

Oracle Database 12c: Analytic SQL for Data Warehousing

Activity Guide

D79991GC10

Edition 1.0

September 2013

D83927

ORACLE®

Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

Disclaimer

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 and Java are registered trademarks of Oracle and/or its affiliates. Other names may be trademarks of their respective owners.

Authors

Supriya Ananth, Lauran Serhal

Technical Contributors and Reviewers

Nancy Greenberg, Swarnapriya Shridhar, Hermann Baer, Angela Amor, John Haydu, Laszlo Czinkoczki, Mirella Tumolo, Anjulaponni Aazhagulekshmi, Sailaja Pasupuleti, Milos Randak
Yanti Chang, Charlie Berger, Joel Goodman

This book was published using: Oracle Tutor

Table of Contents

Practices for Lesson 1: Introduction.....	1-1
Practices for Lesson 1: Overview.....	1-2
Practice 1-1: Introduction.....	1-3
Practices for Lesson 2: Grouping and Aggregating Data Using SQL.....	2-1
Practices for Lesson 2: Overview.....	2-2
Practice 2-1: Grouping and Aggregating Data Using SQL.....	2-3
Case Study.....	2-8
Practices for Lesson 3: Hierarchical Retrieval.....	3-1
Practices for Lesson 3: Overview.....	3-2
Practice 3-1: Hierarchical Retrieval.....	3-3
Case Study.....	3-5
Practices for Lesson 4: Working with Regular Expressions.....	4-1
Practices for Lesson 4: Overview.....	4-2
Practice 4-1: Analyzing Data Using Regular Expressions	4-3
Case Study.....	4-6
Practices for Lesson 5: Analyzing and Reporting Data Using SQL.....	5-1
Practices for Lesson 5: Overview.....	5-2
Practice 5-1: Analyzing and Reporting Data Using SQL.....	5-3
Case Study.....	5-6
Practices for Lesson 6: Performing Pivoting and Unpivoting Operations.....	6-1
Practice 6-1: Performing Pivoting and Unpivoting Operations	6-2
Case Study.....	6-5
Practices for Lesson 7: Pattern Matching using SQL	7-1
Practices for Lesson 7: Overview.....	7-2
Practice 7-1: Pattern Matching using SQL.....	7-3
Case Study.....	7-7
Practices for Lesson 8: Modeling Data Using SQL	8-1
Practices for Lesson 8: Overview.....	8-2
Practice 8-1: Modeling Data by Using SQL	8-3
Case Study.....	8-10

Practices for Lesson 1: Introduction

Chapter 1

Practices for Lesson 1: Overview

Practice Overview

In these practices, you will ...

Practice 1-1: Introduction

Overview

Assumptions

Task

In this practice, you review the available SQL Developer resources. You also learn about the user account that you use in this course. You then start SQL Developer, create a new database connection, and browse your SH and HR tables. You also set some SQL Developer preferences, execute SQL statements, and access and bookmark the Oracle Database documentation and other useful websites that you can use in this course.

Identifying the Available SQL Developer Resources

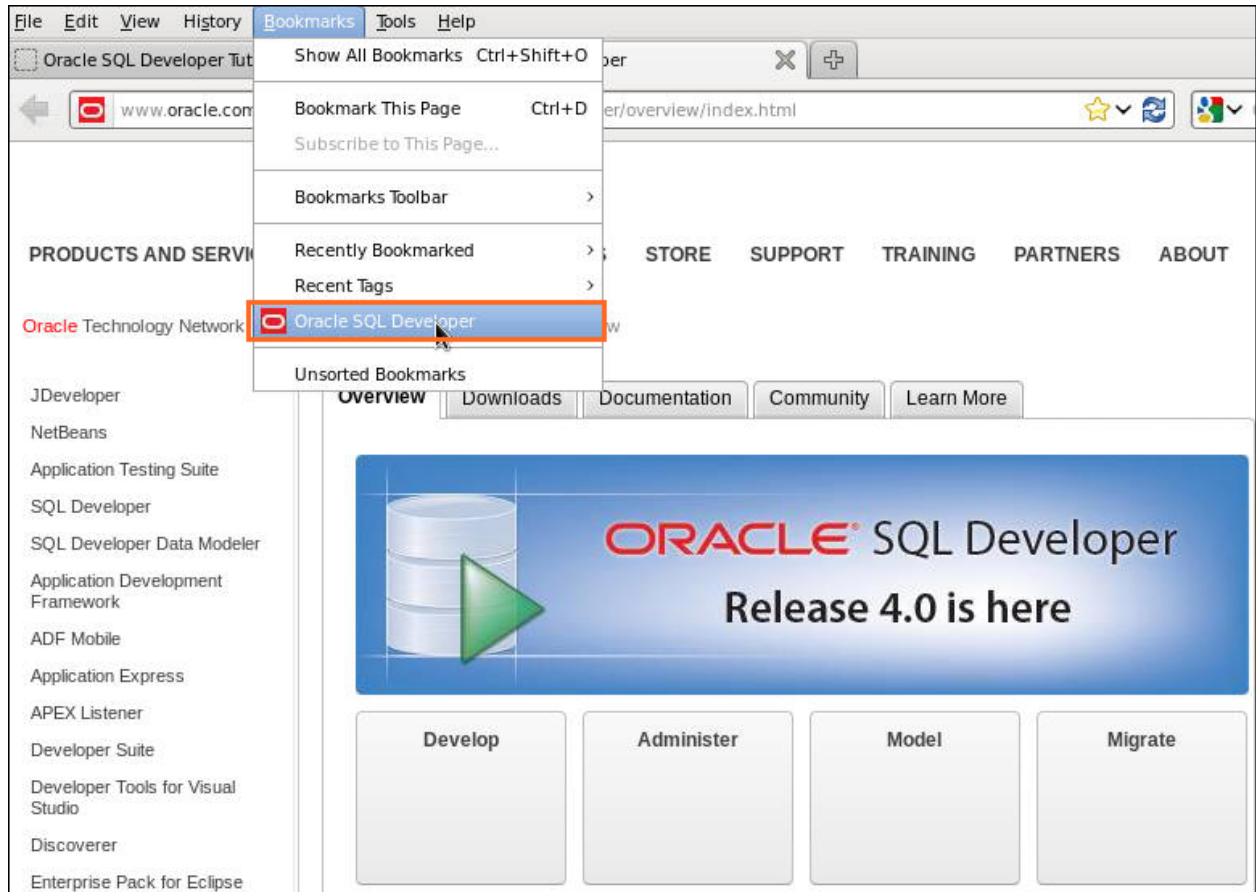
1. Familiarize yourself with Oracle SQL Developer, as needed using “Appendix C: Using SQL Developer.”
2. Access the online SQL Developer Home Page available online at:
www.oracle.com/technetwork/developer-tools/sql-developer/overview/index.html

The SQL Developer Home page is displayed as follows:

The screenshot displays the Oracle SQL Developer Home page. On the left is a sidebar menu listing various Oracle tools: JDeveloper, NetBeans, Application Testing Suite, SQL Developer, SQL Developer Data Modeler, Application Development Framework, ADF Mobile, Application Express, APEX Listener, Developer Suite, Developer Tools for Visual Studio, Discoverer, Enterprise Pack for Eclipse, JHeadstart, Warehouse Builder, Workshop for WebLogic, and XML Developer's Kit. The main content area has a navigation bar with tabs: Overview (selected), Downloads, Documentation, Community, and Learn More. Below the navigation bar is a large blue banner with the text "ORACLE SQL Developer Release 4.0 is here" and a green arrow pointing right. Underneath the banner are four square tiles: "Develop" (showing a database icon with binary code), "Administer" (showing a database icon with a wrench), "Model" (showing a database icon with a flow diagram), and "Migrate" (showing a database icon with a red arrow). Below these tiles is a paragraph of text describing Oracle SQL Developer as a free integrated development environment for Oracle Database, highlighting its capabilities in PL/SQL development, database management, data modeling, and migration. At the bottom of the main area, a red text line states "SQL Developer 4.0 Early Adopter 1 is now available." followed by a "Download" link with a download icon.

3. Bookmark the page for easier future access.

In Firefox, Click the **Bookmarks** tab. A drop-down list appears. Select **Bookmark This Page** option. You see that the page is bookmarked as follows:



4. Access the SQL Developer tutorial available online at:

<http://apex.oracle.com/pls/apex/f?p=44785:2:0::NO:::>

- Select **Database** under Product Family
- Select **SQL Developer** under Product
- Click the **Search** button

SQL Developer tutorials get listed.

Creating and Using a New SQL Developer Database Connection

1. Start up SQL Developer.

Click the SQL Developer icon on your desktop.



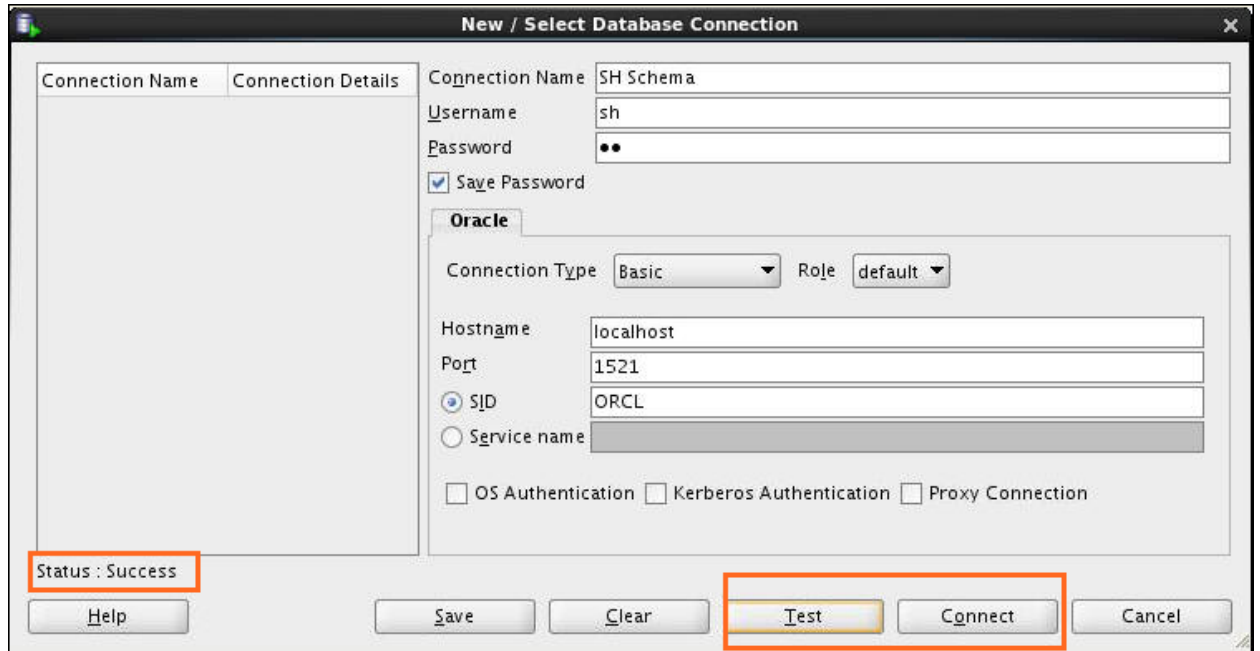
2. Create a database connection using the following information:
 - a. Connection Name: SH Schema
 - b. Username: sh
 - c. Password: sh
 - d. Hostname: Enter the hostname for your PC as provided by the instructor.
 - e. Port: 1521
 - f. SID: ORCL

Right-click the Connections icon on the Connections tabbed page, and then select the New Database Connection option from the shortcut menu. The New/Select Database Connection window is displayed. Use the preceding information provided to create the new database connection.

