of the Enter SQL Statement box, click the Save icon or use the **File > Save** menu item. Alternatively, you can right-click in the Enter SQL Statement box and select the Save File option from the drop-down menu.

Note: Some of the SQL*Plus statements are not supported by SQL Worksheet. For the complete list of SQL*Plus statements that are supported, and not supported by SQL Worksheet, refer to the topic titled *SQL*Plus Statements Supported and Not Supported in SQL Worksheet* in the SQL Developer online Help.

The Complete Picture

```
SET ECHO OFF
SET FEEDBACK OFF
SET PAGESIZE 0

SELECT 'DROP TABLE ' || object_name || ';'
FROM   user_objects
WHERE  object_type = 'TABLE'
/

SET FEEDBACK ON
SET PAGESIZE 24
SET ECHO ON
```

ORACLE

Copyright © 2009, Oracle. All rights reserved.

The Complete Picture

The output of the command in the slide is saved into a file called `dropem.sql` in SQL Developer. To save the output into a file in SQL Developer, you use the Save File option under the Script Output pane. The `dropem.sql` file contains the following data. This file can now be started from SQL Developer by locating the script file, loading it, and executing it.

	'DROPTABLE' OBJECT_NAME ';'
1	DROP TABLE REGIONS;
2	DROP TABLE COUNTRIES;
3	DROP TABLE LOCATIONS;
4	DROP TABLE DEPARTMENTS;
5	DROP TABLE JOBS;
6	DROP TABLE EMPLOYEES;
7	DROP TABLE JOB_HISTORY;
8	DROP TABLE JOB_GRADES;

Dumping the Contents of a Table to a File

```
SET HEADING OFF ECHO OFF FEEDBACK OFF
SET PAGESIZE 0

SELECT
  'INSERT INTO departments_test VALUES
  (' || department_id || ', ' || department_name ||
  ', ' || location_id || ');'
  AS "Insert Statements Script"
FROM   departments
/

SET PAGESIZE 24
SET HEADING ON ECHO ON FEEDBACK ON
```

ORACLE

Copyright © 2009, Oracle. All rights reserved.

Dumping Table Contents to a File

Sometimes, it is useful to have the values for the rows of a table in a text file in the format of an INSERT INTO VALUES statement. This script can be run to populate the table in case the table has been dropped accidentally.

The example in the slide produces INSERT statements for the DEPARTMENTS_TEST table, captured in the data.sql file using the Save File option in SQL Developer.

The contents of the data.sql script file are as follows:

```
INSERT INTO departments_test VALUES
  (10, 'Administration', 1700);
INSERT INTO departments_test VALUES
  (20, 'Marketing', 1800);
INSERT INTO departments_test VALUES
  (50, 'Shipping', 1500);
INSERT INTO departments_test VALUES
  (60, 'IT', 1400);
...
```

Dumping the Contents of a Table to a File

Source	Result
<code>' ' 'X' ' '</code>	<code>'X'</code>
<code>' ' ' ' '</code>	<code>'</code>
<code>' ' ' ' department_name ' ' ' ' '</code>	<code>'Administration'</code>
<code>' ' ', ' ' ' '</code>	<code>','</code>
<code>' ' ') ; ' '</code>	<code>');</code>

ORACLE

Copyright © 2009, Oracle. All rights reserved.

Dumping Table Contents to a File (continued)

You may have noticed the large number of single quotation marks in the previous slide. A set of four single quotation marks produces one single quotation mark in the final statement. Also remember that character and date values must be enclosed within quotation marks.

Within a string, to display one quotation mark, you need to prefix it with another single quotation mark. For example, in the fifth example in the slide, the surrounding quotation marks are for the entire string. The second quotation mark acts as a prefix to display the third quotation mark. Thus, the result is a single quotation mark followed by the parenthesis, followed by the semicolon.

Generating a Dynamic Predicate

```
COLUMN my_col NEW_VALUE dyn_where_clause

SELECT DECODE('&deptno', null,
DECODE ('&hiredate', null, ' ',
'WHERE hire_date=TO_DATE('' || '&hiredate'', 'DD-MON-YYYY'')),
DECODE ('&hiredate', null,
'WHERE department_id = ' || '&deptno',
'WHERE department_id = ' || '&deptno' ||
' AND hire_date = TO_DATE('' || '&hiredate'', 'DD-MON-YYYY''))
AS my_col FROM dual;
```

```
SELECT last_name FROM employees &dyn_where_clause;
```

ORACLE

Copyright © 2009, Oracle. All rights reserved.

Generating a Dynamic Predicate

The example in the slide generates a `SELECT` statement that retrieves data of all employees in a department who were hired on a specific day. The script generates the `WHERE` clause dynamically.

Note: After the user variable is in place, you must use the `UNDEFINE` command to delete it.

The first `SELECT` statement prompts you to enter the department number. If you do not enter any department number, the department number is treated as null by the `DECODE` function, and the user is then prompted for the hire date. If you do not enter any hire date, the hire date is treated as null by the `DECODE` function and the dynamic `WHERE` clause that is generated is also a null, which causes the second `SELECT` statement to retrieve all the rows from the `EMPLOYEES` table.

Note: The `NEW_V[ALUE]` variable specifies a variable to hold a column value. You can reference the variable in `TTITLE` commands. Use `NEW_VALUE` to display column values or the date in the top title. You must include the column in a `BREAK` command with the `SKIP PAGE` action. The variable name cannot contain a pound sign (`#`). `NEW_VALUE` is useful for master/detail reports in which there is a new master record for each page.

Generating a Dynamic Predicate (continued)

Note: Here, the hire date must be entered in the DD-MON-YYYY format.

The SELECT statement in the slide can be interpreted as follows:

```

IF    (<<deptno>> is not entered) THEN
      IF (<<hiredate>> is not entered) THEN
          return empty string
      ELSE
          return the string 'WHERE hire_date =
TO_DATE('<<hiredate>>', 'DD-MON-YYYY')'
      ELSE
          IF (<<hiredate>> is not entered) THEN
              return the string 'WHERE department_id =
<<deptno>> entered'
          ELSE
              return the string 'WHERE department_id =
<<deptno>> entered
                                AND hire_date =
                                TO_DATE(' <<hiredate>>', 'DD-MON-YYYY')'
      END IF

```

The returned string becomes the value of the DYN_WHERE_CLAUSE variable, which will be used in the second SELECT statement.

Note: Use SQL*Plus for these examples.

When the first example in the slide is executed, the user is prompted for the values for DEPTNO and HIREDATE:

Enter value for deptno: 10

Enter value for hiredate: 17-SEP-1987

The following value for MY_COL is generated:

```

MY_COL
-----
WHERE department_id = 10 AND hire_date = TO_DATE('27-SEP-1987','DD-MON-YYYY')

```

When the second example in the slide is executed, the following output is generated:

```

LAST_NAME
-----
Whalen

```

Summary

In this appendix, you should have learned that:

- You can write a SQL script to generate another SQL script
- Script files often use the data dictionary
- You can capture the output in a file

ORACLE

Copyright © 2009, Oracle. All rights reserved.

Summary

SQL can be used to generate SQL scripts. These scripts can be used to avoid repetitive coding, drop or re-create objects, get help from the data dictionary, and generate dynamic predicates that contain run-time parameters.

Oracle Database Architectural Components



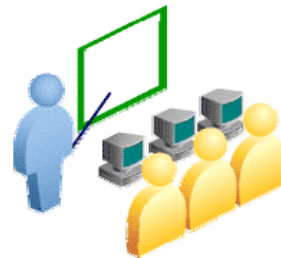
ORACLE

Copyright © 2009, Oracle. All rights reserved.

Objectives

After completing this appendix, you should be able to do the following:

- List the major database architectural components
- Describe the background processes
- Explain the memory structures
- Correlate the logical and physical storage structures



ORACLE

Copyright © 2009, Oracle. All rights reserved.

Objectives

This appendix provides an overview of the Oracle Database architecture. You learn about the physical and logical structures and various components of Oracle Database and their functions.

Oracle Database Architecture: Overview

The Oracle Relational Database Management System (RDBMS) is a database management system that provides an open, comprehensive, integrated approach to information management.



ORACLE

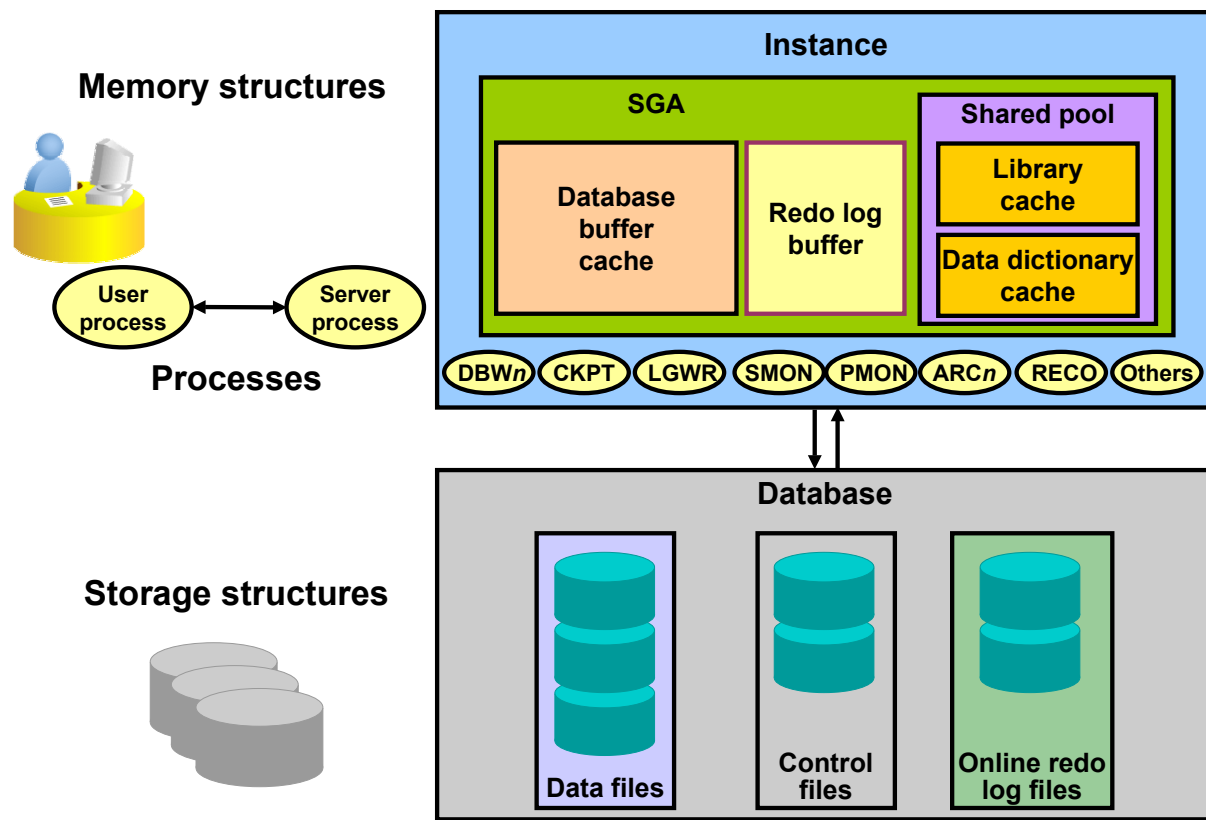
Copyright © 2009, Oracle. All rights reserved.

Oracle Database Architecture: Overview

A database is a collection of data treated as a unit. The purpose of a database is to store and retrieve related information.

An Oracle database reliably manages a large amount of data in a multiuser environment so that many users can concurrently access the same data. This is accomplished while delivering high performance. At the same time, it prevents unauthorized access and provides efficient solutions for failure recovery.

Oracle Database Server Structures



Copyright © 2009, Oracle. All rights reserved.

ORACLE

Oracle Database Server Structures

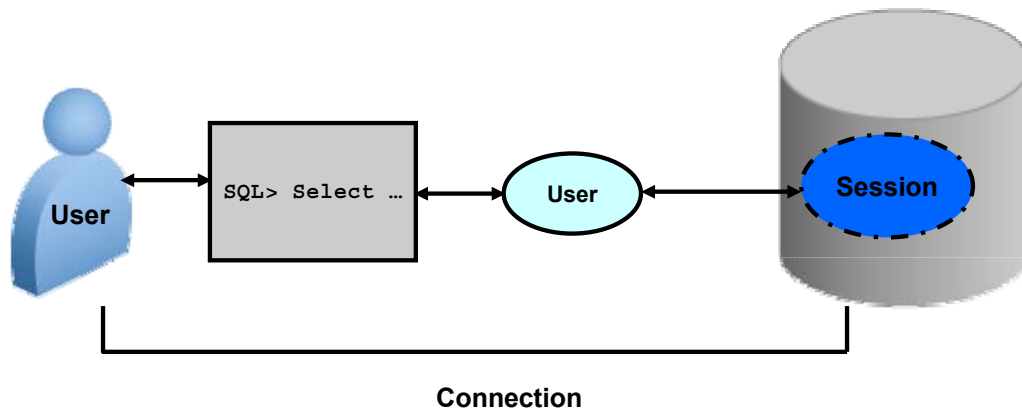
The Oracle Database consists of two main components—the instance and the database.

- The instance consists of the System Global Area (SGA), which is a collection of memory structures, and the background processes that perform tasks within the database. Every time an instance is started, the SGA is allocated and the background processes are started.
- The database consists of both physical structures and logical structures. Because the physical and logical structures are separate, the physical storage of data can be managed without affecting access to logical storage structures. The physical storage structures include:
 - The control files where the database configuration is stored
 - The redo log files that have information required for database recovery
 - The data files where all data is stored

An Oracle instance uses memory structures and processes to manage and access the database storage structures. All memory structures exist in the main memory of the computers that constitute the database server. Processes are jobs that work in the memory of these computers. A process is defined as a “thread of control” or a mechanism in an operating system that can run a series of steps.

Connecting to the Database

- Connection: Communication pathway between a user process and a database instance
- Session: A specific connection of a user to a database instance through a user process



ORACLE

Copyright © 2009, Oracle. All rights reserved.

Connecting to the Database

To access information in the database, the user needs to connect to the database using a tool (such as SQL*Plus). After the user establishes connection, a session is created for the user. Connection and session are closely related to user process but are very different in meaning.

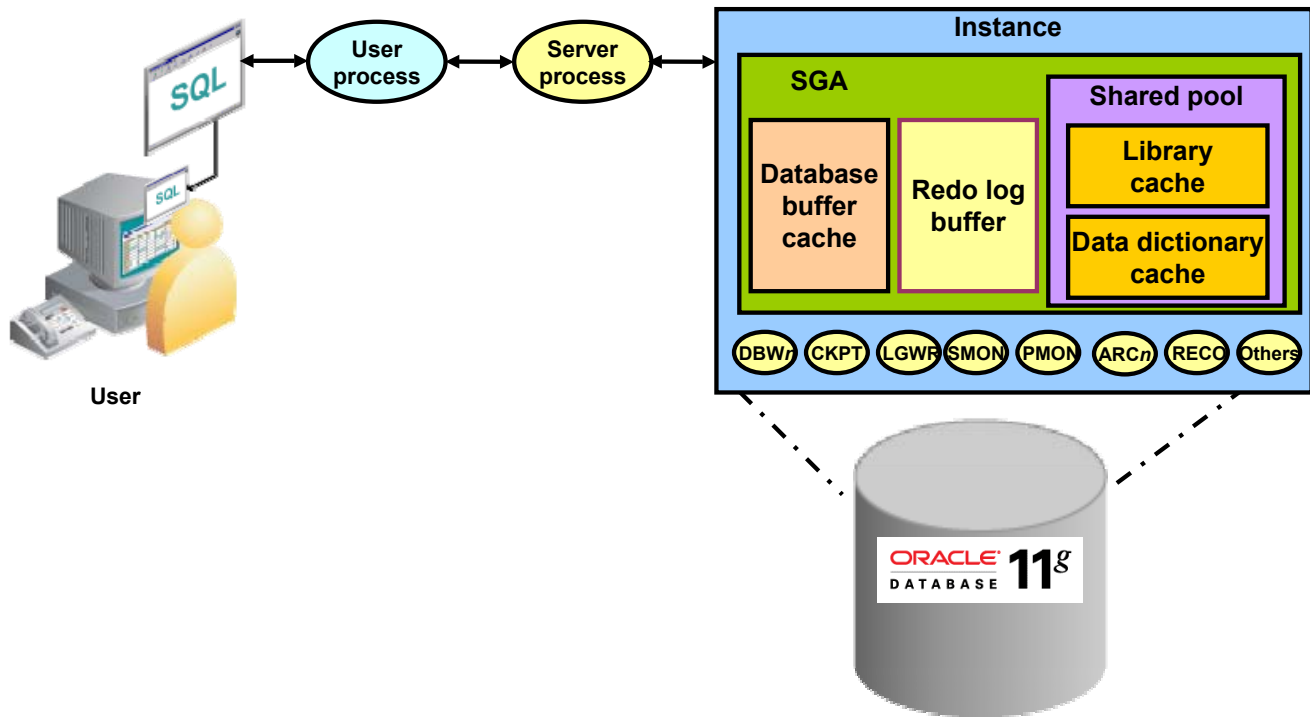
A connection is a communication pathway between a user process and an Oracle Database instance. A communication pathway is established using available interprocess communication mechanisms or network software (when different computers run the database application and Oracle Database, and communicate through a network).

A session represents the state of a current user login to the database instance. For example, when a user starts SQL*Plus, the user must provide a valid username and password, and then a session is established for that user. A session lasts from the time the user connects until the time the user disconnects or exits the database application.

In the case of a dedicated connection, the session is serviced by a permanent dedicated process. In the case of a shared connection, the session is serviced by an available server process selected from a pool, either by the middle tier or by Oracle shared server architecture.

Multiple sessions can be created and exist concurrently for a single Oracle Database user using the same username, but through different applications, or multiple invocations of the same application.

Interacting with an Oracle Database



Copyright © 2009, Oracle. All rights reserved.

ORACLE

Interacting with an Oracle Database

The following example describes Oracle Database operations at the most basic level. It illustrates an Oracle Database configuration where the user and associated server process are on separate computers, connected through a network.

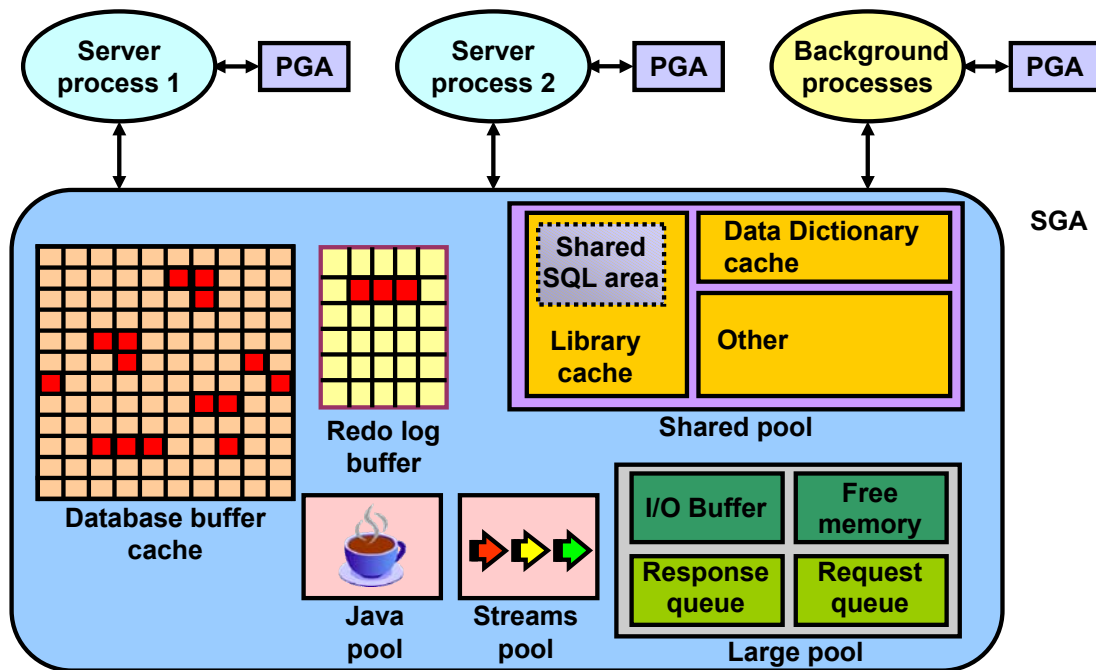
1. An instance has started on a node where Oracle Database is installed, often called the host or database server.
2. A user starts an application spawning a user process. The application attempts to establish a connection to the server. (The connection may be local, client server, or a three-tier connection from a middle tier.)
3. The server runs a listener that has the appropriate Oracle Net Services handler. The server detects the connection request from the application and creates a dedicated server process on behalf of the user process.
4. The user runs a DML-type SQL statement and commits the transaction. For example, the user changes the address of a customer in a table and commits the change.
5. The server process receives the statement and checks the shared pool (an SGA component) for any shared SQL area that contains a similar SQL statement. If a shared SQL area is found, the server process checks the user's access privileges to the requested data, and the existing shared SQL area is used to process the statement. If not, a new shared SQL area is allocated for the statement, so it can be parsed and processed.

Interacting with an Oracle Database (continued)

6. The server process retrieves any necessary data values, either from the actual data file (in which the table is stored) or those cached in the SGA.
7. The server process modifies data in the SGA. Because the transaction is committed, the log writer process (LGWR) immediately records the transaction in the redo log file. The database writer process (DBW n) writes modified blocks permanently to disk when doing so is efficient.
8. If the transaction is successful, the server process sends a message across the network to the application. If it is not successful, an error message is transmitted.
9. Throughout this entire procedure, the other background processes run, watching for conditions that require intervention. In addition, the database server manages other users' transactions and prevents contention between transactions that request the same data.

Oracle Memory Architecture

DB structures
→ **Memory**
- Process
- Storage



ORACLE

Copyright © 2009, Oracle. All rights reserved.

Oracle Memory Structures

Oracle Database creates and uses memory structures for various purposes. For example, memory stores program code being run, data shared among users, and private data areas for each connected user.

Two basic memory structures are associated with an instance:

- The System Global Area (SGA) is a group of shared memory structures, known as SGA components, that contain data and control information for one Oracle Database instance. The SGA is shared by all server and background processes. Examples of data stored in the SGA include cached data blocks and shared SQL areas.
- The Program Global Areas (PGA) are memory regions that contain data and control information for a server or background process. A PGA is nonshared memory created by Oracle Database when a server or background process is started. Access to the PGA is exclusive to the server process. Each server process and background process has its own PGA.

Oracle Memory Structures (continued)

The SGA is the memory area that contains data and control information for the instance. The SGA includes the following data structures:

- **Database buffer cache:** Caches blocks of data retrieved from the database
- **Redo Log buffer:** Caches redo information (used for instance recovery) until it can be written to the physical redo log files stored on the disk
- **Shared pool:** Caches various constructs that can be shared among users
- **Large pool:** Is an optional area that provides large memory allocations for certain large processes, such as Oracle backup and recovery operations, and input/output (I/O) server processes
- **Java pool:** Is used for all session-specific Java code and data within the Java Virtual Machine (JVM)
- **Streams pool:** Is used by Oracle Streams to store information required by capture and apply

When you start the instance by using Enterprise Manager or SQL*Plus, the amount of memory allocated for the SGA is displayed.