Note: To display the properties of the newly created connection, right-click the connection name, and then select Properties from the shortcut menu. The following is a sample of the newly created database connection for the SH schema using a local connection:

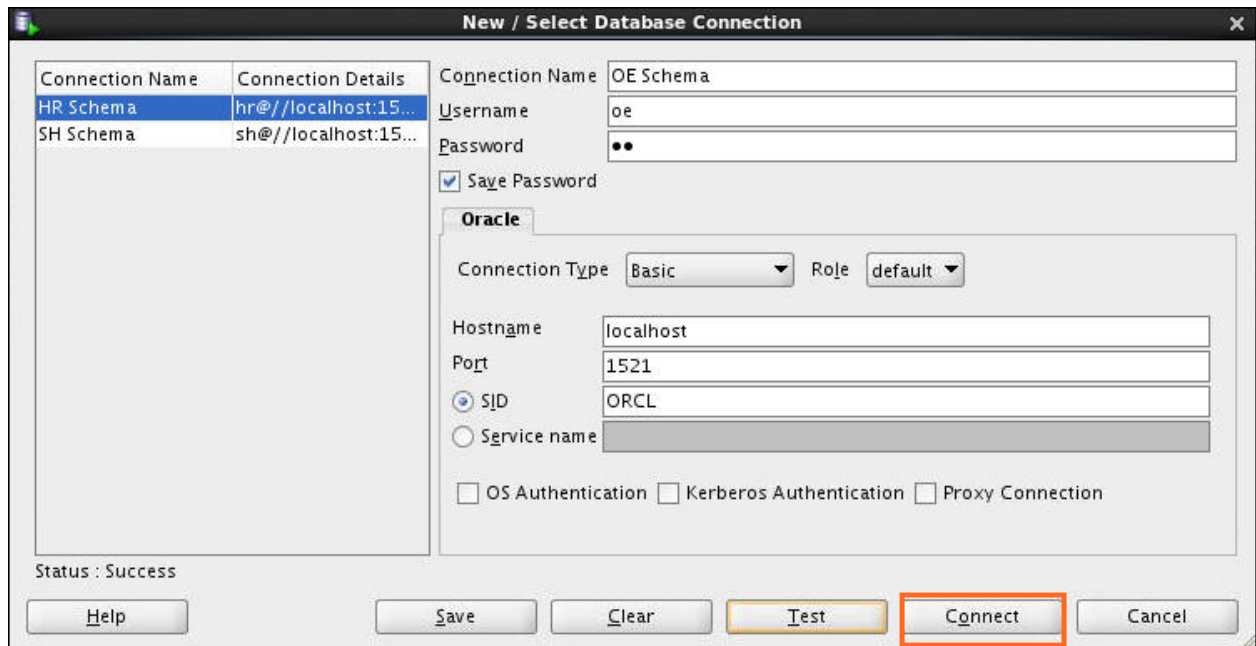
The screenshot shows the 'New / Select Database Connection' dialog box. The 'Connection Name' is 'SH Schema'. The 'Username' is 'sh' and the 'Password' is 'sh'. The 'Save Password' checkbox is checked. Under the 'Oracle' tab, the 'Connection Type' is 'Basic' and the 'Role' is 'default'. The 'Hostname' is 'localhost', the 'Port' is '1521', and the 'SID' is 'ORCL'. The 'Service name' is empty. At the bottom, there are checkboxes for 'OS Authentication', 'Kerberos Authentication', and 'Proxy Connection', all of which are unchecked. The 'Status' field is empty. Buttons for 'Help', 'Save', 'Clear', 'Test', 'Connect', and 'Cancel' are at the bottom.

3. Test the new connection. If the Status is Success, connect to the database using the new connection.
 - a. Double-click the *SH Schema* icon on the Connections tabbed page.
 - b. Click the Test button in the New/Select Database Connection window. If the status is Success, click the Connect button.



4. Create a new database connection named *HR Schema*.
 - a. Right-click the *SH Schema* connection in the Object Navigation tree, and select the Properties menu option.
 - b. Enter *HR Schema* as the connection name and *hr* as the username and password, and then click Save. This creates the new connection.
 - c. Repeat step 3 to test the new *HR Schema* connection.

5. Repeat step 4 to create and test a new database connection named OE Schema. Enter oe as the database connection username and password.



Close all the SQL worksheets that are open.

Browsing Your HR, SH, and OE Schemas Tables

1. Browse the structure of the EMPLOYEES table in the HR schema and display its data.
 - a. Expand the HR Schema 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.

Double-click the EMPLOYEES table. The Columns tab displays the columns in the EMPLOYEES table as follows:

The screenshot shows the Oracle SQL Developer interface. On the left, the 'Connections' tree is expanded to 'HR Schema', and the 'EMPLOYEES' table is selected. The main window displays the 'Columns' tab for the 'EMPLOYEES' table. The table structure is as follows:

	COLUMN_NAME	DATA_TYPE	NULLABLE	DATA_DEFAULT	COLUMN_ID	COMMENTS
1	EMPLOYEE_ID	NUMBER(6,0)	No	(null)	1	Primary key of
2	FIRST_NAME	VARCHAR2(20 BYTE)	Yes	(null)	2	First name of
3	LAST_NAME	VARCHAR2(25 BYTE)	No	(null)	3	Last name of
4	EMAIL	VARCHAR2(25 BYTE)	No	(null)	4	Email id of t
5	PHONE_NUMBER	VARCHAR2(20 BYTE)	Yes	(null)	5	Phone number
6	HIRE_DATE	DATE	No	(null)	6	Date when the
7	JOB_ID	VARCHAR2(10 BYTE)	No	(null)	7	Current job c
8	SALARY	NUMBER(8,2)	Yes	(null)	8	Monthly salar
9	COMMISSION_PCT	NUMBER(2,2)	Yes	(null)	9	Commission pe
10	MANAGER_ID	NUMBER(6,0)	Yes	(null)	10	Manager id of
11	DEPARTMENT_ID	NUMBER(4,0)	Yes	(null)	11	Department id

2. Browse the `EMPLOYEES` table and display its data.


To display the employees' data, click the Data tab. The `EMPLOYEES` table data is displayed as follows:

	EMPLOYEE_ID	FIRST_NAME	LAST_NAME	EMAIL	PHONE_NUMBER	HIRE_DATE	JOB_ID
1	100	Steven	King	SKING	515.123.4567	17-JUN-03	AD_P
2	101	Neena	Kochhar	NKOCHHAR	515.123.4568	21-SEP-05	AD_V
3	102	Lex	De Haan	LDEHAAN	515.123.4569	13-JAN-01	AD_V
4	103	Alexander	Hunold	AHUNOLD	590.423.4567	03-JAN-06	IT_P
5	104	Bruce	Ernst	BERNST	590.423.4568	21-MAY-07	IT_P
6	105	David	Austin	DAUSTIN	590.423.4569	25-JUN-05	IT_P
7	106	Valli	Pataballa	VPATABAL	590.423.4560	05-FEB-06	IT_P
8	107	Diana	Lorentz	DLORENTZ	590.423.5567	07-FEB-07	IT_P
9	108	Nancy	Greenberg	NGREENBE	515.124.4569	17-AUG-02	FI_M
10	109	Daniel	Faviet	DFAVIET	515.124.4169	16-AUG-02	FI_A
11	110	John	Chen	JCHEN	515.124.4269	28-SEP-05	FI_A
12	111	Ismael	Sciarra	ISCIARRA	515.124.4369	30-SEP-05	FI_A
13	112	Jose Manuel	Urman	JWURMAN	515.124.4469	07-MAR-06	FI_A
14	113	Luis	Popp	LPOPP	515.124.4567	07-DEC-07	FI_A
15	114	Den	Raphaely	DRAPHEAL	515.127.4561	07-DEC-02	PU_M
16	115	Alexander	Khoo	AKHOO	515.127.4562	18-MAY-03	PU_C
17	116	Shelli	Baida	SBAIDA	515.127.4563	24-DEC-05	PU_C

3. Use the SQL Worksheet to select the last names and salaries of all employees whose annual salary is greater than \$10,000. Use both the Execute Statement (F9) and the Run Script (F5) icons to execute the `SELECT` statement. Review the results of both methods of executing the `SELECT` statements in the appropriate tabs.

Note: Take a few minutes to familiarize yourself with the data, or consult “Appendix B: Table Descriptions,” which provides the description and data for all tables in the HR, SH, and OE schemas that you will use in this course.

Display the SQL Worksheet using either of the following two methods: Select Tools > SQL

Worksheet or click the Open SQL Worksheet icon . The Select Connection window is displayed. Using the HR Database Connection, enter the following statement in the SQL Worksheet area, and then click the Run Script (F5) icon.

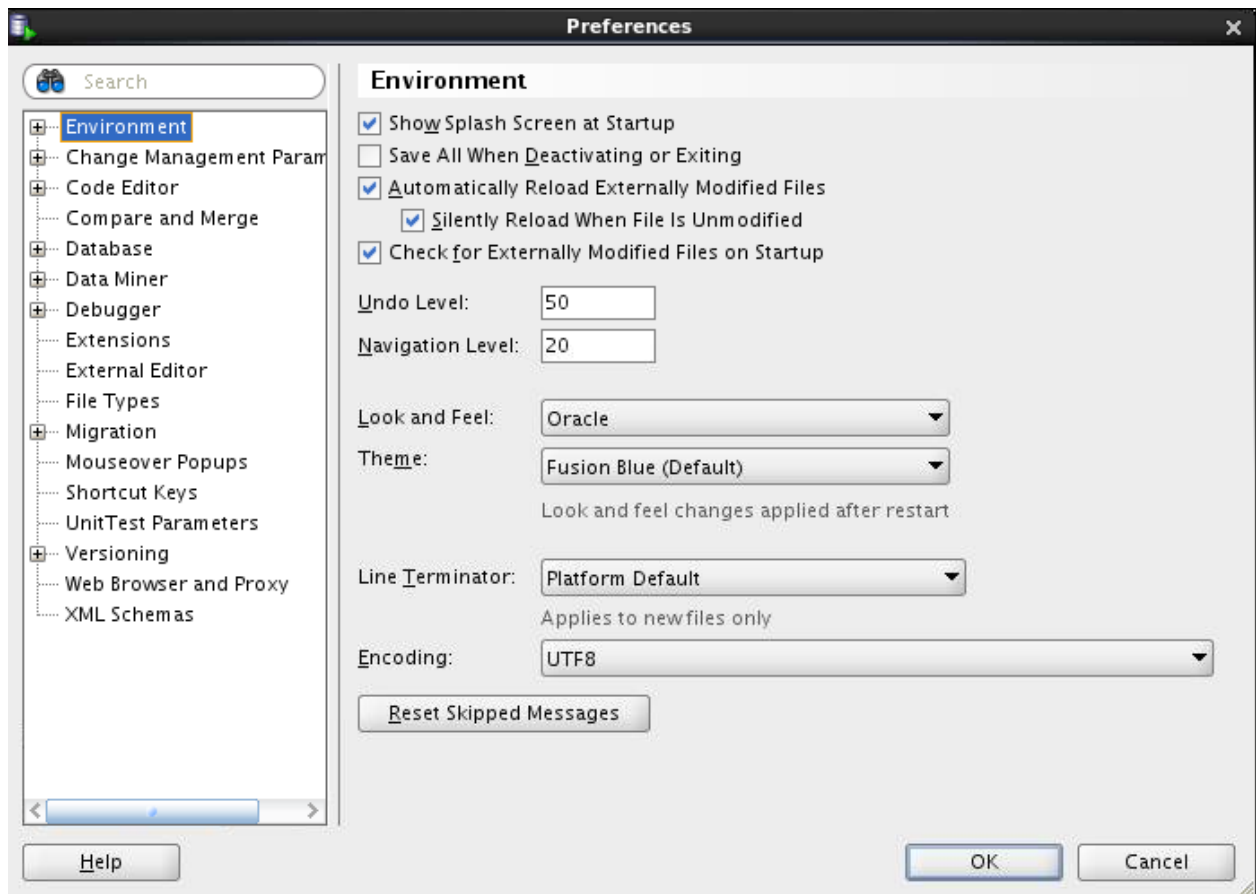
```
SELECT *
FROM employees
WHERE SALARY > 10000;
```

4. Browse the structure of the `SALES` table in the `SH` schema and display its data.
 - a. Double-click the `SH Schema` database connection.
 - b. Expand the `Tables` icon by clicking the plus sign next to it.
 - c. Display the structure of the `SALES` table.
5. Browse the structure of the `ORDERS` table and display its data.
 - a. Double-click the `OE Schema` database connection.
 - b. Expand the `Tables` icon by clicking the plus sign next to it.
 - c. Display the structure of the `ORDERS` table.