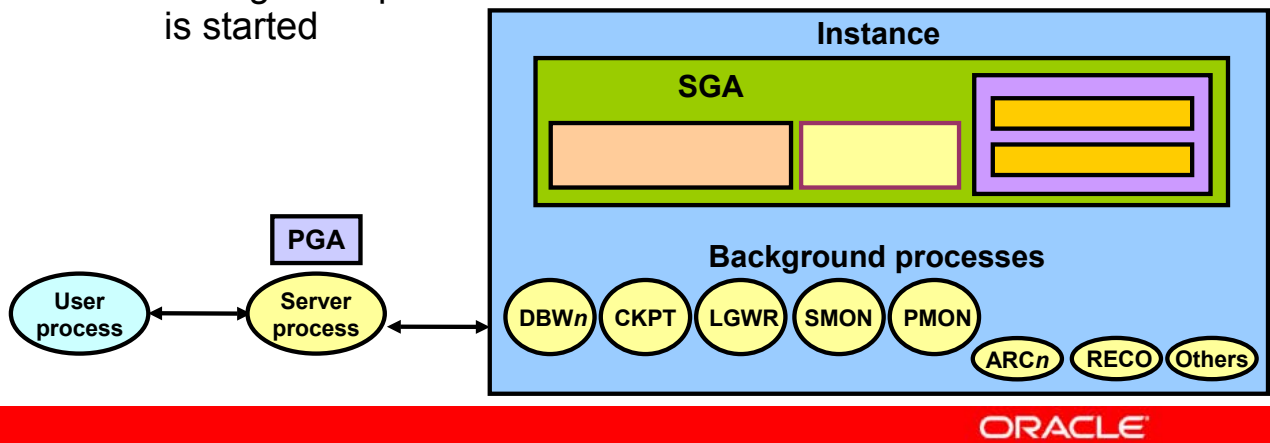
With the dynamic SGA infrastructure, the size of the database buffer cache, the shared pool, the large pool, the Java pool, and the Streams pool changes without shutting down the instance.

Oracle Database uses initialization parameters to create and configure memory structures. For example, the `SGA_TARGET` parameter specifies the total size of the SGA components. If you set `SGA_TARGET` to 0, Automatic Shared Memory Management is disabled.

Process Architecture

DB structures
 - Memory
 → **Process**
 - Storage

- User process:
 - Is started when a database user or a batch process connects to the Oracle Database
- Database processes:
 - Server process: Connects to the Oracle instance and is started when a user establishes a session
 - Background processes: Are started when an Oracle instance is started



Copyright © 2009, Oracle. All rights reserved.

Process Architecture

The processes in an Oracle Database server can be categorized into two major groups:

- User processes that run the application or Oracle tool code
- Oracle Database processes that run the Oracle database server code. These include server processes and background processes.

When a user runs an application program or an Oracle tool such as SQL*Plus, Oracle Database creates a *user process* to run the user's application. The Oracle Database also creates a *server process* to execute the commands issued by the user process. In addition, the Oracle server also has a set of *background processes* for an instance that interact with each other and with the operating system to manage the memory structures and asynchronously perform I/O to write data to disk, and perform other required tasks.

The process structure varies for different Oracle Database configurations, depending on the operating system and the choice of Oracle Database options. The code for connected users can be configured as a dedicated server or a shared server.

- With dedicated server, for each user, the database application is run by a user process, which is served by a dedicated server process that executes Oracle database server code.
- A shared server eliminates the need for a dedicated server process for each connection. A dispatcher directs multiple incoming network session requests to a pool of shared server processes. A shared server process serves any client request.

Process Architecture (continued)

Server Processes

Oracle Database creates server processes to handle the requests of user processes connected to the instance. In some situations when the application and Oracle Database operate on the same computer, it is possible to combine the user process and the corresponding server process into a single process to reduce system overhead. However, when the application and Oracle Database operate on different computers, a user process always communicates with Oracle Database through a separate server process.

Server processes created on behalf of each user's application can perform one or more of the following:

- Parse and run SQL statements issued through the application.
- Read necessary data blocks from data files on disk into the shared database buffers of the SGA, if the blocks are not already present in the SGA.
- Return results in such a way that the application can process the information.

Background Processes

To maximize performance and accommodate many users, a multiprocess Oracle Database system uses some additional Oracle Database processes called background processes. An Oracle Database instance can have many background processes.

The following background processes are required for a successful startup of the database instance:

- Database writer (DBWn)
- Log writer (LGWR)
- Checkpoint (CKPT)
- System monitor (SMON)
- Process monitor (PMON)

The following background processes are a few examples of optional background processes that can be started if required:

- Recoverer (RECO)
- Job queue
- Archiver (ARCn)
- Queue monitor (QMNn)

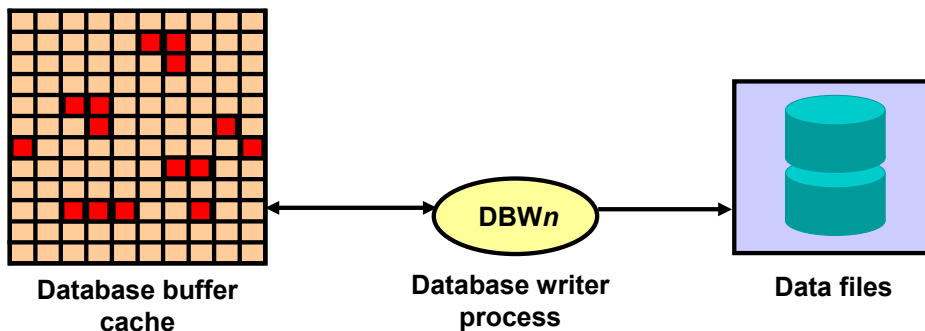
Other background processes may be found in more advanced configurations such as Real Application Clusters (RAC). See the V\$BGPROCESS view for more information about the background processes.

On many operating systems, background processes are created automatically when an instance is started.

Database Writer Process

Writes modified (dirty) buffers in the database buffer cache to disk:

- Asynchronously while performing other processing
- Periodically to advance the checkpoint



ORACLE

Copyright © 2009, Oracle. All rights reserved.

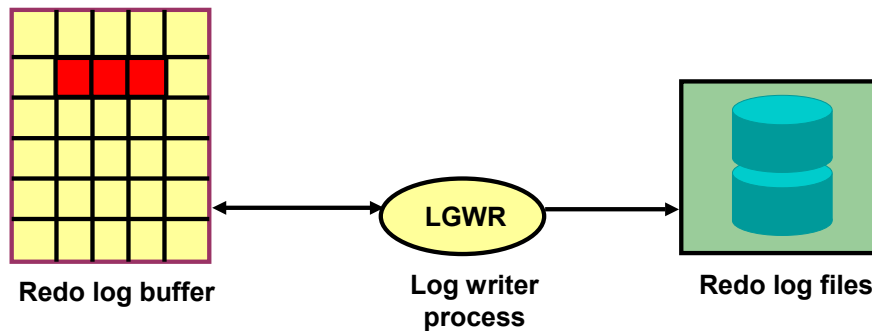
Database Writer Process

The database writer (DBW n) process writes the contents of buffers to data files. The DBW n processes are responsible for writing modified (dirty) buffers in the database buffer cache to disk. Although one database writer process (DBW0) is adequate for most systems, you can configure additional processes (DBW1 through DBW9 and DBWa through DBWj) to improve write performance if your system modifies data heavily. These additional DBW n processes are not useful on uniprocessor systems.

When a buffer in the database buffer cache is modified, it is marked “dirty” and is added to the LRUW list of dirty buffers that is kept in system change number (SCN) order, thereby matching the order of Redo corresponding to these changed buffers that is written to the Redo logs. When the number of available buffers in the buffer cache falls below an internal threshold such that server processes find it difficult to obtain available buffers, DBW n writes dirty buffers to the data files in the order that they were modified by following the order of the LRUW list.

Log Writer Process

- Writes the redo log buffer to a redo log file on disk
- LGWR writes:
 - A process commits a transaction
 - When the redo log buffer is one-third full
 - Before a DBWn process writes modified buffers to disk



ORACLE

Copyright © 2009, Oracle. All rights reserved.

Log Writer Process

The log writer (LGWR) process is responsible for redo log buffer management by writing the redo log buffer entries to a redo log file on disk. LGWR writes all redo entries that have been copied into the buffer since the last time it wrote.

The redo log buffer is a circular buffer. When LGWR writes redo entries from the redo log buffer to a redo log file, server processes can then copy new entries over the entries in the redo log buffer that have been written to disk. LGWR normally writes fast enough to ensure that space is always available in the buffer for new entries, even when access to the redo log is heavy.

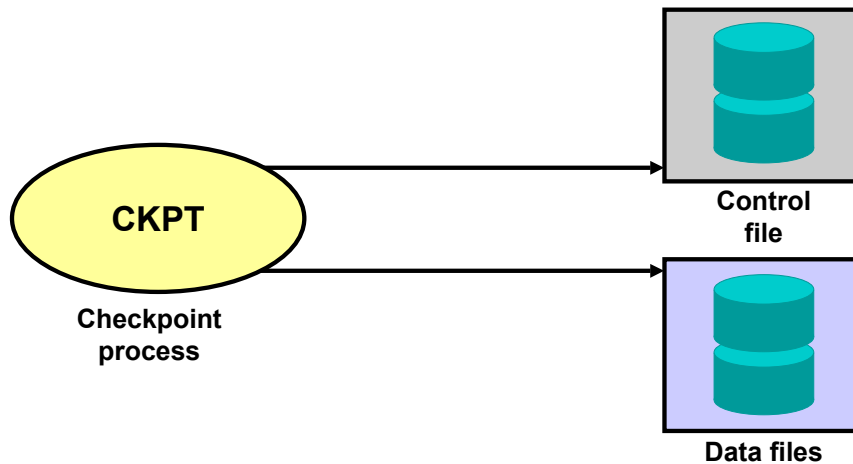
LGWR writes one contiguous portion of the buffer to disk. LGWR writes:

- When a user process commits a transaction
- When the redo log buffer is one-third full
- Before a DBWn process writes modified buffers to disk, if necessary

Checkpoint Process

Records checkpoint information in:

- The control file
- Each datafile header



ORACLE

Copyright © 2009, Oracle. All rights reserved.

Checkpoint Process

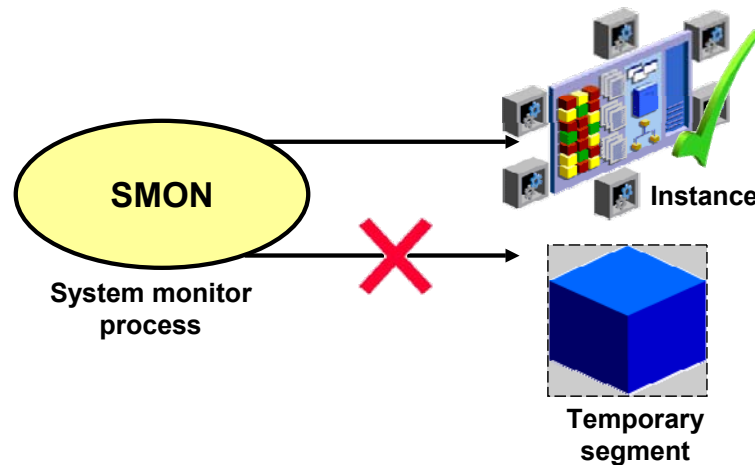
A checkpoint is a data structure that defines an SCN in the redo thread of a database. Checkpoints are recorded in the control file and each data file header, and are a crucial element of recovery.

When a checkpoint occurs, Oracle Database must update the headers of all data files to record the details of the checkpoint. This is done by the CKPT process. The CKPT process does not write blocks to disk; DBWR always performs that work. The SCNs recorded in the file headers guarantee that all the changes made to database blocks before that SCN have been written to disk.

The statistic DBWR checkpoints displayed by the `SYSTEM_STATISTICS` monitor in Oracle Enterprise Manager indicate the number of checkpoint requests completed.

System Monitor Process

- Performs recovery at instance startup
- Cleans up unused temporary segments



ORACLE

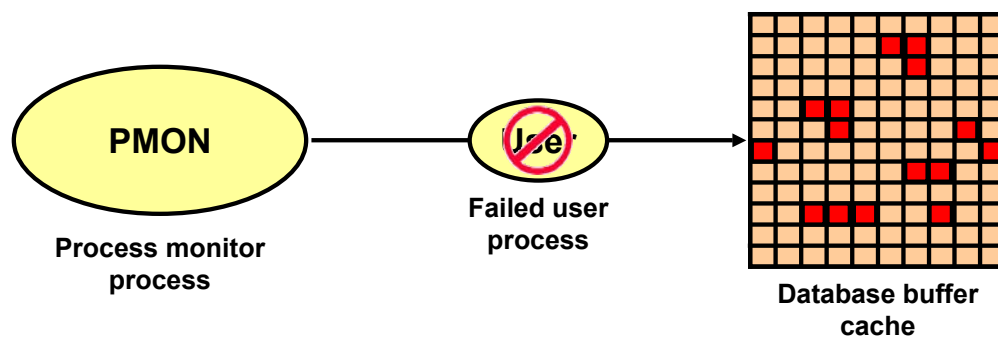
Copyright © 2009, Oracle. All rights reserved.

System Monitor Process

The system monitor (SMON) process performs recovery, if necessary, at instance startup. SMON is also responsible for cleaning up temporary segments that are no longer in use. If any terminated transactions were skipped during instance recovery because of file-read or offline errors, SMON recovers them when the tablespace or file is brought back online. SMON checks regularly to see whether it is needed. Other processes can call SMON if they detect a need for it.

Process Monitor Process

- Performs process recovery when a user process fails:
 - Cleans up the database buffer cache
 - Frees resources used by the user process
- Monitors sessions for idle session timeout
- Dynamically registers database services with listeners



ORACLE

Copyright © 2009, Oracle. All rights reserved.

Process Monitor Process

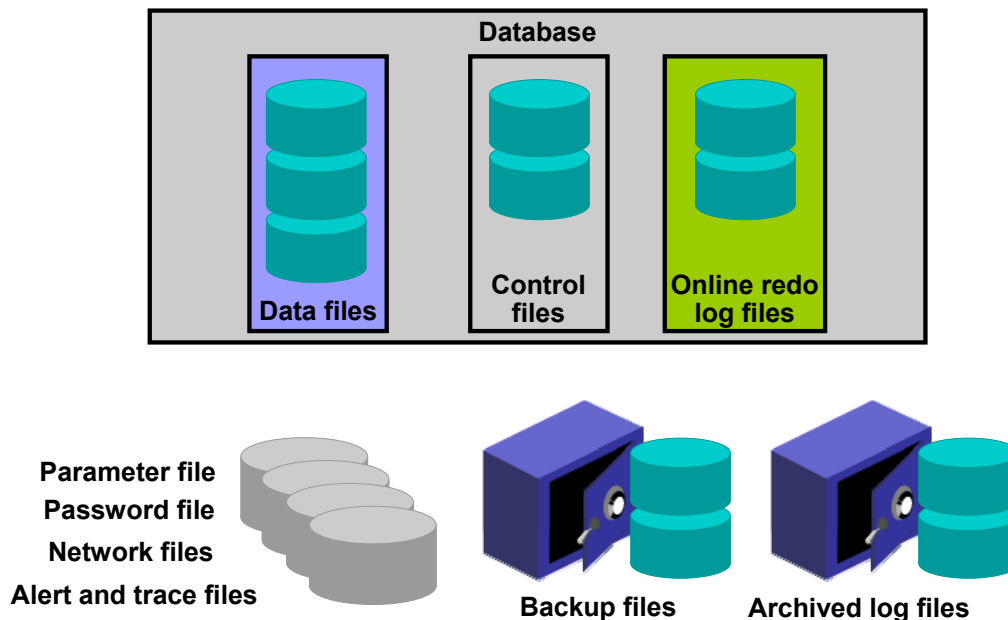
The process monitor (PMON) performs process recovery when a user process fails. PMON is responsible for cleaning up the database buffer cache and freeing resources that the user process was using. For example, it resets the status of the active transaction table, releases locks, and removes the process ID from the list of active processes.

PMON periodically checks the status of dispatcher and server processes, and restarts any that have stopped running (but not any that Oracle Database has terminated intentionally). PMON also registers information about the instance and dispatcher processes with the network listener. Like SMON, PMON checks regularly to see whether it is needed and can be called if another process detects the need for it.

Oracle Database Storage Architecture

DB structures

- Memory
- Process
- **Storage**



Copyright © 2009, Oracle. All rights reserved.

Oracle Database Storage Architecture

The files that constitute an Oracle database are organized into the following:

- **Control files:** Contain data about the database itself (that is, physical database structure information). These files are critical to the database. Without them, you cannot open data files to access the data within the database.
- **Data files:** Contain the user or application data of the database, as well as metadata and the data dictionary
- **Online redo log files:** Allow for instance recovery of the database. If the database server crashes and does not lose any data files, the instance can recover the database with the information in these files.

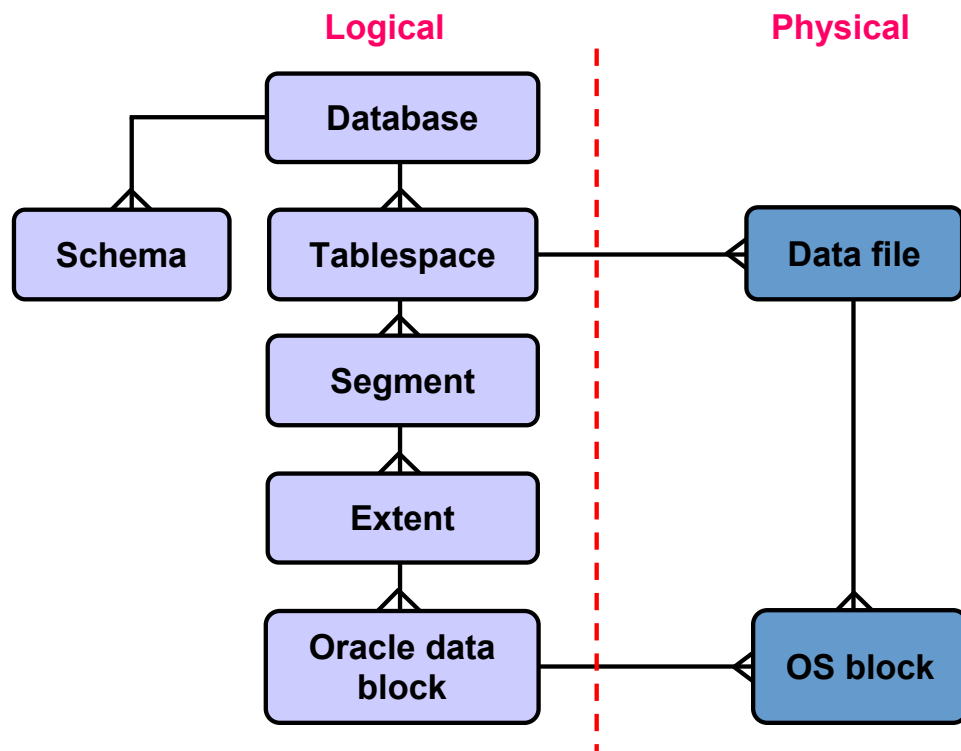
The following additional files are important to the successful running of the database:

- **Backup files:** Are used for database recovery. You typically restore a backup file when a media failure or user error has damaged or deleted the original file.
- **Archived log files:** Contain an ongoing history of the data changes (redo) that are generated by the instance. Using these files and a backup of the database, you can recover a lost data file. That is, archive logs enable the recovery of restored data files.
- **Parameter file:** Is used to define how the instance is configured when it starts up
- **Password file:** Allows `sysdba/sysoper/sysasm` to connect remotely to the database and perform administrative tasks

Oracle Database Storage Architecture (continued)

- **Network files:** Are used for starting the database listener and store information required for user connections
- **Trace files:** Each server and background process can write to an associated trace file. When an internal error is detected by a process, the process dumps information about the error to its trace file. Some of the information written to a trace file is intended for the database administrator, whereas other information is for Oracle Support Services.
- **Alert log files:** These are special trace entries. The alert log of a database is a chronological log of messages and errors. Each instance has one alert log file. Oracle recommends that you review this alert log periodically.

Logical and Physical Database Structures



ORACLE

Copyright © 2009, Oracle. All rights reserved.

Logical and Physical Database Structures

An Oracle database has logical and physical storage structures.

Tablespaces

A database is divided into logical storage units called tablespaces, which group related logical structures together. For example, tablespaces commonly group all of an application's objects to simplify some administrative operations. You may have a tablespace for application data and an additional one for application indexes.

Databases, Tablespaces, and Data Files

The relationship among databases, tablespaces, and data files is illustrated in the slide. Each database is logically divided into one or more tablespaces. One or more data files are explicitly created for each tablespace to physically store the data of all logical structures in a tablespace. If it is a TEMPORARY tablespace, instead of a data file, the tablespace has a temporary file.

Logical and Physical Database Structures (continued)

Schemas

A schema is a collection of database objects that are owned by a database user. Schema objects are the logical structures that directly refer to the database's data. Schema objects include such structures as tables, views, sequences, stored procedures, synonyms, indexes, clusters, and database links. In general, schema objects include everything that your application creates in the database.

Data Blocks

At the finest level of granularity, an Oracle database's data is stored in data blocks. One data block corresponds to a specific number of bytes of physical database space on the disk. A data block size is specified for each tablespace when it is created. A database uses and allocates free database space in Oracle data blocks.

Extents

The next level of logical database space is called an extent. An extent is a specific number of contiguous data blocks (obtained in a single allocation) that are used to store specific type of information.

Segments

The level of logical database storage above an extent is called a segment. A segment is a set of extents allocated for a certain logical structure. For example, the different types of segments include:

- **Data segments:** Each nonclustered, non-indexed-organized table has a data segment with the exception of external tables, global temporary tables, and partitioned tables, where each table has one or more segments. All of the table's data is stored in the extents of its data segment. For a partitioned table, each partition has a data segment. Each cluster has a data segment. The data of every table in the cluster is stored in the cluster's data segment.
- **Index segments:** Each index has an index segment that stores all of its data. For a partitioned index, each partition has an index segment.
- **Undo segments:** One UNDO tablespace is created per database instance that contains numerous undo segments to temporarily store *undo* information. The information in an undo segment is used to generate read-consistent database information and, during database recovery, to roll back uncommitted transactions for users.
- **Temporary segments:** Temporary segments are created by the Oracle Database when a SQL statement needs a temporary work area to complete execution. When the statement finishes execution, the temporary segment's extents are returned to the instance for future use. Specify a default temporary tablespace for every user or a default temporary tablespace, which is used database-wide.

The Oracle Database dynamically allocates space. When the existing extents of a segment are full, additional extents are added. Because extents are allocated as needed, the extents of a segment may or may not be contiguous on the disk.

Processing a SQL Statement

- Connect to an instance using:
 - The user process
 - The server process
- The Oracle server components that are used depend on the type of SQL statement:
 - Queries return rows.
 - Data manipulation language (DML) statements log changes.
 - Commit ensures transaction recovery.
- Some Oracle server components do not participate in SQL statement processing.

ORACLE

Copyright © 2009, Oracle. All rights reserved.

Processing a SQL Statement

Not all the components of an Oracle instance are used to process SQL statements. The user and server processes are used to connect a user to an Oracle instance. These processes are not part of the Oracle instance, but are required to process a SQL statement.

Some of the background processes, SGA structures, and database files are used to process SQL statements. Depending on the type of SQL statement, different components are used:

- Queries require additional processing to return rows to the user.
- DML statements require additional processing to log the changes made to the data.
- Commit processing ensures that the modified data in a transaction can be recovered.

Some required background processes do not directly participate in processing a SQL statement, but are used to improve performance and to recover the database. For example, the optional Archiver background process, *ARCn*, is used to ensure that a production database can be recovered.

Processing a Query

- Parse:
 - Search for an identical statement.
 - Check the syntax, object names, and privileges.
 - Lock the objects used during parse.
 - Create and store the execution plan.
- Execute: Identify the rows selected.
- Fetch: Return the rows to the user process.

ORACLE

Copyright © 2009, Oracle. All rights reserved.