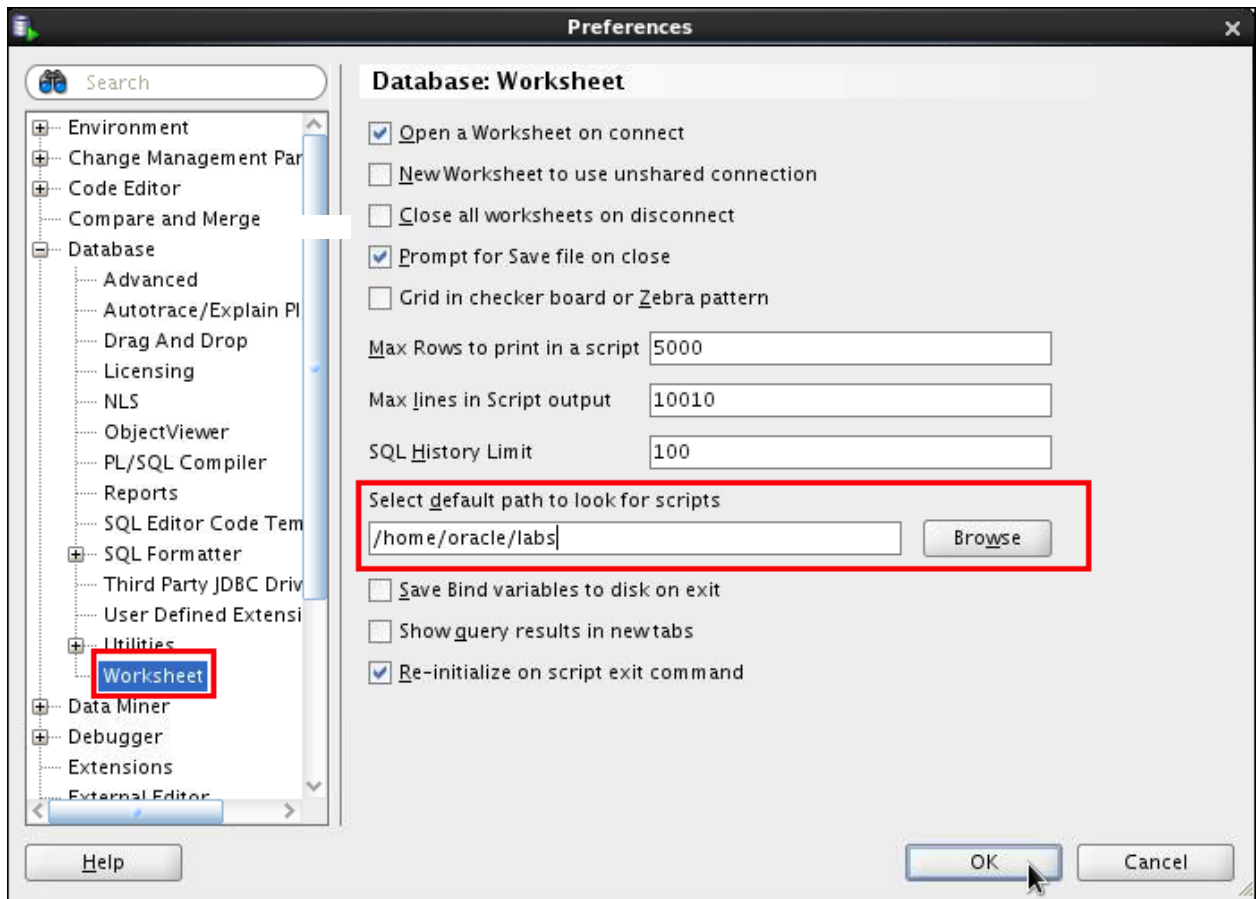
No formal solution.

Setting Some SQL Developer Preferences

1. In the SQL Developer menu, navigate to `Tools > Preferences`. The Preferences window is displayed.

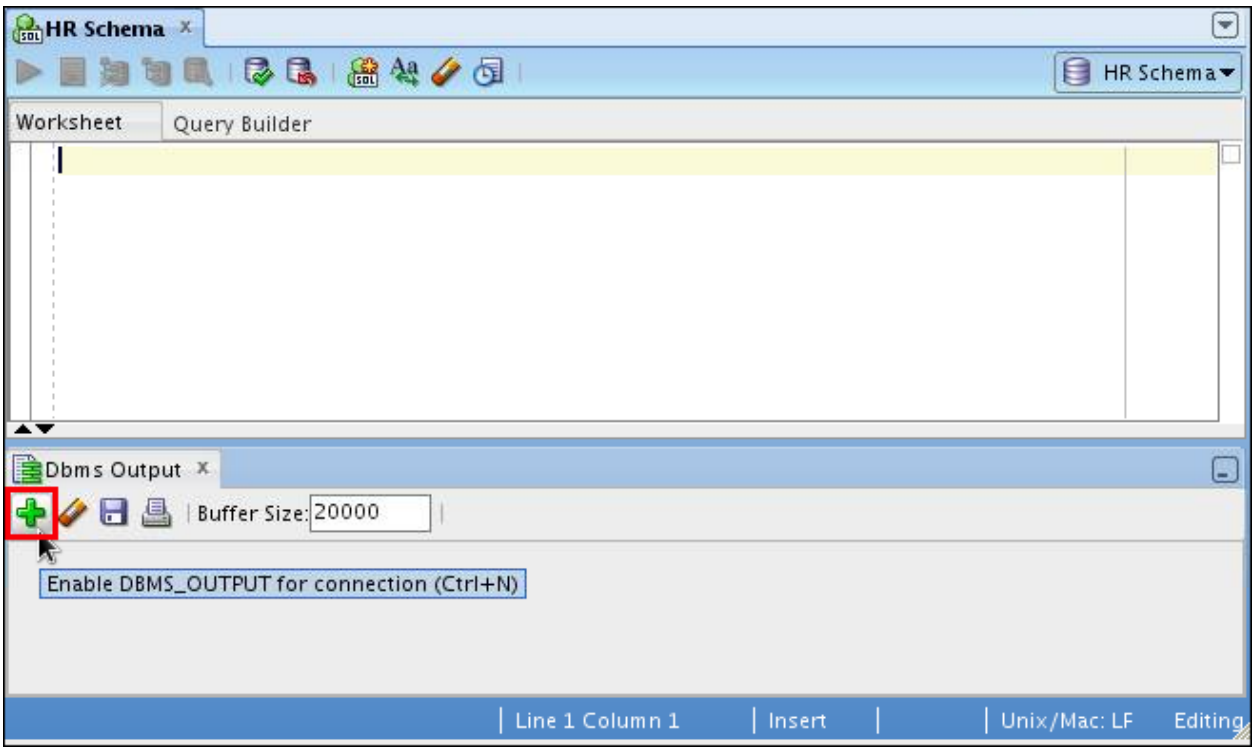


- Click the Worksheet Parameters option under the Database option. In the “Select default path to look for scripts” text box, browse to select the /home/oracle/labs directory, and then click OK. This directory contains the solutions scripts, code examples scripts, and any labs or demonstrations used in this course.



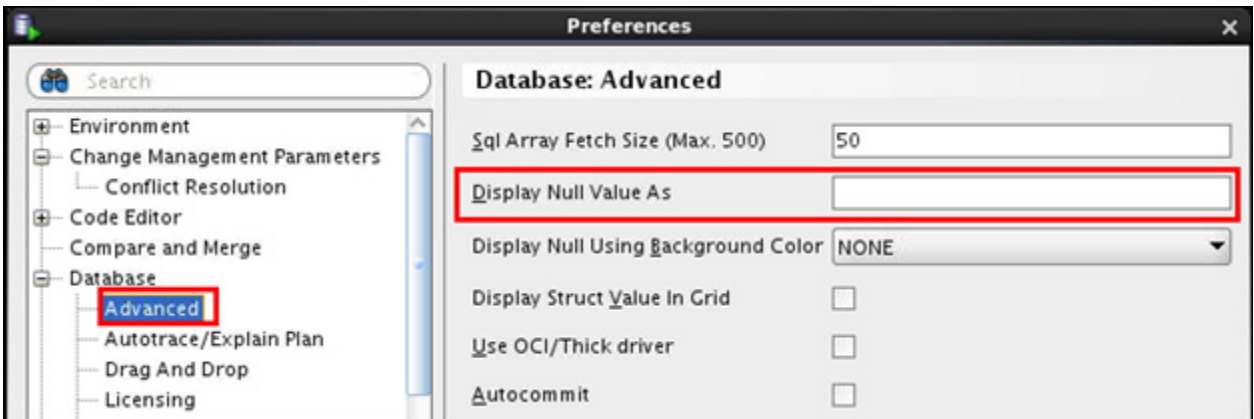
3. Enable `SET SERVEROUTPUT ON` to display the output of the `DBMS_OUTPUT` package statements.

In the SQL Developer menu, click View and select DBMS Output. The DBMS Output tab opens. Click the Enable DBMS Output icon on the toolbar as follows:

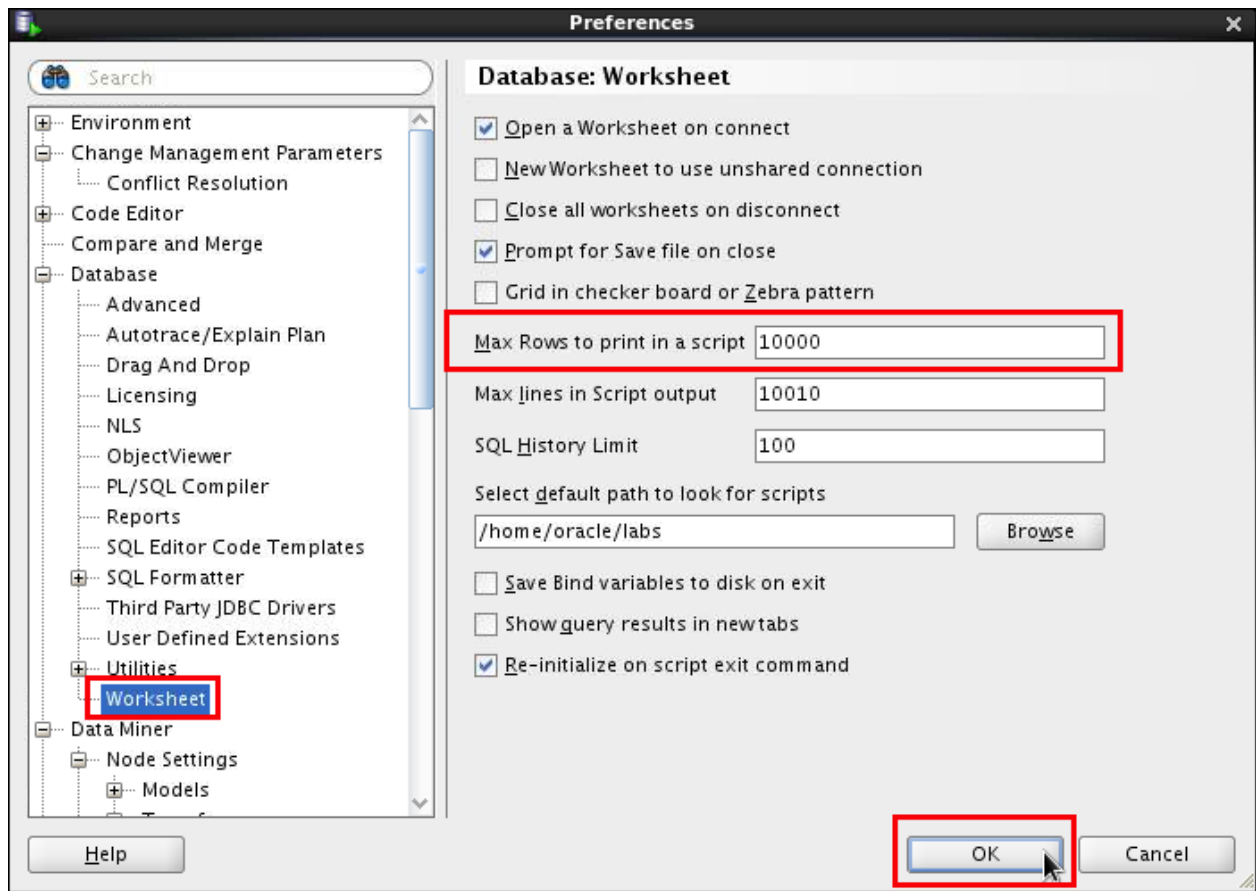


When prompted, select HR Schema from the Select Connection window.