Processing a Query

Queries are different from other types of SQL statements because, if successful, they return data as results. Other statements simply return success or failure, whereas a query can return one row or thousands of rows.

There are three main stages in the processing of a query:

- Parse
- Execute
- Fetch

During the *parse* stage, the SQL statement is passed from the user process to the server process, and a parsed representation of the SQL statement is loaded into a shared SQL area.

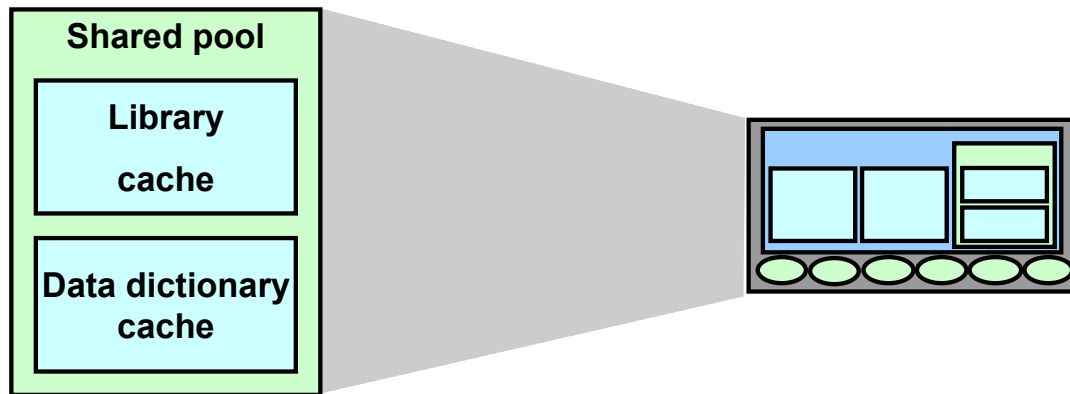
During parse, the server process performs the following functions:

- Searches for an existing copy of the SQL statement in the shared pool
- Validates the SQL statement by checking its syntax
- Performs data dictionary lookups to validate table and column definitions

The execute stage executes the statement using the best optimizer approach and the fetch retrieves the rows back to the user.

Shared Pool

- The library cache contains the SQL statement text, parsed code, and execution plan.
- The data dictionary cache contains table, column, and other object definitions and privileges.
- The shared pool is sized by `SHARED_POOL_SIZE`.



ORACLE

Copyright © 2009, Oracle. All rights reserved.

Shared Pool

During the parse stage, the server process uses the area in the SGA known as the shared pool to compile the SQL statement. The shared pool has two primary components:

- Library cache
- Data dictionary cache

Library Cache

The library cache stores information about the most recently used SQL statements in a memory structure called a shared SQL area. The shared SQL area contains:

- The text of the SQL statement
- The parse tree, which is a compiled version of the statement
- The execution plan, with steps to be taken when executing the statement

The optimizer is the function in the Oracle server that determines the optimal execution plan.

If a SQL statement is reexecuted and a shared SQL area already contains the execution plan for the statement, the server process does not need to parse the statement. The library cache improves the performance of applications that reuse SQL statements by reducing parse time and memory requirements. If the SQL statement is not reused, it is eventually aged out of the library cache.

Shared Pool (continued)

Data Dictionary Cache

The data dictionary cache, also known as the dictionary cache or row cache, is a collection of the most recently used definitions in the database. It includes information about database files, tables, indexes, columns, users, privileges, and other database objects.

During the parse phase, the server process looks for the information in the dictionary cache to resolve the object names specified in the SQL statement and to validate the access privileges. If necessary, the server process initiates the loading of this information from the data files.

Sizing the Shared Pool

The size of the shared pool is specified by the `SHARED_POOL_SIZE` initialization parameter.

Database Buffer Cache

- The database buffer cache stores the most recently used blocks.
- The size of a buffer is based on `DB_BLOCK_SIZE`.
- The number of buffers is defined by `DB_BLOCK_BUFFERS`.



ORACLE

Copyright © 2009, Oracle. All rights reserved.

Database Buffer Cache

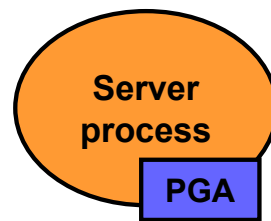
When a query is processed, the server process looks in the database buffer cache for any blocks it needs. If the block is not found in the database buffer cache, the server process reads the block from the data file and places a copy in the buffer cache. Because subsequent requests for the same block may find the block in memory, the requests may not require physical reads. The Oracle server uses a least recently used algorithm to age out buffers that have not been accessed recently to make room for new blocks in the buffer cache.

Sizing the Database Buffer Cache

The size of each buffer in the buffer cache is equal to the size of an Oracle block, and it is specified by the `DB_BLOCK_SIZE` parameter. The number of buffers is equal to the value of the `DB_BLOCK_BUFFERS` parameter.

Program Global Area (PGA)

- Is not shared
- Is writable only by the server process
- Contains:
 - Sort area
 - Session information
 - Cursor state
 - Stack space



ORACLE

Copyright © 2009, Oracle. All rights reserved.

Program Global Area (PGA)

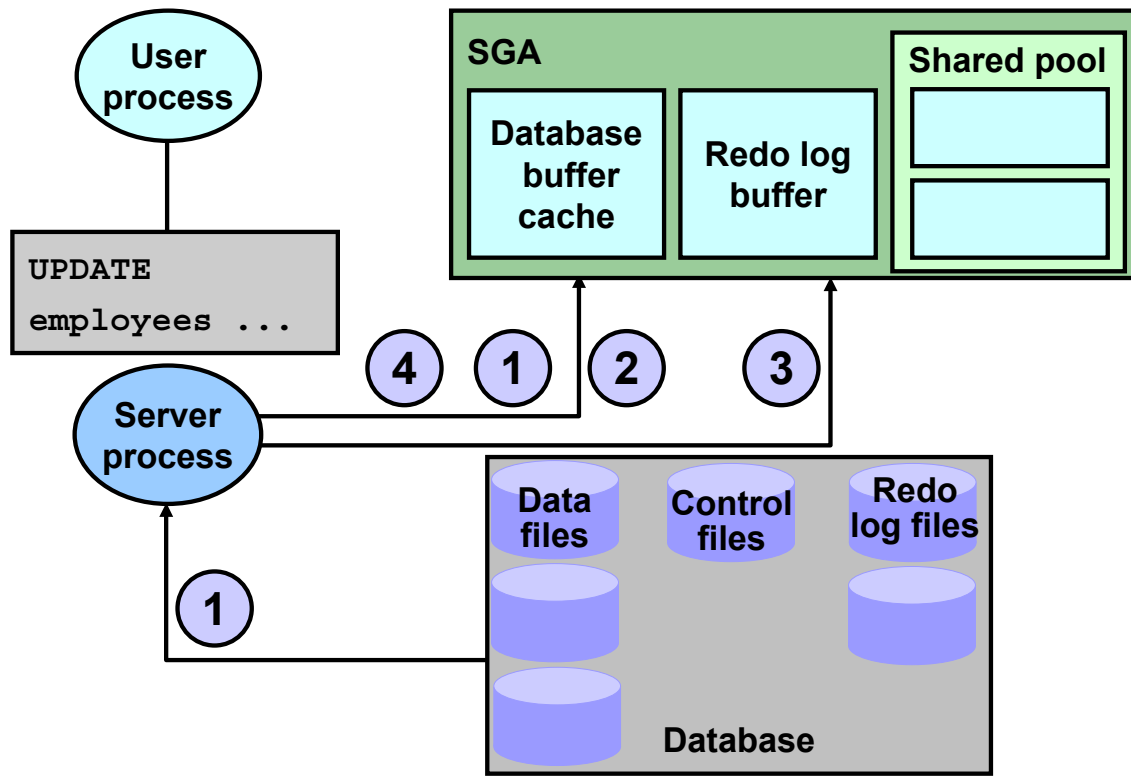
A Program Global Area (PGA) is a memory region that contains data and control information for a server process. It is a nonshared memory created by Oracle when a server process is started. Access to it is exclusive to that server process, and is read and written only by the Oracle server code acting on behalf of it. The PGA memory allocated by each server process attached to an Oracle instance is referred to as the aggregated PGA memory allocated by the instance.

In a dedicated server configuration, the PGA of the server includes the following components:

- **Sort area:** Is used for any sorts that may be required to process the SQL statement
- **Session information:** Includes user privileges and performance statistics for the session
- **Cursor state:** Indicates the stage in the processing of the SQL statements that are currently used by the session
- **Stack space:** Contains other session variables

The PGA is allocated when a process is created, and deallocated when the process is terminated.

Processing a DML Statement



Copyright © 2009, Oracle. All rights reserved.

ORACLE

Oracle University and NETEC, S.A. de C.V. use only.

Processing a DML Statement

A data manipulation language (DML) statement requires only two phases of processing:

- Parse is the same as the parse phase used for processing a query.
- Execute requires additional processing to make data changes.

DML Execute Phase

To execute a DML statement:

- If the data and rollback blocks are not already in the buffer cache, the server process reads them from the data files into the buffer cache
- The server process places locks on the rows that are to be modified
- In the redo log buffer, the server process records the changes to be made to the rollback and data blocks
- The rollback block changes record the values of the data before it is modified. The rollback block is used to store the “before image” of the data, so that the DML statements can be rolled back if necessary.
- The data block changes record the new values of the data

Processing a DML Statement (continued)

The server process records the “before image” to the rollback block and updates the data block. Both of these changes are done in the database buffer cache. Any changed blocks in the buffer cache are marked as dirty buffers (that is, buffers that are not the same as the corresponding blocks on the disk).

The processing of a DELETE or INSERT command uses similar steps. The “before image” for a DELETE contains the column values in the deleted row, and the “before image” of an INSERT contains the row location information.

Because the changes made to the blocks are only recorded in memory structures and are not written immediately to disk, a computer failure that causes the loss of the SGA can also lose these changes.

Redo Log Buffer

- Has its size defined by `LOG_BUFFER`
- Records changes made through the instance
- Is used sequentially
- Is a circular buffer



ORACLE

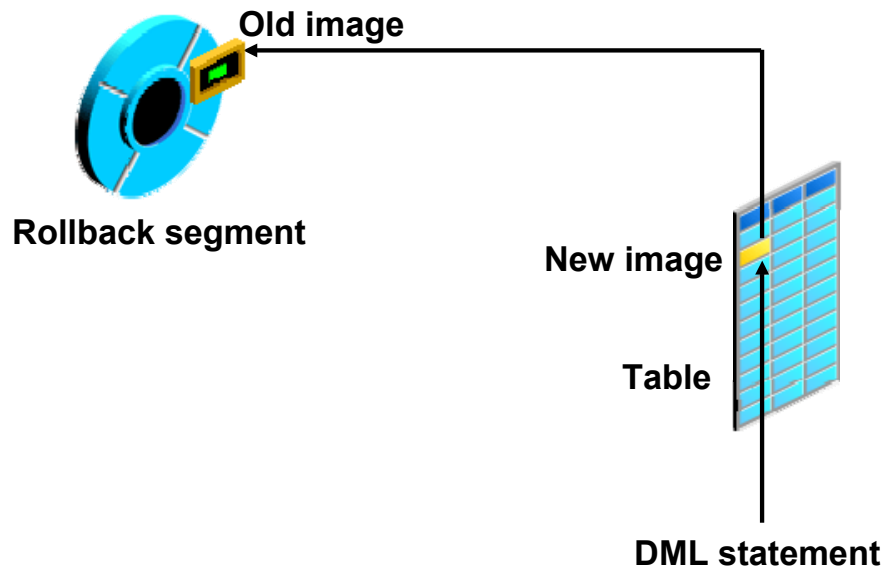
Copyright © 2009, Oracle. All rights reserved.