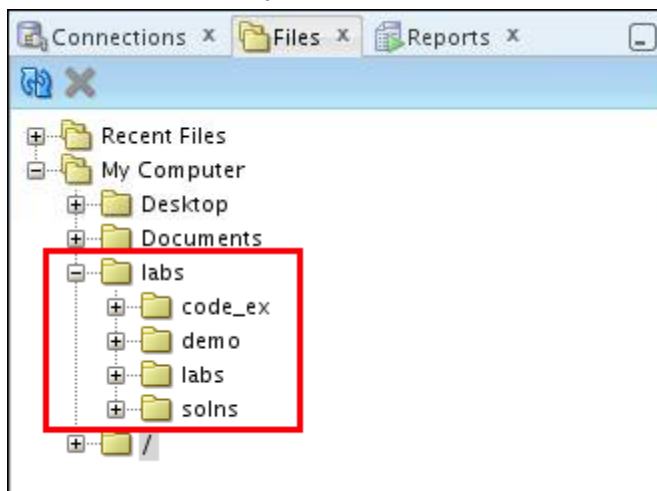
4. In the Preferences window, expand the Database option to display the Advanced option. To display `NULL` values as blanks, remove the (null) from the Display Null Values As text box.



5. In the Preferences window, navigate to Database > Worksheet Parameters. Enter 10000 in the “Max Rows to print in a script” text box, and then click OK to exit the Preferences window.



6. Familiarize yourself with the labs directory:
 - a. Click the Files navigation tab. Expand My Computer > labs directory.
 - b. How many subdirectories do you see in the labs directory?
 - c. Navigate through the directories, and open a script file without executing the code.
 - d. Clear the displayed code in the SQL Worksheet area.



Note: If the Files navigation tab is not displayed, go to tools bar and select View > Files.

Accessing the Oracle Database Online Documentation Library

1. Access the Oracle Database documentation Web page at:
<http://www.oracle.com/pls/db121/homepage>
2. Bookmark the page for easier future access.

Practices for Lesson 2: Grouping and Aggregating Data Using SQL

Chapter 2

Practices for Lesson 2: Overview

Overview

1. In this practice, you will use the `ROLLUP` and `CUBE` operators as extensions of the `GROUP BY` clause
2. In the case study, you will use the `GROUP BY` extension to create a simple cross-tabular report.

Practice 2-1: Grouping and Aggregating Data Using SQL

Overview

In this practice you will inquire

Assumptions

Task

In this practice, you use the `ROLLUP` and `CUBE` operators as extensions of the `GROUP BY` clause. You also use `GROUPING SETS`.

1. Use the `HR` schema to write a query to display the following for those employees whose manager ID is less than 120.
 - a. Manager ID
 - b. Job ID and total salary for every job ID of employees who report to the same manager
 - c. Total salary of all employees reporting to each manager, irrespective of the job IDs
 - d. Your results should look as follows:

	MANAGER_ID	JOB_ID	SUM(SALARY)
1	100	AD_VP	34000
2	100	MK_MAN	13000
3	100	PU_MAN	11000
4	100	SA_MAN	61000
5	100	ST_MAN	36400
6	100		155400
7	101	AC_MGR	12008
8	101	FI_MGR	12008
9	101	HR_REP	6500
10	101	PR_REP	10000
11	101	AD_ASST	4400
12	101		44916
13	102	IT_PROG	9000
14	102		9000
15	103	IT_PROG	19800
16	103		19800
17	108	FI_ACCOUNT	39600
18	108		39600
19	114	PU_CLERK	13900
20	114		13900
21			282616

Connect to the HR Schema using the HR Schema database connection. Open the /home/oracle/labs/solns/sol_02.sql script. Uncomment and select the code under Task 1. Click the Run Statement (F9) icon on the SQL Worksheet toolbar to execute the script. The code is displayed as follows:

```
SELECT manager_id, job_id, sum(salary)
FROM   hr.employees
WHERE  manager_id < 120
GROUP BY ROLLUP(manager_id,job_id);
```

2. Observe the output from step 1. Write a query using the GROUPING function to determine whether the NULL values in the columns corresponding to the GROUP BY expressions are caused by the ROLLUP operation. Compare your results with the following:

	RZ	MGR	RZ	JOB	RZ	SUM(SALARY)	RZ	GROUPING(MANAGER_ID)	RZ	GROUPING(JOB_ID)
1		100		AD_VP		34000		0		0
2		100		MK_MAN		13000		0		0
3		100		PU_MAN		11000		0		0
4		100		SA_MAN		61000		0		0
5		100		ST_MAN		36400		0		0
6		100				155400		0		1
7		101		AC_MGR		12008		0		0
8		101		FI_MGR		12008		0		0
9		101		HR_REP		6500		0		0
10		101		PR_REP		10000		0		0
11		101		AD_ASST		4400		0		0
12		101				44916		0		1
13		102		IT_PROG		9000		0		0
14		102				9000		0		1
15		103		IT_PROG		19800		0		0
16		103				19800		0		1
17		108		FI_ACCOUNT		39600		0		0
18		108				39600		0		1
19		114		PU_CLERK		13900		0		0
20		114				13900		0		1
21						282616		1		1

Uncomment and select the code under Task 2. Click the Run Statement (F9) icon on the SQL Worksheet toolbar to execute the script. The code is displayed as follows:

```
SELECT manager_id MGR , job_id JOB,
sum(salary), GROUPING(manager_id), GROUPING(job_id)
FROM   hr.employees
WHERE  manager_id < 120
GROUP BY ROLLUP(manager_id, job_id);
```

3. Write a query to display the following for those employees whose manager ID is less than 120:
- Manager ID
 - Job and total salaries for every job for employees who report to the same manager
 - Total salary of those managers
 - Cross-tabulation values to display the total salary for every job, irrespective of the manager
 - Total salary irrespective of all job titles
 - Compare your results with the following:

	MANAGER_ID	JOB_ID	SUM(SALARY)
1			282616
2		AD_VP	34000
3		AC_MGR	12008
4		FI_MGR	12008
5		HR_REP	6500
6		MK_MAN	13000
7		PR_REP	10000
8		PU_MAN	11000
9		SA_MAN	61000
10		ST_MAN	36400
11		AD_ASST	4400
12		IT_PROG	28800
13		PU_CLERK	13900
14		FI_ACCOUNT	39600
15	100		155400
16	100	AD_VP	34000
17	100	MK_MAN	13000
18	100	PU_MAN	11000
19	100	SA_MAN	61000
20	100	ST_MAN	36400
21	101		44916
22	101	AC_MGR	12008
23	101	FI_MGR	12008
24	101	HR_REP	6500
25	101	PR_REP	10000
26	101	AD_ASST	4400
27	102		9000
28	102	IT_PROG	9000
29	103		19800
30	103	IT_PROG	19800
31	108		39600
32	108	FI_ACCOUNT	39600
33	114		13900
34	114	PU_CLERK	13900

Uncomment and select the code under Task 3. Click the Run Statement (F9) icon on the SQL Worksheet toolbar to execute the script. The code is displayed as follows:

```
SELECT manager_id, job_id, sum(salary)
FROM   hr.employees
WHERE  manager_id < 120
GROUP BY CUBE(manager_id, job_id);
```

4. Observe the output from question 3. Write a query using the GROUPING function to determine whether the NULL values in the columns corresponding to the GROUP BY expressions are caused by the CUBE operation.

	MGR	JOB	SUM(SALARY)	GROUPING(MANAGER_ID)	GROUPING(JOB_ID)
1			282616	1	1
2		AD_VP	34000	1	0
3		AC_MGR	12008	1	0
4		FI_MGR	12008	1	0
5		HR_REP	6500	1	0
6		MK_MAN	13000	1	0
7		PR_REP	10000	1	0
8		PU_MAN	11000	1	0
9		SA_MAN	61000	1	0
10		ST_MAN	36400	1	0
11		AD_ASST	4400	1	0
12		IT_PROG	28800	1	0
13		PU_CLERK	13900	1	0
14		FI_ACCOUNT	39600	1	0
15	100		155400	0	1
16	100	AD_VP	34000	0	0
17	100	MK_MAN	13000	0	0
18	100	PU_MAN	11000	0	0
19	100	SA_MAN	61000	0	0
20	100	ST_MAN	36400	0	0
21	101		44916	0	1
22	101	AC_MGR	12008	0	0
23	101	FI_MGR	12008	0	0
24	101	HR_REP	6500	0	0
25	101	PR_REP	10000	0	0
26	101	AD_ASST	4400	0	0
27	102		9000	0	1
28	102	IT_PROG	9000	0	0
29	103		19800	0	1
30	103	IT_PROG	19800	0	0
31	108		39600	0	1
32	108	FI_ACCOUNT	39600	0	0
33	114		13900	0	1
34	114	PU_CLERK	13900	0	0

Uncomment and select the code under Task 4. Click the Run Statement (F9) icon on the SQL Worksheet toolbar to execute the script. The code is displayed as follows:

```
SELECT manager_id MGR ,job_id JOB,
sum(salary),GROUPING(manager_id),GROUPING(job_id)
FROM hr.employees
WHERE manager_id < 120
GROUP BY CUBE(manager_id,job_id);
```

5. Using GROUPING SETS, write a query to display the following groupings:
- department_id, manager_id, job_id
 - department_id, job_id
 - manager_id, job_id

The query should calculate the sum of the salaries for each of these groups. Compare your results with the following partial results.

	DEPARTMENT_ID	MANAGER_ID	JOB_ID	SUM(SALARY)
1	90		AD_PRES	24000
2	90	100	AD_VP	34000
3	20	100	MK_MAN	13000
4	30	100	PU_MAN	11000
5	80	100	SA_MAN	61000
6	50	100	ST_MAN	36400
7	110	101	AC_MGR	12008
8	100	101	FI_MGR	12008
9	40	101	HR_REP	6500
10	70	101	PR_REP	10000
11	10	101	AD_ASST	4400
12	60	102	IT_PROG	9000

...

Uncomment and select the code under Task 5. Click the Run Statement (F9) icon on the SQL Worksheet toolbar to execute the script. The code is displayed as follows:

```
SELECT department_id, manager_id, job_id, SUM(salary)
FROM hr.employees
GROUP BY
GROUPING SETS ((department_id, manager_id, job_id),
                (department_id, job_id), (manager_id, job_id));
```

Case Study

Overview

In this case study, you will perform business intelligence queries on the data available in the `SH` schema, `HR` schema and the `OE` schema, for a company named ABC Sales Consultants. This company handles high volume of business information. One of their requirements involves creating business statistics reports to aid in decision support. To achieve this requirement, the IT Database Support group loads sales and customer related data into their data warehouse on quarterly basis.

To help support the ITDB folks (in this case scenario) you will be required to perform business intelligence queries using aggregate functions on the data available in the `SH` schema. First, let us understand the role of aggregate functions.