Redo Log Buffer

The server process records most of the changes made to data file blocks in the redo log buffer, which is a part of the SGA. The redo log buffer has the following characteristics:

- Its size in bytes is defined by the `LOG_BUFFER` parameter.
- It records the block that is changed, the location of the change, and the new value in a redo entry. A redo entry makes no distinction between the types of block that is changed; it only records which bytes are changed in the block.
- The redo log buffer is used sequentially, and changes made by one transaction may be interleaved with changes made by other transactions.
- It is a circular buffer that is reused after it is filled, but only after all the old redo entries are recorded in the redo log files.

Rollback Segment



ORACLE

Copyright © 2009, Oracle. All rights reserved.

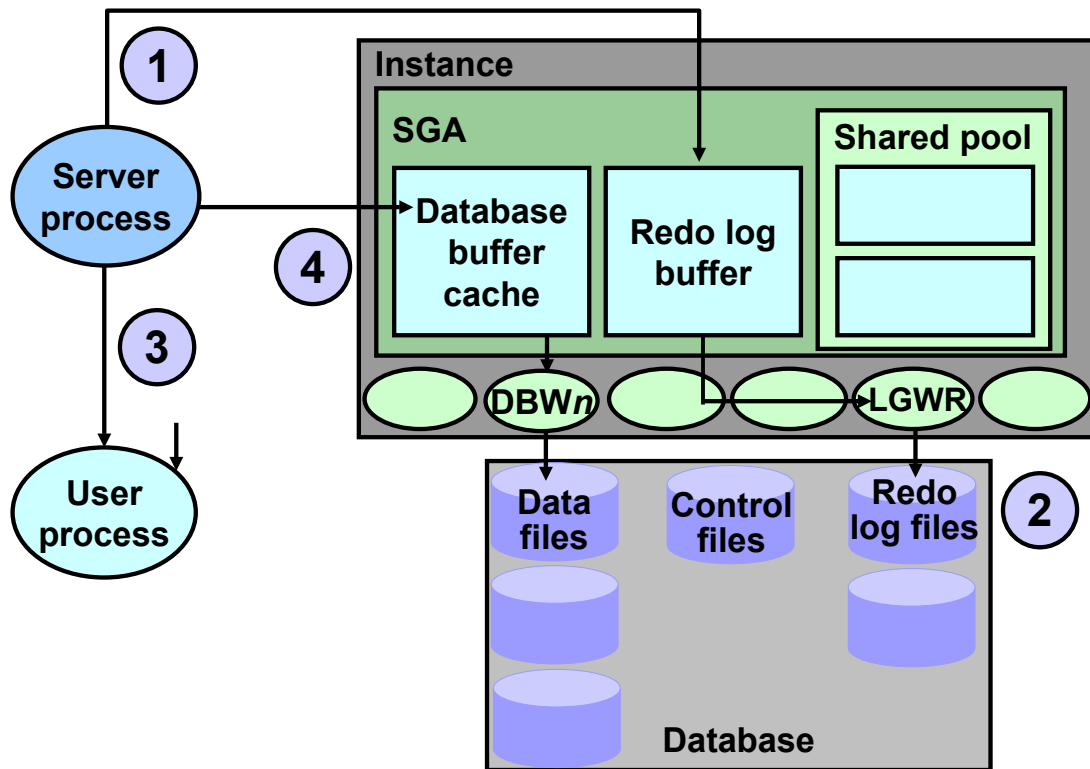
Rollback Segment

Before making a change, the server process saves the old data value in a rollback segment. This “before image” is used to:

- Undo the changes if the transaction is rolled back
- Provide read consistency by ensuring that other transactions do not see uncommitted changes made by the DML statement
- Recover the database to a consistent state in case of failures

Rollback segments, such as tables and indexes, exist in data files, and rollback blocks are brought into the database buffer cache as required. Rollback segments are created by the DBA. Changes to rollback segments are recorded in the redo log buffer.

COMMIT Processing



ORACLE

Copyright © 2009, Oracle. All rights reserved.

COMMIT Processing

The Oracle server uses a fast COMMIT mechanism that guarantees that the committed changes can be recovered in case of instance failure.

System Change Number

Whenever a transaction commits, the Oracle server assigns a commit SCN to the transaction. The SCN is monotonically incremented and is unique within the database. It is used by the Oracle server as an internal time stamp to synchronize data and to provide read consistency when data is retrieved from the data files. Using the SCN enables the Oracle server to perform consistency checks without depending on the date and time of the operating system.

Steps in Processing COMMITs

When a COMMIT is issued, the following steps are performed:

1. The server process places a commit record, along with the SCN, in the redo log buffer.
2. LGWR performs a contiguous write of all the redo log buffer entries up to and including the commit record to the redo log files. After this point, the Oracle server can guarantee that the changes will not be lost even if there is an instance failure.

COMMIT Processing (continued)

3. The user is informed that the COMMIT is complete.
4. The server process records information to indicate that the transaction is complete and that resource locks can be released.

Flushing of the dirty buffers to the data file is performed independently by DBW0 and can occur either before or after the commit.

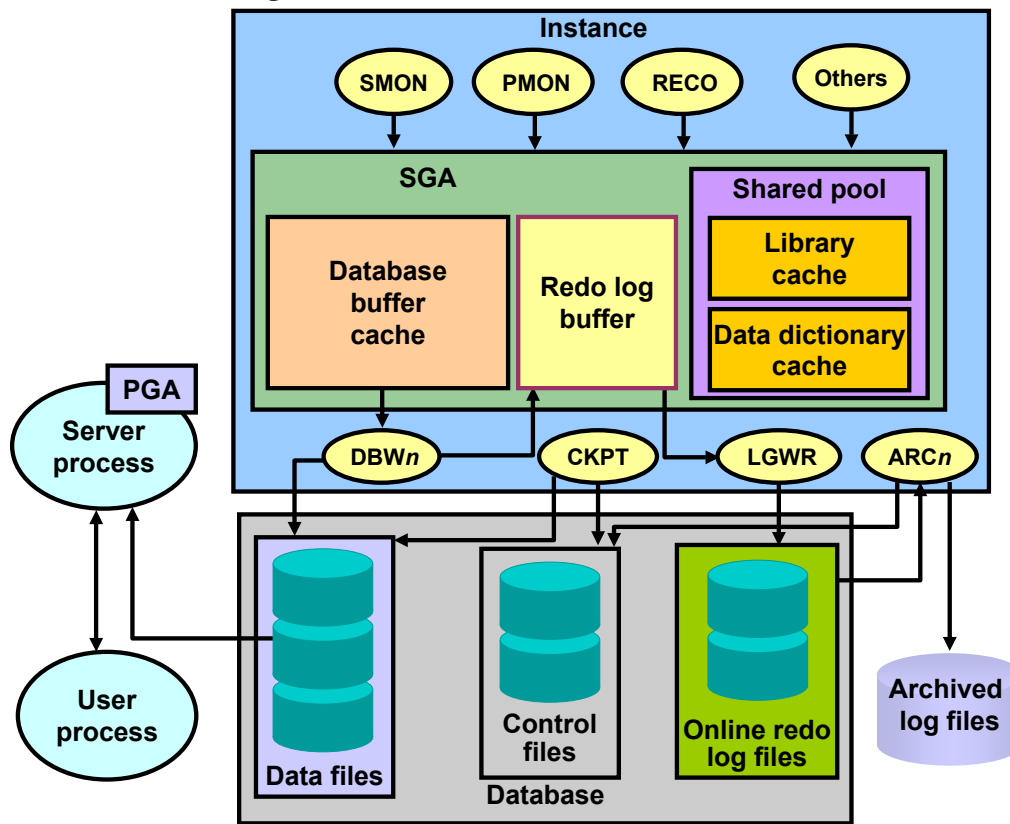
Advantages of the Fast COMMIT

The fast COMMIT mechanism ensures data recovery by writing changes to the redo log buffer instead of the data files. It has the following advantages:

- Sequential writes to the log files are faster than writing to different blocks in the data file.
- Only the minimal information that is necessary to record changes is written to the log files; writing to the data files would require whole blocks of data to be written.
- If multiple transactions request to commit at the same time, the instance piggybacks redo log records into a single write.
- Unless the redo log buffer is particularly full, only one synchronous write is required per transaction. If piggybacking occurs, there can be less than one synchronous write per transaction.
- Because the redo log buffer may be flushed before the COMMIT, the size of the transaction does not affect the amount of time needed for an actual COMMIT operation.

Note: Rolling back a transaction does not trigger LGWR to write to disk. The Oracle server always rolls back uncommitted changes when recovering from failures. If there is a failure after a rollback, before the rollback entries are recorded on disk, the absence of a commit record is sufficient to ensure that the changes made by the transaction are rolled back.

Summary of the Oracle Database Architecture



ORACLE

Copyright © 2009, Oracle. All rights reserved.

Summary of the Oracle Database Architecture

An Oracle database comprises an instance and its associated database:

- An instance comprises the SGA and the background processes
 - **SGA:** Database buffer cache, redo log buffer, shared pool, and so on
 - **Background processes:** SMON, PMON, DBWn, CKPT, LGWR, and so on
- A database comprises storage structures:
 - **Logical:** Tablespaces, schemas, segments, extents, and Oracle block
 - **Physical:** Data files, control files, redo log files

When a user accesses the Oracle database through an application, a server process communicates with the instance on behalf of the user process.

Additional Practices and Solutions

Table of Contents

Additional Practices	3
Additional Practices	4
Additional Practices: Case Study	10
Additional Practices Solution	13
Additional Practices Solution	14
Additional Practices: Case Study Solutions.....	20

Additional Practices

The following exercises can be used for extra practice after you have discussed data manipulation language (DML) and data definition language (DDL) statements in the lessons titled “Managing Schema Objects” and “Manipulating Large Data Sets.”

Note: Run the `lab_ap_cre_special_sal.sql`, `lab_ap_cre_sal_history.sql`, and `lab_ap_cre_mgr_history.sql` scripts in the labs folder to create the `SPECIAL_SAL`, `SAL_HISTORY`, and `MGR_HISTORY` tables.

Additional Practices

1. The Human Resources department wants to get a list of underpaid employees, salary history of employees, and salary history of managers based on an industry salary survey. So they have asked you to do the following:

Write a statement to do the following:

- Retrieve details such as the employee ID, hire date, salary, and manager ID of those employees whose employee ID is more than or equal to 200 from the EMPLOYEES table.
 - If the salary is less than \$5,000, insert details such as the employee ID and salary into the SPECIAL_SAL table.
 - Insert details such as the employee ID, hire date, and salary into the SAL_HISTORY table.
 - Insert details such as the employee ID, manager ID, and salary into the MGR_HISTORY table.
2. Query the SPECIAL_SAL, SAL_HISTORY, and MGR_HISTORY tables to view the inserted records.

SPECIAL_SAL

	EMPLOYEE_ID	SALARY
1	200	4400

SAL_HISTORY

	EMPLOYEE_ID	HIRE_DATE	SALARY
1	201	17-FEB-1996	13000
2	202	17-AUG-1997	6000
3	203	07-JUN-1994	6500
4	204	07-JUN-1994	10000
5	205	07-JUN-1994	12000
6	206	07-JUN-1994	8300

MGR_HISTORY

Additional Practices (continued)

EMPLOYEE_ID	MANAGER_ID	SALARY
201	100	13000
202	201	6000
203	101	6500
204	101	10000
205	101	12000
206	205	8300

3. Nita, the DBA, needs you to create a table, which has a primary key constraint, but she wants the index to have a different name than the constraint. Create the LOCATIONS_NAMED_INDEX table based on the following table instance chart. Name the index for the PRIMARY KEY column as LOCATIONS_PK_IDX.

Column Name	Deptno	Dname
Primary Key	Yes	
Data Type	Number	VARCHAR2
Length	4	30

4. Query the USER_INDEXES table to display the INDEX_NAME for the LOCATIONS_NAMED_INDEX table.

INDEX_NAME	TABLE_NAME
1 LOCATIONS_PK_IDX	LOCATIONS_NAMED_INDEX


Additional Practices (continued)

The following exercises can be used for extra practice after you have discussed datetime functions.

You work for a global company and the new vice president of operations wants to know the different time zones of all the company branches. The new vice president has requested the following information:

5. Alter the session to set the NLS_DATE_FORMAT to DD-MON-YYYY HH24 : MI : SS.
6.
 - a. Write queries to display the time zone offsets (TZ_OFFSET) for the following time zones:





– Australia/Sydney

 TZ_OFFSET('AUSTRALIA/SYDNEY')
1 +10:00

– Chile/Easter Island

 TZ_OFFSET('CHILE/EASTERISLAND')
1 -06:00

- b. Alter the session to set the TIME_ZONE parameter value to the time zone offset of Australia/Sydney.
 - c. Display SYSDATE, CURRENT_DATE, CURRENT_TIMESTAMP, and LOCALTIMESTAMP for this session.
- Note:** The output might be different based on the date when the command is executed.

 SYSDATE	 CURRENT_DATE	 CURRENT_TIMESTAMP	 LOCALTIMESTAMP
1 02-JUL-2009 17:11:46	02-JUL-2009 20:11:46	02-JUL-09 08.11.46.000000000 PM +10:00	02-JUL-09 08.11.46.000000000 PM

- d. Alter the session to set the TIME_ZONE parameter value to the time zone offset of Chile/Easter Island.
- Note:** The results of the preceding question are based on a different date, and in some cases, they will not match the actual results that the students get. Also, the time zone offset of the various countries may differ, based on daylight saving time.
- e. Display SYSDATE, CURRENT_DATE, CURRENT_TIMESTAMP, and LOCALTIMESTAMP for this session.
- Note:** The output may be different based on the date when the command is executed.

Additional Practices (continued)

	SYSDATE	CURRENT_DATE	CURRENT_TIMESTAMP	LOCALTIMESTAMP
1	02-JUL-2009 17:12:33	02-JUL-2009 04:12:33	02-JUL-09 04.12.33.000000000 AM -06:00	02-JUL-09 04.12.33.000000000 AM

f. Alter the session to set NLS_DATE_FORMAT to DD-MON-YYYY.

Note

- Observe in the preceding question that CURRENT_DATE, CURRENT_TIMESTAMP, and LOCALTIMESTAMP are all sensitive to the session time zone. Observe that SYSDATE is not sensitive to the session time zone.
- The results of the preceding question are based on a different date, and in some cases, they will not match the actual results that the students get. Also the time zone offset of the various countries may differ, based on daylight saving time.

7. The Human Resources department wants a list of employees who are up for review in January, so they have requested you to do the following:

Write a query to display the last names, month of the date of hire, and hire date of those employees who have been hired in the month of January, irrespective of the year of hire.

	LAST_NAME	EXTRACT(MONTHFROMHIRE_DATE)	HIRE_DATE
1	Grant		1 13-JAN-2000
2	De Haan		1 13-JAN-1993
3	Hunold		1 03-JAN-1990
4	Landry		1 14-JAN-1999
5	Davies		1 29-JAN-1997
6	Partners		1 05-JAN-1997
7	Zlotkey		1 29-JAN-2000
8	Tucker		1 30-JAN-1997
9	King		1 30-JAN-1996
10	Marvins		1 24-JAN-2000
11	Fox		1 24-JAN-1998
12	Johnson		1 04-JAN-2000
13	Taylor		1 24-JAN-1998
14	Sarchand		1 27-JAN-1996

Additional Practices (continued)

The following exercises can be used for extra practice after you have discussed advanced subqueries.

- The CEO needs a report on the top three earners in the company for profit sharing. You are responsible to provide the CEO with a list. Write a query to display the top three earners in the EMPLOYEES table. Display their last names and salaries.

	LAST_NAME	SALARY
1	King	24000
2	Kochhar	17000
3	De Haan	17000

- The benefits for the state of California have been changed based on a local ordinance. So the benefits representative has asked you to compile a list of the people who are affected.
Write a query to display the employee ID and last names of the employees who work in the state of California.

Hint: Use scalar subqueries.

	EMPLOYEE_ID	LAST_NAME
1	198	OConnell
2	199	Grant
3	120	Weiss
4	121	Fripp
5	122	Kaufling
6	123	Vollman
7	124	Mourgos
8	125	Nayer
9	126	Mikkilineni
10	127	Landry
11	128	Markle

- Nita, the DBA, wants to remove old information from the database. One of the things she thinks is unnecessary is the old employment records. She has asked you to do the following:

Write a query to delete the oldest JOB_HISTORY row of an employee by looking up the JOB_HISTORY table for the MIN(START_DATE) for the employee. Delete the records of *only* those employees who have changed at least two jobs.

Hint: Use a correlated DELETE command.

Additional Practices (continued)

11. The vice president of Human Resources needs the complete employment records for the annual employee recognition banquet speech. The vice president makes a quick phone call to stop you from following the DBA's orders.

Roll back the transaction.

12. The sluggish economy is forcing management to take cost reduction actions. The CEO wants to review the highest paid jobs in the company. You are responsible to provide the CEO with a list based on the following specifications:

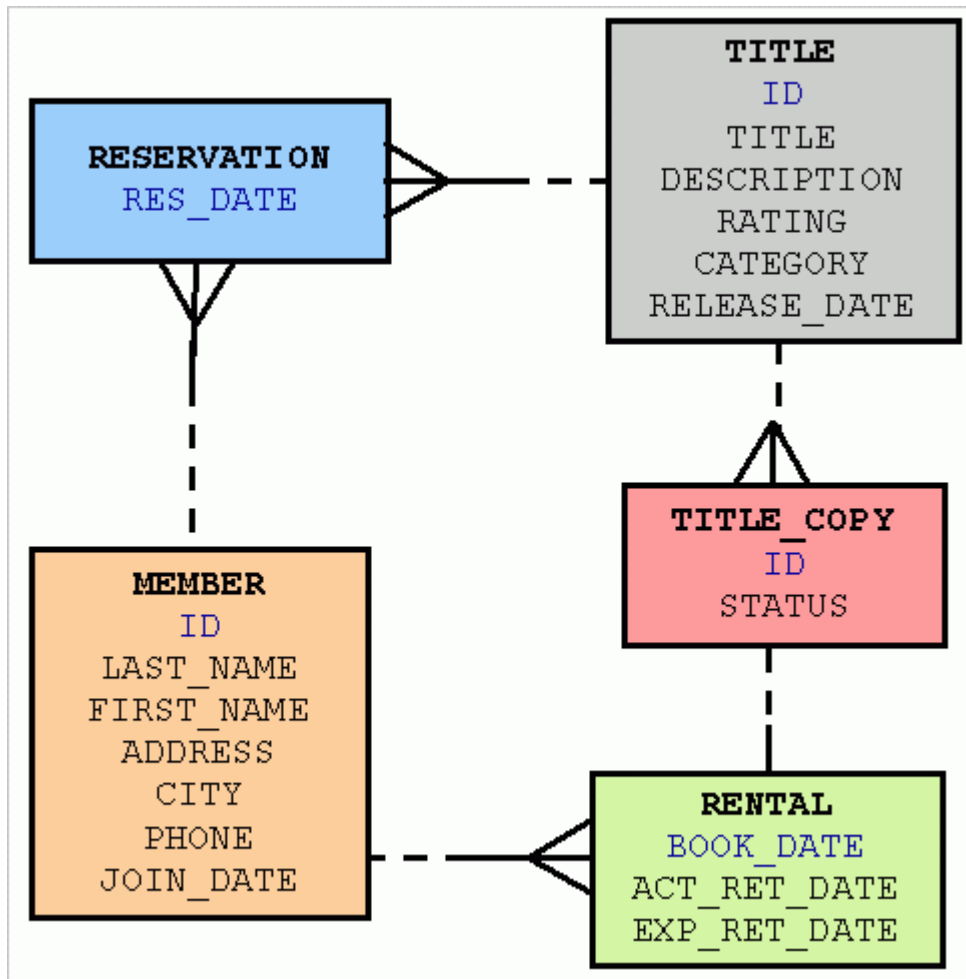
Write a query to display the job IDs of those jobs whose maximum salary is above half the maximum salary in the entire company. Use the WITH clause to write this query. Name the query MAX_SAL_CALC.

	JOB_TITLE	JOB_TOTAL
1	President	24000
2	Administration Vice President	17000
3	Sales Manager	14000
4	Marketing Manager	13000

Additional Practices: Case Study

In the case study for the *SQL Fundamentals I* course, you built a set of database tables for a video application. In addition, you inserted, updated, and deleted records in a video store database and generated a report.

The following is a diagram of the tables and columns that you created for the video application:



Note: First, run the `dropvid.sql` script in the labs folder to drop tables if they already exist. Then run the `buildvid.sql` script in the labs folder to create and populate the tables.

1. Verify that the tables were created properly by running a report to show the list of tables and their column definitions.

Additional Practices: Case Study (continued)

	TABLE_NAME	COLUMN_NAME	DATA_TYPE	NULLABLE
1	MEMBER	MEMBER_ID	NUMBER	N
2	MEMBER	LAST_NAME	VARCHAR2	N
3	MEMBER	FIRST_NAME	VARCHAR2	Y
4	MEMBER	ADDRESS	VARCHAR2	Y
5	MEMBER	CITY	VARCHAR2	Y
6	MEMBER	PHONE	VARCHAR2	Y
7	MEMBER	JOIN_DATE	DATE	N
8	RENTAL	BOOK_DATE	DATE	N
9	RENTAL	COPY_ID	NUMBER	N
10	RENTAL	MEMBER_ID	NUMBER	N
11	RENTAL	TITLE_ID	NUMBER	N
12	RENTAL	ACT_RET_DATE	DATE	Y
13	RENTAL	EXP_RET_DATE	DATE	Y
14	RESERVATION	RES_DATE	DATE	N
15	RESERVATION	MEMBER_ID	NUMBER	N
16	RESERVATION	TITLE_ID	NUMBER	N

- Verify the existence of the MEMBER_ID_SEQ and TITLE_ID_SEQ sequences in the data dictionary.

	SEQUENCE_NAME
1	DEPARTMENTS_SEQ
2	EMPLOYEES_SEQ
3	LOCATIONS_SEQ
4	MEMBER_ID_SEQ
5	TITLE_ID_SEQ

- You want to create some users who have access only to their own rentals. Create a user called Carmen and grant her the privilege to select from the RENTAL table.
Note: Make sure to prefix the username with your database account. For example, if you are the user oraxx, create a user called oraxx_Carmen.
- Add a price column (number 4,2) to the TITLE table to store how much it costs to rent the title.
- Add a CATEGORY table to store CATEGORY_ID and CATEGORY_DESCRIPTION. The table has a foreign key with the CATEGORY column in the TITLE table.
- Select all the tables from the data dictionary.
- There is no real need to store reservations any longer. You can drop the table.

Additional Practices: Case Study (continued)

8. Create a RENTAL_HISTORY table to store the details of a rental by member for the last six months. (**Hint:** You can copy the RENTAL table.)
9. Show a list of the top 10 titles rented in the last month grouped by category.

RANK	CATEGORY	TITLE
1	ACTION	Soda Gang
2	CHILD	Willie and Christmas Too
3	COMEDY	My Day Off
4	SCIFI	Alien Again
5	SCIFI	Interstellar Wars

10. You want to calculate the late fee (price of title/day) if the member brings back the video six days late.

RANK	TITLE	MEMBER_ID	PRICE	LATEFEE
1	Alien Again	101	(null)	(null)
2	My Day Off	102	(null)	(null)
3	Interstellar Wars	101	(null)	(null)

11. Show a list of members who have rented two or more times.

RANK	MEMBER_ID	LAST_NAME	FIRST_NAME
1	101	Velasquez	Carmen

12. Show a list of titles who have a status of rented.

RANK	TITLE
1	Alien Again
2	My Day Off
3	Interstellar Wars

13. Show a list of members who have “99” in their phone numbers.

RANK	POSITION	MEMBER_ID	LAST_NAME	FIRST_NAME
1	1	101	Velasquez	Carmen
2	1	106	Urguhart	Molly
3	1	109	Catchpole	Antoinette

Additional Practices Solution

The following exercises can be used for extra practice after you have discussed data manipulation language (DML) and data definition language (DDL) statements in the lessons titled “Managing Schema Objects” and “Manipulating Large Data Sets.”

Note: Run the `lab_ap_cre_special_sal.sql`, `lab_ap_cre_sal_history.sql`, and `lab_ap_cre_mgr_history.sql` scripts in the labs folder to create the `SPECIAL_SAL`, `SAL_HISTORY`, and `MGR_HISTORY` tables

Additional Practices Solution

1. The Human Resources department wants to get a list of underpaid employees, salary history of employees, and salary history of managers based on an industry salary survey. So they have asked you to do the following:

Write a statement to do the following:

- Retrieve details such as the employee ID, hire date, salary, and manager ID of those employees whose employee ID is more than or equal to 200 from the EMPLOYEES table.
- If the salary is less than \$5,000, insert details such as the employee ID and salary into the SPECIAL_SAL table.
- Insert details such as the employee ID, hire date, and salary into the SAL_HISTORY table.
- Insert details such as the employee ID, manager ID, and salary into the MGR_HISTORY table.

```
INSERT ALL
WHEN SAL < 5000 THEN
INTO special_sal VALUES (EMPID, SAL)
ELSE
INTO sal_history VALUES (EMPID, HIREDATE, SAL)
INTO mgr_history VALUES (EMPID, MGR, SAL)
SELECT employee_id EMPID, hire_date HIREDATE,
       salary SAL, manager_id MGR
FROM employees
WHERE employee_id >=200;
```

2. Query the SPECIAL_SAL, SAL_HISTORY, and the MGR_HISTORY tables to view the inserted records.

```
SELECT * FROM special_sal;
SELECT * FROM sal_history;
SELECT * FROM mgr_history;
```

3. Nita, the DBA, needs you to create a table, which has a primary key constraint, but she wants the index to have a different name than the constraint. Create the LOCATIONS_NAMED_INDEX table based on the following table instance chart. Name the index for the PRIMARY KEY column as LOCATIONS_PK_IDX.

Column Name	Deptno	Dname
Primary Key	Yes	
Data Type	Number	VARCHAR2
Length	4	30

Additional Practices Solution (continued)

```
CREATE TABLE LOCATIONS_NAMED_INDEX  
(location_id NUMBER(4) PRIMARY KEY USING INDEX  
(CREATE INDEX locations_pk_idx ON  
LOCATIONS_NAMED_INDEX(location_id)),  
location_name VARCHAR2(20));
```

4. Query the USER_INDEXES table to display the INDEX_NAME for the LOCATIONS_NAMED_INDEX table.

```
SELECT INDEX_NAME, TABLE_NAME  
FROM USER_INDEXES  
WHERE TABLE_NAME = 'LOCATIONS_NAMED_INDEX';
```

Additional Practices Solution (continued)

The following exercises can be used for extra practice after you have discussed datetime functions.

You work for a global company and the new vice president of operations wants to know the different time zones of all the company branches. The new vice president has requested the following information:

5. Alter the session to set NLS_DATE_FORMAT to DD-MON-YYYY HH24:MI:SS.

```
ALTER SESSION
SET NLS_DATE_FORMAT = 'DD-MON-YYYY HH24:MI:SS';
```

- 6.
- a. Write queries to display the time zone offsets (TZ_OFFSET) for the following time zones:
- Australia/Sydney

```
SELECT TZ_OFFSET ('Australia/Sydney') from dual;
```

- Chile/Easter Island

```
SELECT TZ_OFFSET ('Chile/EasterIsland') from dual;
```

- b. Alter the session to set the TIME_ZONE parameter value to the time zone offset of Australia/Sydney.

```
ALTER SESSION SET TIME_ZONE = '+10:00';
```

- c. Display SYSDATE, CURRENT_DATE, CURRENT_TIMESTAMP, and LOCALTIMESTAMP for this session.
- Note:** The output may be different based on the date when the command is executed.

```
SELECT SYSDATE, CURRENT_DATE, CURRENT_TIMESTAMP,
LOCALTIMESTAMP FROM DUAL;
```

- d. Alter the session to set the TIME_ZONE parameter value to the time zone offset of Chile/Easter Island.

Note: The results of the preceding question are based on a different date, and in some cases, they will not match the actual results that the students get. Also, the time zone offset of the various countries may differ, based on daylight saving time.

```
ALTER SESSION SET TIME_ZONE = '-06:00';
```

- e. Display SYSDATE, CURRENT_DATE, CURRENT_TIMESTAMP, and LOCALTIMESTAMP for this session.

Additional Practices Solution (continued)

Note: The output may be different based on the date when the command is executed.

```
SELECT SYSDATE, CURRENT_DATE, CURRENT_TIMESTAMP,
LOCALTIMESTAMP FROM DUAL;
```

- f. Alter the session to set NLS_DATE_FORMAT to DD-MON-YYYY.

```
ALTER SESSION SET NLS_DATE_FORMAT = 'DD-MON-YYYY';
```

Note

- Observe in the preceding question that CURRENT_DATE, CURRENT_TIMESTAMP, and LOCALTIMESTAMP are all sensitive to the session time zone. Observe that SYSDATE is not sensitive to the session time zone.
- The results of the preceding question are based on a different date, and in some cases, they will not match the actual results that the students get. Also, the time zone offset of the various countries may differ, based on daylight saving time.

7. The Human Resources department wants a list of employees who are up for review in January, so they have requested you to do the following:

Write a query to display the last names, month of the date of hire, and hire date of those employees who have been hired in the month of January, irrespective of the year of hire.

```
SELECT last_name, EXTRACT (MONTH FROM HIRE_DATE), HIRE_DATE
FROM employees
WHERE EXTRACT (MONTH FROM HIRE_DATE) = 1;
```

The following exercises can be used for extra practice after you have discussed advanced subqueries.

Note: If you have converted the HIRE_DATE column to TIMESTAMP using code_05_12_sb.sql, the display of the HIRE_DATE column would be like 13-JAN-00 12.00.00.000000000 AM

8. The CEO needs a report on the top three earners in the company for profit sharing. You are responsible to provide the CEO with a list.

Write a query to display the top three earners in the EMPLOYEES table. Display their last names and salaries.

```
SELECT last_name, salary
FROM employees e
WHERE 3 > (SELECT COUNT (*)
          FROM employees
          WHERE e.salary < salary);
```

Additional Practices Solution (continued)

9. The benefits for the state of California have been changed based on a local ordinance. So the benefits representative has asked you to compile a list of the people who are affected. Write a query to display the employee ID and last names of the employees who work in the state of California.

Hint: Use scalar subqueries.

```
SELECT employee_id, last_name
FROM employees e
WHERE ((SELECT location_id
        FROM departments d
        WHERE e.department_id = department_id )
       IN (SELECT location_id
           FROM locations l
           WHERE state_province = 'California'));
```

10. Nita, the DBA, wants to remove old information from the database. One of the things she thinks is unnecessary is the old employment records. She has asked you to do the following:

Write a query to delete the oldest JOB_HISTORY row of an employee by looking up the JOB_HISTORY table for the MIN (START_DATE) for the employee.

Delete the records of *only* those employees who have changed at least two jobs.

Hint: Use a correlated DELETE command.

Additional Practices Solution (continued)

```
DELETE FROM job_history JH
WHERE employee_id =
      (SELECT employee_id
       FROM employees E
       WHERE JH.employee_id = E.employee_id
       AND START_DATE = (SELECT MIN(start_date)
                        FROM job_history JH
                        WHERE JH.employee_id =
                          E.employee_id)
       AND 3 > (SELECT COUNT(*)
              FROM job_history JH
              WHERE JH.employee_id =
                E.employee_id
              GROUP BY EMPLOYEE_ID
              HAVING COUNT(*) >= 2));
```

11. The vice president of Human Resources needs the complete employment records for the annual employee recognition banquet speech. The vice president makes a quick phone call to stop you from following the DBA's orders.
Roll back the transaction.

```
ROLLBACK;
```

12. The sluggish economy is forcing management to take cost reduction actions. The CEO wants to review the highest paid jobs in the company. You are responsible to provide the CEO with a list based on the following specifications:
Write a query to display the job IDs of those jobs whose maximum salary is above half the maximum salary in the entire company. Use the WITH clause to write this query. Name the query MAX_SAL_CALC.

```
WITH
MAX_SAL_CALC AS (SELECT job_title, MAX(salary) AS
job_total
FROM employees, jobs
WHERE employees.job_id = jobs.job_id
GROUP BY job_title)
SELECT job_title, job_total
FROM MAX_SAL_CALC
WHERE job_total > (SELECT MAX(job_total) * 1/2
FROM MAX_SAL_CALC)
ORDER BY job_total DESC;
```

Additional Practices: Case Study Solutions

First, run the `dropvid.sql` script in the labs folder to drop tables if they already exist. Then run the `buildvid.sql` script in the labs folder to create and populate the tables.

1. Verify that the tables were created properly by running a report to show the list of tables and their column definitions.

```
SELECT table_name,column_name,data_type,nullable
FROM user_tab_columns
WHERE table_name
IN('MEMBER','TITLE','TITLE_COPY','RENTAL','RESERVATION');
```

2. Verify the existence of the `MEMBER_ID_SEQ` and `TITLE_ID_SEQ` sequences in the data dictionary.

```
SELECT sequence_name FROM user_sequences;
```

3. You want to create some users who have access only to their own rentals. Create a user called Carmen and grant her the privilege to select from the `RENTAL` table.

Note: Make sure to prefix the username with your database account. For example, if you are the user `oraxx`, create a user called `oraxx_carmen`.

```
CREATE USER oraxx_carmen IDENTIFIED BY oracle ;
GRANT select ON rental TO oraxx_carmen;
```

4. Add a price column (number 4,2) to the `TITLE` table to store how much it costs to rent the title.

```
ALTER TABLE title ADD(price NUMBER(6))
```

5. Add a `CATEGORY` table to store `CATEGORY_ID` and `CATEGORY_DESCRIPTION`. The table has a foreign key with the `CATEGORY` column in the `TITLE` table.

```
CREATE TABLE CATEGORY
(
    "CATEGORY_ID" NUMBER(6,0) NOT NULL ENABLE,
    "CATEGORY_DESCRIPTION" VARCHAR2(4000 BYTE),
    CONSTRAINT "CATEGORY_PK" PRIMARY KEY ("CATEGORY_ID"))
```

6. Select all the tables from the data dictionary.

```
SELECT table_name FROM user_tables order by table_name;
```

7. There is no real need to store reservations any longer. You can drop the table.

```
DROP TABLE reservation cascade constraints;
```

Additional Practices: Case Study Solutions (continued)

8. Create a RENTAL_HISTORY table to store the details of a rental by member for the last six months. (Hint: You can copy the RENTAL table.)

```
CREATE TABLE rental_history as select * from rental where '1' = '1'
```

9. Show a list of the top 10 titles rented in the last month grouped by category.

```
SELECT t.CATEGORY, t.TITLE
FROM TITLE t, RENTAL r
WHERE t.TITLE_ID = r.TITLE_ID AND
      r.BOOK_DATE > (SYSDATE - 30) AND
      rownum < 10
order by category;
```

10. You want to calculate the late fee (price of title/day) if the member brings back the video six days late.

```
SELECT t.title, m.member_id, t.price, (t.price*6) latefee
FROM title t, member m, rental r
WHERE t.title_id = r.title_id AND
      m.member_id = r.member_id AND
      r.act_ret_date is null;
```

11. Show a list of members who have rented two or more times.

```
SELECT member_id, last_name, first_name FROM member m
where 2 <= (select count(*) from rental_history where
member_id = m.member_id);
```

12. Show a list of titles who have a status of rented.

```
SELECT t.title
FROM title t
JOIN (select title_id, status from title_copy) b
ON t.title_id = b.title_id AND b.status = 'RENTED';
```

13. Show a list of members who have “99” in their phone numbers.

```
SELECT REGEXP_COUNT(phone,'99',1,'i') position, member_id,
last_name, first_name
FROM member
WHERE REGEXP_COUNT(phone,'99',1,'i') > 0;
```