1. What is the role of aggregate functions in SQL analytics/ decision support systems?

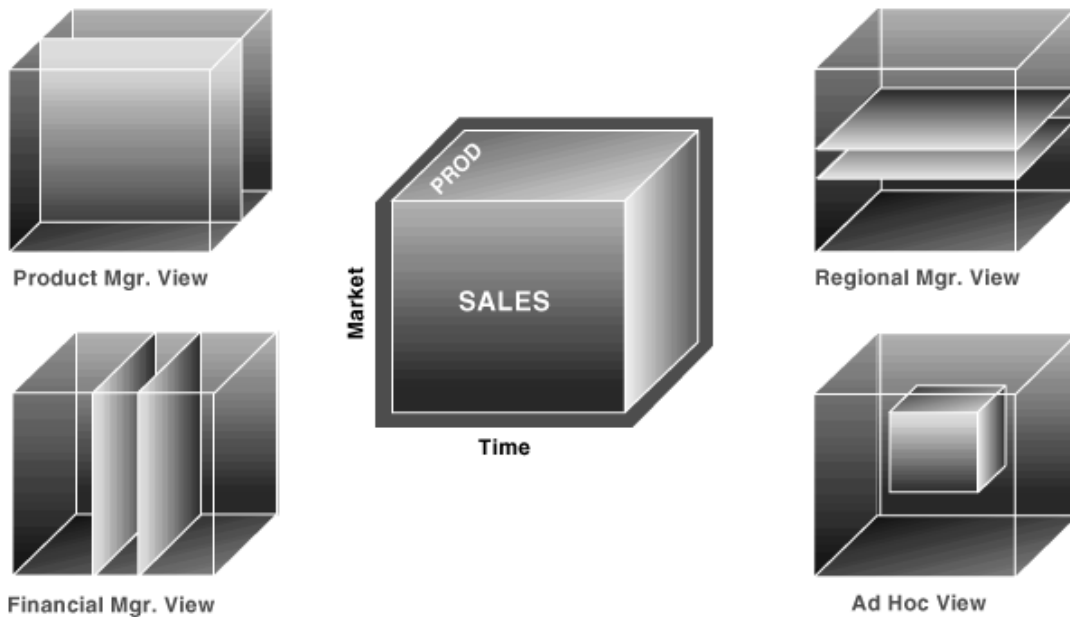
One of the key concepts in decision support systems is multidimensional analysis. That is, examining the enterprise from all necessary combinations of dimensions. The most commonly specified dimensions are time, geography, product, department, and distribution channel. The events or entities associated with a particular set of dimension values are usually referred to as facts. The facts might be sales in units or local currency, profits, customer counts, production volumes, or anything else worth tracking. For example, look at the following multidimensional request:

List the top 10 sales representatives in Asia according to 2000 sales revenue for automotive products, and rank their commissions.

Answering such multidimensional questions often involves accessing and querying huge quantities of data, sometimes in millions of rows. Because the flood of detailed data generated by large organizations cannot be interpreted at the lowest level, aggregated views of the information are essential. Aggregations, such as sums and counts, across many dimensions are vital to multidimensional analyses. Therefore, analytical tasks require convenient and efficient data aggregation.

2. How do you analyse a multi-dimensional request?

To visualize data that has many dimensions, analysts commonly use the analogy of a data cube, that is, a space where facts are stored at the intersection of n dimensions. The cube stores sales data organized by the dimensions of product, market, sales, and time. Note that this is only a metaphor: the actual data is physically stored in normal tables. The cube data consists of both detail and aggregated data.

Logical Cubes and Views by Different Users:**Task**

1. In the SH Schema, create a cross-tabular report showing the total sales by `country_id` and `channel_desc` for the US and France through the Internet and direct sales in September 2000. Compare your results with the following:

	CHANNEL_DESC	COUNTRY_ISO_CODE	SALES\$
1			833,224
2		FR	70,799
3		US	762,425
4	Internet		133,821
5	Internet	FR	9,597
6	Internet	US	124,224
7	Direct Sales		699,403
8	Direct Sales	FR	61,202
9	Direct Sales	US	638,201

Connect to the SH Schema using Choose Db Connection at the top-right corner of the SQL worksheet. Uncomment and select the code under Task 2 of case study. Click the Run Statement (F9) icon on the SQL Worksheet toolbar to execute the script. The code is displayed as follows:

```
SELECT channels.channel_desc, countries.country_iso_code,
       TO_CHAR(SUM(amount_sold), '9,999,999,999') SALES$
```

```
FROM sales, customers, times, channels, countries
WHERE sales.time_id=times.time_id AND
      sales.cust_id=customers.cust_id AND
      sales.channel_id= channels.channel_id AND
      channels.channel_desc IN
      ('Direct Sales', 'Internet') AND
      times.calendar_month_desc='2000-09'
      AND customers.country_id=countries.country_id
      AND countries.country_iso_code IN ('US','FR')
GROUP BY CUBE(channels.channel_desc,
              countries.country_iso_code);
```

Practices for Lesson 3: Hierarchical Retrieval

Chapter 3

Practices for Lesson 3: Overview

Overview

1. In this practice, you will produce hierarchical reports.
2. In the case study, you will create company organization charts.

Practice 3-1: Hierarchical Retrieval

Overview

In this practice you will inquire

Assumptions

Task

In this practice, you gain practical experience in producing hierarchical reports.

- Using the HR schema, produce a report showing an organization chart for Mourgos's department. Print the last names, salaries, and department IDs. Your results should look similar to the following:

LAST_NAME	SALARY	DEPARTMENT_ID
Mourgos	5800	50
Rajs	3500	50
Davies	3100	50
Matos	2600	50
Vargas	2500	50
Walsh	3100	50
Feeney	3000	50
OConnell	2600	50
Grant	2600	50
9 rows selected		

Connect to the HR Schema using the HR Schema database connection. Open the /home/oracle/labs/solns/sol_03.sql script. Uncomment and select the code under Task 1. Click the Run Statement (F9) icon on the SQL Worksheet toolbar to execute the script. The code is displayed as follows:

```
SELECT last_name, salary, department_id
FROM hr.employees
START WITH last_name = 'Mourgos'
CONNECT BY PRIOR employee_id = manager_id;
```

- Create a report that shows the hierarchy of the managers for the employee Lorentz. Display his immediate manager first. Your results should look similar to the following:

LAST_NAME

Hunold
De Haan
King

Uncomment and select the code under Task 2. Click the Run Script (F5) icon on the SQL Worksheet toolbar to execute the script. The code is displayed as follows:

```
SELECT last_name
FROM hr.employees
WHERE last_name != 'Lorentz'
START WITH last_name = 'Lorentz'
CONNECT BY PRIOR manager_id = employee_id;
```

3. Create an indented report showing the management hierarchy starting from the employee whose LAST_NAME is Kochhar. Print the employee's last name, manager ID, and department ID. Give alias names to the columns as shown in the output below.

	NAME	MGR	DEPTNO
1	Kochhar	100	90
2	__Greenberg	101	100
3	___Faviet	108	100
4	___Chen	108	100
5	___Sciarra	108	100
6	___Urman	108	100
7	___Popp	108	100
8	__Whalen	101	10
9	__Mavris	101	40
10	__Baer	101	70
11	__Higgins	101	110
12	___Gietz	205	110

Uncomment and select the code under Task 3. Click the Run Statement (F9) icon on the SQL Worksheet toolbar to execute the script. The code is displayed as follows:

```
SELECT LPAD(last_name, LENGTH(last_name)+(LEVEL*2)-2, '_')
       name, manager_id mgr, department_id deptno
FROM hr.employees
START WITH last_name = 'Kochhar'
CONNECT BY PRIOR employee_id = manager_id;
```


Case Study

Scenario

The ITDB folks are facing issues in generating company organization charts. They need to generate the following two charts:

- a. A company organization chart that shows the management hierarchy.
- b. A company organization chart that shows the management hierarchy excluding:
 - i. All employees with a job ID of IT_PROG
 - ii. Manager De Haan and those employees who report to De Haan.

The task below would help them generate the above mentioned charts:

1. Generate a company organization chart that shows the management hierarchy. Start with the person at the top level. Compare your results with the following output:

Query Result x			
SQL Fetched 100 rows in 0.036 seconds			
	LAST_NAME	EMPLOYEE_ID	MANAGER_ID
1	King	100	
2	Kochhar	101	100
3	Greenberg	108	101
4	Faviet	109	108
5	Chen	110	108
6	Sciarra	111	108
7	Urman	112	108
8	Popp	113	108
9	Whalen	200	101
10	Mavris	203	101
11	Baer	204	101
12	Higgins	205	101
13	Gietz	206	205
14	De Haan	102	100
...			
106	Hartstein	201	100
107	Fay	202	201

Connect to the HR Schema using Choose Db Connection at the top-right corner of the SQL worksheet. Uncomment and select the code under Task 1 of case study. Click the Run Statement (F9) icon on the SQL Worksheet toolbar to execute the script. The code is displayed as follows:

```
SELECT last_name, employee_id, manager_id
FROM   hr.employees
START WITH manager_id IS NULL
CONNECT BY PRIOR employee_id = manager_id;
```

2. Update the query from Task1 to display the company organization chart excluding:
 - a. All employees with a job ID of IT_PROG
 - b. Manager De Haan and his directs.

Compare your results with the following output:

Query Result x			
SQL Fetched 50 rows in 0.038 seconds			
	LAST_NAME	EMPLOYEE_ID	MANAGER_ID
1	King	100	
2	Kochhar	101	100
3	Greenberg	108	101
4	Faviet	109	108
5	Chen	110	108
6	Sciarra	111	108
7	Urman	112	108
8	Popp	113	108
9	Whalen	200	101
10	Mavris	203	101
11	Baer	204	101
12	Higgins	205	101
13	Gietz	206	205
14	Raphaely	114	100
...			
100	Hartstein	201	100
101	Fay	202	201

Uncomment and select the code under Task 2 of case study. Click the Run Statement (F9) icon on the SQL Worksheet toolbar to execute the script. The code is displayed as follows:

```
SELECT last_name, employee_id, manager_id
FROM   hr.employees
WHERE  job_id != 'IT_PROG'
START WITH manager_id IS NULL
CONNECT BY PRIOR employee_id = manager_id
        AND last_name != 'De Haan';
```

Practices for Lesson 4: Working with Regular Expressions

Chapter 4

Practices for Lesson 4: Overview

Overview

1. In this practice, you will use regular expressions functions to search for, replace, and manipulate data in the `HR` schema.
2. In the case study, you will help the ITDB folks to process information in the `contacts` and `people` table.

Practice 4-1: Analyzing Data Using Regular Expressions

Overview



In this practice you will inquire

Assumptions

Task

In this practice, you use regular expressions functions to search for, replace, and manipulate data in the HR schema. 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 employees whose first names start with “Ki” or “Ko.”

	 FIRST_NAME	 LAST_NAME
1	Janette	King
2	Steven	King
3	Neena	Kochhar

Connect to the HR Schema using the HR Schema database connection. Open the /home/oracle/labs/solns/sol_04.sql script. Uncomment and select the code under Task 1. Click the Run Statement (F9) icon on the SQL Worksheet toolbar to execute the script. The code is displayed as follows:

```
SELECT first_name, last_name
FROM hr.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.

	Street Address
1	1297ViaColadiRie
2	93091CalledeIlaTesta
3	2017Shinjuku-ku
4	9450Kamiya-cho
5	2014JabberwockyRd
6	2011InteriorsBlvd
7	2007ZagoraSt
8	2004CharadeRd
9	147SpadinaAve
10	6092BoxwoodSt
11	40-5-12Laogianggen
12	1298Vileparle(E)
13	12-98VictoriaStreet
14	198ClementiNorth
15	8204ArthurSt
16	MagdalenCentre,TheOxfordSciencePark
17	9702ChesterRoad
18	Schwanthalerstr.7031
19	RuaFreiCaneca1360
20	20RuedesCorps-Saints
21	Murtenstrasse921
22	PieterBreughelstraat837
23	MarianoEscobedo9991

Uncomment and select the code under Task 2. Click the Run Statement (F9) icon on the SQL Worksheet toolbar to execute the script. The code is displayed as follows:

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

3. Create a query that searches for the first uppercase character in the `STREET_ADDRESS` column of the `LOCATIONS` table for all street addresses starting with "14" or "12." Use "First_Upper_Position" as the column heading.

	STREET_ADDRESS	First_Upper_Position
1	1297 Via Cola di Rie	6
2	147 Spadina Ave	5
3	1298 Vileparle (E)	6
4	12-98 Victoria Street	7

Uncomment and select the code under Task 3. Click the Run Statement (F9) icon on the SQL Worksheet toolbar to execute the script. The code is displayed as follows:

```
SELECT street_address, REGEXP_INSTR(street_address, '[:upper:]')
   AS First_Upper_Position
FROM   locations
WHERE  REGEXP_LIKE (street_address, '^1(4|2).');
```

4. Create a query that replaces “St” with “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

Uncomment and select the code under Task 4. Click the Run Statement (F9) icon on the SQL Worksheet toolbar to execute the script. The code is displayed as follows:

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

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

	COUNT_DNA
1	2

Uncomment and select the code under Task 7. Click the Run Statement (F9) icon on the SQL Worksheet toolbar to execute the script. The code is displayed as follows:

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

Case Study

Scenario

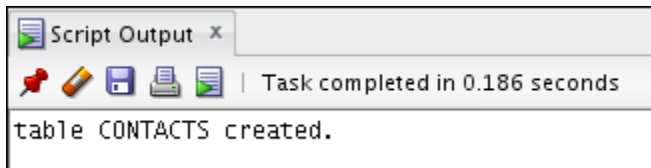
The ITDB folks are facing issues while processing information in the `contacts` and `people` table. They notice the following two discrepancies:

- No standard format enforced to insert the phone numbers in the `contacts` table.
- Readability issues: Names that are in the format “first middle last” needs to be repositioned to the format “last, first middle” in the `names` table.

The task below would help them resolve the above mentioned issues:

Task

- 1) Ensure that phone numbers are entered into the database in a standard format by using a constraint to enforce a phone number format on the `contacts` table.
 - a) Create a `contacts` table with a check constraint to the `p_number` column to ensure that phone numbers are entered into the database in the following format: **(xxx) xxx-xxxx**. The table should have the following columns:
 - i) `l_name varchar2(30)`
 - ii) `p_number varchar2(30)`



Connect to the SH Schema using Choose Db Connection at the top-right corner of the SQL worksheet. Uncomment and select the code under Task 1_a of case study. Click the Run Script (F5) icon on the SQL Worksheet toolbar to execute the script. The code is displayed as follows:

```
DROP TABLE contacts;

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}$' ) )
);
```

Note: If you are executing the code for the first time, ignore the error message caused due to the drop table statement.

- b) Insert the following seven last names and phone numbers into the `contacts` table.

l_name Column Value	p_number Column Value
NULL	' (111) 555-5555'
NULL	' (222) 555-5555'
NULL	'333 555-5555'
NULL	'444 555 5555'
NULL	'555-555-5555'
NULL	' (666) 555-5555'
NULL	' (777) 555-5555'

c) Check which numbers are added?

Uncomment and select the code under Task 1_c of case study. Click the Run Statement (F9) icon on the SQL Worksheet toolbar to execute the script. The output is displayed as follows:

```
1 rows inserted.
```

```
1 rows inserted.
```

```
Error starting at line 7 in command:
```

```
insert into contacts
```

```
values (null, '333 555 5555')
```

```
Error report:
```

```
SQL Error: ORA-02290: check constraint (SH.P_NUMBER_FORMAT) violated
```

```
02290. 00000 - "check constraint (%s.%s) violated"
```

```
*Cause:      The values being inserted do not satisfy the named check
```

```
*Action:     do not insert values that violate the constraint.
```

```
Error starting at line 10 in command:
```

```
insert into contacts
```

```
values (null, '444 555 5555')
```

Error report:

SQL Error: ORA-02290: check constraint (SH.P_NUMBER_FORMAT) violated
02290. 00000 - "check constraint (%s.%s) violated"

*Cause: The values being inserted do not satisfy the named check

*Action: do not insert values that violate the constraint.

Error starting at line 13 in command:

insert into contacts

values (null, '555-555-5555')

Error report:

SQL Error: ORA-02290: check constraint (SH.P_NUMBER_FORMAT) violated
02290. 00000 - "check constraint (%s.%s) violated"

*Cause: The values being inserted do not satisfy the named check

*Action: do not insert values that violate the
constraint.

Error starting at line 16 in command:

insert into contacts

values (null, '((666)555-5555')

Error report:

SQL Error: ORA-02290: check constraint (SH.P_NUMBER_FORMAT) violated
02290. 00000 - "check constraint (%s.%s) violated"

*Cause: The values being inserted do not satisfy the named check

*Action: do not insert values that violate the constraint.

Error starting at line 19 in command:

insert into contacts

values (null, ' (777) 555-5555')

Error report:

SQL Error: ORA-02290: check constraint (SH.P_NUMBER_FORMAT) violated

```
02290. 00000 - "check constraint (%s.%s) violated"
```

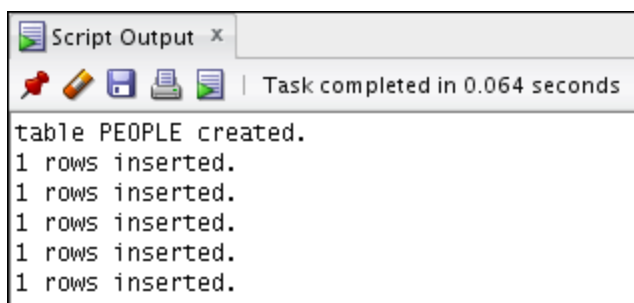
*Cause: The values being inserted do not satisfy the named check

*Action: do not insert values that violate the constraint.

You will notice that the first two INSERT statements use a format that conforms to the p_number_format constraint; the remaining statements generate CHECK constraint errors.

2) Reposition names that are in the format first middle last to the format last, first middle in the people table.

a) Create the people table with the names VARCHAR2(20) column. Populate it with names in the first middle last format.



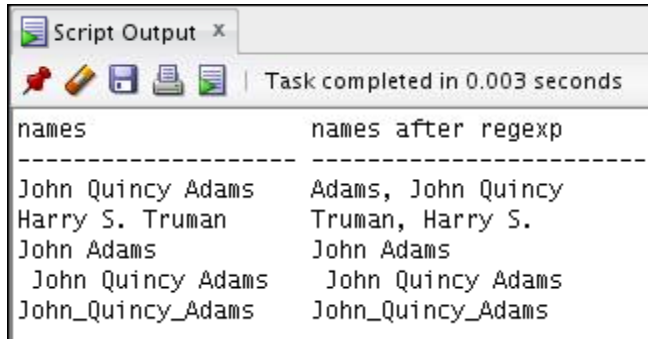
Uncomment and select the code under Task 2_a of case study. Click the Run Script (F5) icon on the SQL Worksheet toolbar to execute the script. The code is displayed as follows:

```
DROP TABLE people;

CREATE TABLE people (names VARCHAR2(20));

INSERT INTO people VALUES ('John Quincy Adams');
INSERT INTO people VALUES ('Harry S. Truman');
INSERT INTO people VALUES ('John Adams');
INSERT INTO people VALUES (' John Quincy Adams');
INSERT INTO people VALUES ('John_Quincy_Adams');
```

- b) Create a `SELECT` statement using the `REGEXP_REPLACE` function to reposition the names into the last, first middle format. Compare your result with the following:



names	names after regexp
John Quincy Adams	Adams, John Quincy
Harry S. Truman	Truman, Harry S.
John Adams	John Adams
John Quincy Adams	John Quincy Adams
John_Quincy_Adams	John_Quincy_Adams

Uncomment and select the code under Task 2_b of case study. Click the Run Script (F5) icon on the SQL Worksheet toolbar to execute the script. The code is displayed as follows:

```
SELECT names "names",
       REGEXP_REPLACE(names, '^(\\S+)\\s(\\S+)\\s(\\S+)$', '\\3, \\1 \\2')
       AS "names after regexp"
FROM people;
```

Explanation of the regular expression pattern in the above query:

Regular Expression Element	Description
^	Matches the beginning of the string.
\$	Matches the end of the string.
(\\S+)	Matches one or more nonspace characters. The parentheses are not escaped so they function as a grouping expression.
\\s	Matches a whitespace character.
\\1	Substitutes the first subexpression, that is, the first group of parentheses in the matching pattern.
\\2	Substitutes the second subexpression, that is, the second group of parentheses in the matching pattern.
\\3	Substitutes the third subexpression, that is, the third group of parentheses in the matching pattern.

Practices for Lesson 5: Analyzing and Reporting Data by Using SQL

Chapter 5

Practices for Lesson 5: Overview

Overview

1. In the practice, you will use the `RANK` function, the `LISTAGG` function and the `row_limiting_clause` in the `HR` schema.
2. In the case study, you will generate statistical reports using analytical and reporting functions.

Practice 5-1: Analyzing and Reporting Data Using SQL

Overview

In this practice you will inquire

Assumptions

Task

In this practice, you use the RANK function, the LISTAGG function and the row_limiting_clause in the HR schema. If you encounter any errors, you can use the Compiler-Log tab in SQL Developer.

1. Use the RANK function to rank the employees in the employees table (in the HR schema) in department 80 based on their salary and commission. Your results should look similar to the following:

DEPARTMENT_ID	LAST_NAME	SALARY	COMMISSION_PCT	Rank
80	Abel	11000	0.3	5
80	Ande	6400	0.1	31
80	Banda	6200	0.1	32
80	Bates	7300	0.15	26
80	Bernstein	9500	0.25	14
80	Bloom	10000	0.2	9
80	Cambrault	7500	0.2	23
80	Cambrault	11000	0.3	5
80	Doran	7500	0.3	24
80	Erazuriz	12000	0.3	3
80	Fox	9600	0.2	12
80	Greene	9500	0.15	13
80	Hall	9000	0.25	16
80	Hutton	8800	0.25	18
80	Johnson	6200	0.1	32
80	King	10000	0.35	11
80	Kumar	6100	0.1	34
80	Lee	6800	0.1	30
80	Livingston	8400	0.2	20
80	Marvins	7200	0.1	27
80	McEwen	9000	0.35	17
80	Olsen	8000	0.2	21
80	Ozer	11500	0.25	4
80	Partners	13500	0.3	2
80	Russell	14000	0.4	1
80	Sewall	7000	0.25	29
80	Smith	7400	0.15	25
80	Smith	8000	0.3	22
80	Sully	9500	0.35	15
80	Taylor	8600	0.2	19
80	Tucker	10000	0.3	10
80	Tuvault	7000	0.15	28
80	Vishney	10500	0.25	8
80	Zlotkey	10500	0.2	7
34 rows selected				

Connect to the HR Schema using the HR Schema database connection. Open the /home/oracle/labs/solns/sol_05.sql script. Uncomment and select the code under

Task 1. Click the Run Statement (F9) icon on the SQL Worksheet toolbar to execute the script. The code is displayed as follows:

```
SELECT department_id, last_name, salary, commission_pct,
       RANK() OVER (PARTITION BY department_id
                   ORDER BY salary DESC, commission_pct) "Rank"
FROM hr.employees
WHERE department_id = 80
ORDER BY department_id, last_name, salary, commission_pct, "Rank";
```

2. Use the LISTAGG function to list the employees in the employees table (in the HR schema) in department 30 based on their salary and commission. Your results should look similar to the following:

Emp_list	Least
1 Colmenares; Himuro; Tobias; Baida; Khoo; Raphaely	2500

Uncomment and select the code under Task 2. Click the Run Statement (F9) icon on the SQL Worksheet toolbar to execute the script. The code is displayed as follows:

```
SELECT LISTAGG(last_name, ';' )
       WITHIN GROUP (ORDER BY salary, commission_pct) "Emp_list",
       MIN(salary) "Least"
FROM employees
WHERE department_id = 30;
```

3. Use the RATIO_TO_REPORT function to calculate the ratio of the quantity of products sold for the product 1797 in the order_items table (in the OE schema). Your results should look similar to the following:

PRODUCT_ID	QUANTITY	Ratio
1797	9	0.3214285714
1797	7	0.25
1797	12	0.4285714286

Connect to the OE Schema. Uncomment and select the code under Task 3. Click the Run Statement (F9) icon on the SQL Worksheet toolbar to execute the script. The code is displayed as follows:

```
SELECT product_id, quantity, RATIO_TO_REPORT(quantity)
       OVER () AS "Ratio of Products"
FROM order_items
WHERE product_id = 1797
ORDER BY product_id;
```


4. Observe the output from step 1. Use the `row_limiting_clause` to display only the first 5 ranks from the result using the HR schema. Compare your results with the following:

DEPARTMENT_ID	LAST_NAME	SALARY	COMMISSION_PCT	Rank
80	Russell	14000	0.4	1
80	Partners	13500	0.3	2
80	Errazuriz	12000	0.3	3
80	Ozer	11500	0.25	4
80	Cambrault	11000	0.3	5

Connect to the HR Schema. Uncomment and select the code under Task 4. Click the Run Script (F5) icon on the SQL Worksheet toolbar to execute the script. The code is displayed as follows:

```
SELECT department_id, last_name, salary, commission_pct,
       RANK() OVER (PARTITION BY department_id
                    ORDER BY salary DESC, commission_pct) "Rank"
FROM hr.employees
WHERE department_id = 80
ORDER BY "Rank"
FETCH FIRST 5 ROWS ONLY;
```

5. Use the `row_limiting_clause` to display the next 5 ranks from the result. Compare your results with the following:

DEPARTMENT_ID	LAST_NAME	SALARY	COMMISSION_PCT	Rank
80	Abel	11000	0.3	5
80	Zlotkey	10500	0.2	7
80	Vishney	10500	0.25	8
80	Bloom	10000	0.2	9
80	Tucker	10000	0.3	10

Uncomment and select the code under Task 5. Click the Run Script (F5) icon on the SQL Worksheet toolbar to execute the script. The code is displayed as follows:

```
SELECT department_id, last_name, salary, commission_pct,
       RANK() OVER (PARTITION BY department_id
                    ORDER BY salary DESC, commission_pct) "Rank"
FROM hr.employees
WHERE department_id = 80
ORDER BY "Rank"
OFFSET 5 ROWS FETCH NEXT 5 ROWS ONLY;
```

Case Study

Scenario

The ITDB folks create statistical reports using the analytical and reporting functions. These reports will be used by the business analysts to apply strategies to improve business in areas where the products sell less.

The tasks mentioned below will help the ITDB find the following:

1. The region which had minimum sales for each product category.
2. The top 5 selling products for each product subcategory

Task




1. For each product category, find the region in which it had minimum sales. Compare your result with the following:

	PROD_CATEGORY	COUNTRY_REGION	SALES
1	Electron	Americas	581.92
2	Hardware	Americas	925.93
3	Peripher	Oceania	940.43
4	Software	Oceania	890.25

Connect to the SH Schema using Choose Db Connection at the top-right corner of the SQL worksheet. Uncomment and select the code under Task 1 of case study. Click the Run Statement (F9) icon on the SQL Worksheet toolbar to execute the script. The code is displayed as follows:

```
SELECT prod_category, country_region, sales
FROM (SELECT SUBSTR(p.prod_category,1,8)
      AS prod_category, co.country_region,
      SUM(amount_sold) AS sales,
      MIN(SUM(amount_sold))
      OVER (PARTITION BY prod_category) AS MIN_REG_SALES
FROM sales s, customers c, countries co, products p
WHERE s.cust_id=c.cust_id
      AND c.country_id=co.country_id
      AND s.prod_id =p.prod_id
      AND s.time_id = TO_DATE('11-OCT-2001'))
GROUP BY prod_category, country_region)
WHERE sales = MIN_REG_SALES;
```

2. Find the top-selling channels in US, in the last quarter of the year 2000.

	 CHANNEL_DESC	 SALES\$	 RANKING
1	Direct Sales	1,851,511	1
2	Partners	1,147,564	2
3	Internet	511,124	3

Uncomment and select the code under Task 2 of case study. Click the Run Statement (F9) icon on the SQL Worksheet toolbar to execute the script. The code is displayed as follows:

```
SELECT channel_desc, SUM(amount_sold) SALES$,
       RANK() OVER (ORDER BY SUM(amount_sold)desc) AS ranking
FROM sales, products, customers, times, channels, countries
WHERE sales.prod_id=products.prod_id
      AND sales.cust_id=customers.cust_id
      AND customers.country_id = countries.country_id
      AND sales.time_id=times.time_id
      AND sales.channel_id=channels.channel_id
      AND times.calendar_month_desc IN ('2000-10', '2000-11',
    '2000-12')
      AND country_iso_code='US'
GROUP BY channel_desc;
```


Practices for Lesson 6: Performing Pivoting and Unpivoting Operations

Chapter 6

Practice 6-1: Performing Pivoting and Unpivoting Operations

Overview

1. In this practice, you will gain experience to use the `PIVOT` and `UNPIVOT` clauses in the `OE` schema.
2. In the case study, you will find the following:
 - a. Customer sales in each month for each product
 - b. Product sales for each customer in each month

Practice 6-1: Performing Pivoting and Unpivoting Operations

Overview

In this practice you will inquire

Assumptions

Tasks

In this practice, you use the `PIVOT` and `UNPIVOT` clauses in the `OE` schema. If you encounter any errors, you can use the Compiler-Log tab in SQL Developer.

1. The orders table in the `OE` schema contains information about when an order was placed (`order_date`), how it was placed (`order_mode`), and the total amount of the order (`order_total`), as well as other information. Use the `PIVOT` clause to pivot `order_mode` values into columns, aggregating `order_total` data in the process, to get yearly totals by order mode.

Connect to the `OE` schema and create a new table named `pivot_table` in your schema using a subquery and insert the rows returned by the subquery from the orders table into the newly created table. Use the alias `STORE` for the direct pivot column value, and the alias `INTERNET` for the online pivot column value. Query the `pivot_table` table and view the results. Compare your results with the following:

YEAR	STORE	INTERNET
2004	5546.6	
2006	371895.5	100056.6
2007	1274078.8	1271019.5
2008	252108.3	393349.4

Connect to the `OE` schema using the `OE` Schema database connection. Open the `/home/oracle/labs/solns/sol_06.sql` script. Uncomment and select the code under Task 1. Click the Run Script (F5) icon on the SQL Worksheet toolbar to execute the script. The code is displayed as follows:

```
DROP TABLE pivot_table;

CREATE TABLE pivot_table AS
SELECT * FROM
  (SELECT EXTRACT(YEAR FROM order_date) year, order_mode,
    order_total FROM oe.orders)
PIVOT
  (SUM(order_total) FOR order_mode IN ('direct' AS Store, 'online'
    AS Internet))
ORDER BY year;

SELECT * FROM pivot_table ORDER BY year;
```

Note: If you are executing the code for the first time, ignore the error message caused due to the drop table statement. Also, ignore the warning messages when executing the `CREATE TABLE` statement.

2. Use the UNPIVOT clause in a query to restore the `pivot_table` table so that selected columns are pivoted into values in a single column. Use the alias `YEARLY_TOTALS` for the unpivoted `ORDER_MODE` column. Compare your results with the following:

YEAR	ORDER_MODE	YEARLY_TOTAL
2004	direct	5546.6
2006	direct	371895.5
2006	online	100056.6
2007	direct	1274078.8
2007	online	1271019.5
2008	direct	252108.3
2008	online	393349.4

7 rows selected

Uncomment and select the code under Task 2. Click the Run Script (F9) icon on the SQL Worksheet toolbar to execute the script. The code is displayed as follows:

```
SELECT *
FROM pivot_table
     UNPIVOT (yearly_total FOR order_mode IN (store AS 'direct',
                                              internet AS 'online'))
ORDER BY year, order_mode;
```


Case Study

Scenario

For the purpose of generating quarterly reports, ABC Sales Consultants Company maintains the track record of customer sales for the following three products:

Prod_A, Prod_B, Prod_C.

In this scenario, the sales data is collected for three privileged customers `cust1`, `cust2` and `cust3`, for the months of January, February, and March. The relational table `SALES_INFO` stores the data for each customer against each product in each month.

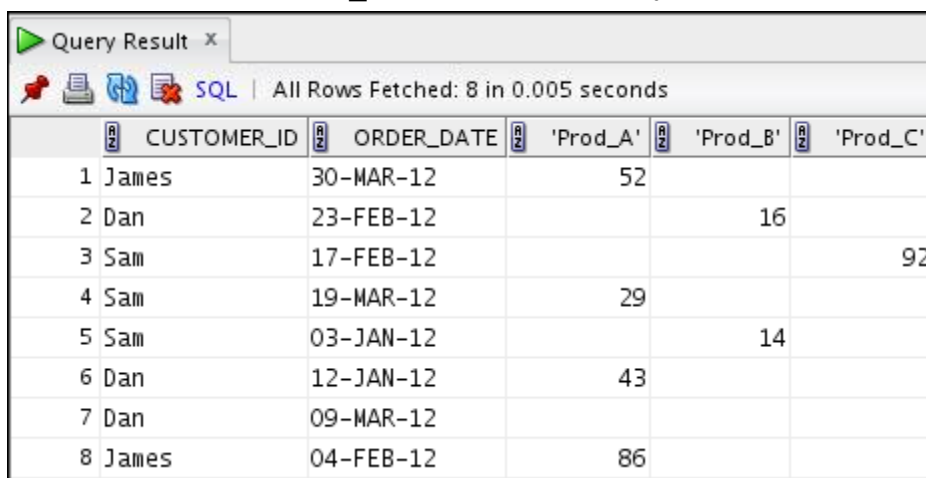
The tasks mentioned below will help the ITDB folks query the existing `SALES_INFO` table to find the following:

1. Customer sales in each month for each product
2. Product sales for each customer in each month

Note: Execute `lab_06_01.sql` script from `/home/oracle/labs/labs` directory using the `OE` schema before performing the tasks below. The `lab_06_01.sql` script creates and populates values into the `event_log` table.

Task

1. Create a query using the `PIVOT` clause to find the customer sales in each month for each product in the `sales_info` table. Compare your results with the following:



	CUSTOMER_ID	ORDER_DATE	'Prod_A'	'Prod_B'	'Prod_C'
1	James	30-MAR-12	52		
2	Dan	23-FEB-12		16	
3	Sam	17-FEB-12			92
4	Sam	19-MAR-12	29		
5	Sam	03-JAN-12		14	
6	Dan	12-JAN-12	43		
7	Dan	09-MAR-12			
8	James	04-FEB-12	86		

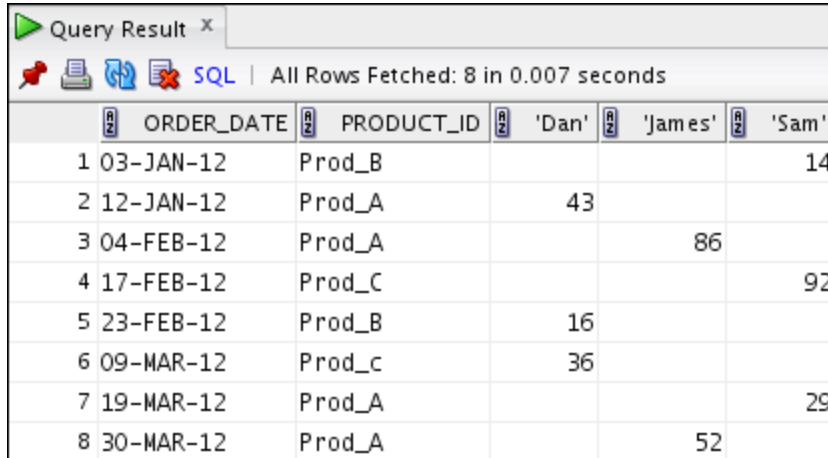
Connect to the OE Schema using Choose Db Connection at the top-right corner of the SQL worksheet. Uncomment and select the code under Task 1 of case study. Click the Run Script (F5) icon on the SQL Worksheet toolbar to execute the script. The code is displayed as follows:

```
SELECT *
FROM sales_info
  PIVOT (
    SUM(SALES)
    FOR PRODUCT_ID
    IN ('Prod_A', 'Prod_B', 'Prod_C') );
```

In the above query, the `PRODUCT_ID` column of the `sales_info` table is pivoted. This column is transposed as the column header.

The query retains the positions of the `CUSTOMER_ID` and `ORDER_DATE` columns, and formats the sales data in accordance with the pivoted column.

- Query the `sales_info` table to find the product sales for each customer in each month in the ascending order of the date of order. Compare your results with the following:



	ORDER_DATE	PRODUCT_ID	'Dan'	'James'	'Sam'
1	03-JAN-12	Prod_B			14
2	12-JAN-12	Prod_A	43		
3	04-FEB-12	Prod_A		86	
4	17-FEB-12	Prod_C			92
5	23-FEB-12	Prod_B	16		
6	09-MAR-12	Prod_c	36		
7	19-MAR-12	Prod_A			29
8	30-MAR-12	Prod_A		52	

Uncomment and select the code under Task 2 of case study. Click the Run Script (F5) icon on the SQL Worksheet toolbar to execute the script. The code is displayed as follows:

```
SELECT *
FROM sales_info
  PIVOT (
    SUM(SALES)
    FOR customer_id
    IN ('Dan', 'James', 'Sam') )
ORDER BY order_date;
```

Practices for Lesson 7: Pattern Matching using SQL

Chapter 7

Practices for Lesson 7: Overview

Overview

1. In this practice, you will query the `Ticker3Wave` table to find row patterns.
2. In the case study, you will search the `SALES_INFO` table for suspicious financial patterns during money transfer.

Practice 7-1: Pattern Matching using SQL

Overview

In this practice you will inquire

Assumptions

Task

In this practice, you perform pattern matching on the `Ticker3Wave` table.

1. In this step, you set up the data for this practice.
 - a. Run the `lab_07_01.sql` script using the HR Schema to set up the sample data for this practice. The script contains the following code:

```
CREATE TABLE Ticker3Wave (SYMBOL VARCHAR2(10), tstamp DATE,
PRICE NUMBER);

INSERT INTO Ticker3Wave VALUES('ACME', '01-Apr-11', 1000);
INSERT INTO Ticker3Wave VALUES('ACME', '02-Apr-11', 775);
INSERT INTO Ticker3Wave VALUES('ACME', '03-Apr-11', 900);
INSERT INTO Ticker3Wave VALUES('ACME', '04-Apr-11', 775);
INSERT INTO Ticker3Wave VALUES('ACME', '05-Apr-11', 900);
INSERT INTO Ticker3Wave VALUES('ACME', '06-Apr-11', 775);
INSERT INTO Ticker3Wave VALUES('ACME', '07-Apr-11', 900);
INSERT INTO Ticker3Wave VALUES('ACME', '08-Apr-11', 775);
INSERT INTO Ticker3Wave VALUES('ACME', '09-Apr-11', 800);
INSERT INTO Ticker3Wave VALUES('ACME', '10-Apr-11', 550);
INSERT INTO Ticker3Wave VALUES('ACME', '11-Apr-11', 900);
INSERT INTO Ticker3Wave VALUES('ACME', '12-Apr-11', 800);
INSERT INTO Ticker3Wave VALUES('ACME', '13-Apr-11', 1100);
INSERT INTO Ticker3Wave VALUES('ACME', '14-Apr-11', 800);
INSERT INTO Ticker3Wave VALUES('ACME', '15-Apr-11', 550);
INSERT INTO Ticker3Wave VALUES('ACME', '16-Apr-11', 800);
INSERT INTO Ticker3Wave VALUES('ACME', '17-Apr-11', 875);
INSERT INTO Ticker3Wave VALUES('ACME', '18-Apr-11', 950);
INSERT INTO Ticker3Wave VALUES('ACME', '19-Apr-11', 600);
INSERT INTO Ticker3Wave VALUES('ACME', '20-Apr-11', 300);
```

Connect to the HR schema using the HR Schema database connection. Open the `lab_07_01.sql` file from `/home/oracle/labs/labs` directory. Click the Run Script (F5) icon on the SQL Worksheet.

Enter the following code in the SQL Worksheet area:

Script Output x

Task completed in 0.003 seconds

```
COUNT(*)
-----
      20
```





[illegible]

```

SELECT *
FROM Ticker3Wave MATCH_RECOGNIZE (
    PARTITION BY symbol
    ORDER BY tstamp
    MEASURES B.tstamp AS timestamp,
             A.price AS Aprice,
             B.price AS Bprice,
             (B.price - A.price)*100) / A.price AS PctDrop
ONE ROW PER MATCH
AFTER MATCH SKIP TO B
PATTERN (A B)
DEFINE
    B AS (B.price - A.price) / A.price < -0.08
);

```





3. Write a query using `MATCH_RECOGNIZE` and `DEFINE` clauses to find V-patterns with one row per match in the `Ticker3Wave` table. Compare your results with the following:

	 SYMBOL	 START_TSTAMP	 BOTTOM_TSTAMP	 END_TSTAMP
1	ACME	01-APR-11	02-APR-11	03-APR-11
2	ACME	03-APR-11	04-APR-11	05-APR-11
3	ACME	05-APR-11	06-APR-11	07-APR-11
4	ACME	07-APR-11	08-APR-11	09-APR-11
5	ACME	09-APR-11	10-APR-11	11-APR-11
6	ACME	11-APR-11	12-APR-11	13-APR-11
7	ACME	13-APR-11	15-APR-11	18-APR-11

Uncomment and select the code under Task 3. Click the Run Statement (F9) icon on the SQL Worksheet toolbar to execute the script. The code is displayed as follows:

```
SELECT *
FROM Ticker3Wave MATCH_RECOGNIZE (
  PARTITION BY symbol
  ORDER BY tstamp
  MEASURES  STRT.tstamp AS start_tstamp,
            LAST(DOWN.tstamp) AS bottom_tstamp,
            LAST(UP.tstamp) AS end_tstamp
  ONE ROW PER MATCH
  AFTER MATCH SKIP TO LAST UP
  PATTERN (STRT DOWN+ UP+)
  DEFINE
    DOWN AS DOWN.price < PREV(DOWN.price),
    UP AS UP.price > PREV(UP.price)
) MR
ORDER BY MR.symbol, MR.start_tstamp;
```

4. Write a query to find w-patterns with all rows per match in the `Ticker3Wave` table. Compare your results with the following:

	 SYMBOL	 MATCH_NUM	 START_TSTAMP	 END_TSTAMP
1	ACME	1	01-APR-11	05-APR-11
2	ACME	2	05-APR-11	09-APR-11
3	ACME	3	09-APR-11	13-APR-11

Uncomment and select the code under Task 4. Click the Run Statement (F9) icon on the SQL Worksheet toolbar to execute the script. The code is displayed as follows:

```
SELECT *
FROM Ticker3Wave MATCH_RECOGNIZE (
  PARTITION BY symbol
  ORDER BY tstamp
  MEASURES
    MATCH_NUMBER() AS match_num,
    STRT.tstamp AS start_tstamp,
    FINAL LAST(UP.tstamp) AS end_tstamp
  ONE ROW PER MATCH
  AFTER MATCH SKIP TO LAST UP
  PATTERN (STRT DOWN+ UP+ DOWN+ UP+)
  DEFINE
    DOWN AS DOWN.price < PREV(DOWN.price),
    UP AS UP.price > PREV(UP.price)
) MR
ORDER BY MR.symbol, MR.start_tstamp;
```


Case Study

Scenario

The ITDB folks search for suspicious financial patterns during money transfer by creating applications specific to financial tracking. They detect suspicious money transfers by defining certain criteria as being unusual. For example, more than \$9,000,000 money transfer to a single recipient can be defined as an unusual criterion in financial tracking.

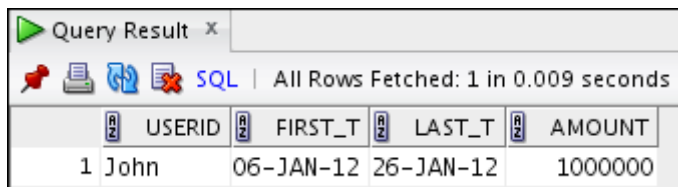
The task below would help the ITDB folks detect the following:

- A suspicious money transfer
- The recipient of the suspicious money transfer

Note: Execute `lab_07_02.sql` script from `/home/oracle/labs/labs` directory using the OE schema before performing the tasks below. The `lab_07_02.sql` script creates and populates values into the `EVENT_LOG` table.

Task

- Create a query to search for patterns that seem suspicious while transferring funds. Consider three or more small (less than \$2000) money transfers within 30 days followed by a large transfer (over \$1,000,000) within 10 days of the last small transfer as the suspicious search pattern for this query. Compare your result with the following:



	USERID	FIRST_T	LAST_T	AMOUNT
1	John	06-JAN-12	26-JAN-12	1000000

Connect to the OE Schema using the Choose Db Connection drop-down at the top-right corner of the SQL worksheet. Uncomment and select the code under Task 1 of case study. Click the Run Statement (F9) icon on the SQL Worksheet toolbar to execute the script. The code is displayed as follows:

```
SELECT userid, first_t, last_t, amount
FROM (SELECT * FROM event_log WHERE event = 'transfer')
MATCH_RECOGNIZE
  (PARTITION BY userid ORDER BY time
   MEASURES FIRST(x.time) first_t,
             y.time last_t, y.amount amount
   PATTERN ( x{3,} y )
   DEFINE
     x AS (event='transfer' AND amount < 2000), ← 1
     y AS (event='transfer' AND amount >= 1000000 AND ← 2
           LAST(x.time) - FIRST(x.time) < 30 AND ← 3
           y.time - LAST(x.time) < 10)); ← 4
```

The numbers 1, 2, 3, and 4 represent the following:

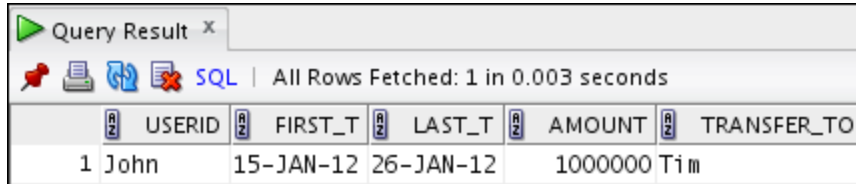
- 1: Represents all the small transfers
- 2: Represents a large transfer

- c) 3: Represents the small transfers occurred within 30 days
- d) 4: Represents the large transfer occurred within 10 days of the last small transfer.

Update the query from Task 1 to include the recipient of the suspicious transfer. Modify the variables *x*, *y* and *z* to represent the following:

- a) *x*: Two or more small transfers to different accounts
- b) *y*: The sum of all small transfers less than \$20,000
- c) *z*: The first small transfer

Compare your result with the following:



	USERID	FIRST_T	LAST_T	AMOUNT	TRANSFER_TO
1	John	15-JAN-12	26-JAN-12	1000000	Tim

Uncomment and select the code under Task 2 of case study. Click the Run Statement (F9) icon on the SQL Worksheet toolbar to execute the script. The code is displayed as follows:

```
SELECT userid, first_t, last_t, amount, transfer_to
FROM (SELECT * FROM event_log WHERE event = 'transfer')
MATCH_RECOGNIZE
  (PARTITION BY userid ORDER BY time
   MEASURES z.time first_t, y.time last_t, y.amount amount,
            y.transfer_to transfer_to
   PATTERN ( z x{2,} y )
   DEFINE z AS (event='transfer' AND amount < 2000),
          x AS (event='transfer' AND amount <= 2000 AND
                PREV(x.transfer_to) <> x.transfer_to),
          y AS (event='transfer' AND amount >= 1000000 AND
                LAST(x.time) - z.time < 30 AND
                y.time - LAST(x.time) < 10 AND
                SUM(x.amount) + z.amount < 20000));
```

Practices for Lesson 8: Modeling Data Using SQL

Chapter 8

Practices for Lesson 8: Overview

Overview

1. In this practice, you will use the SQL `MODEL` clause in the `SH` schema to perform interrow calculations.
2. In the case study, you will help the ITDB staff calculate the following:
 - a. The difference of sales in Italy and Spain for each product
 - b. Fourth quarter's sales for the company and for each country

Practice 8-1: Modeling Data by Using SQL

Overview

In this practice you will inquire

Assumptions

Task

In this practice, you use the SQL `MODEL` clause in the `SH` schema to perform interrow calculations.

1. In this step, you set up the data for this practice.
 - a. Run the `lab_08_01.sql` script to set up the sample data for this practice. The script contains the following code:

```
CREATE OR REPLACE VIEW sales_view AS
  SELECT country_name country, prod_name prod,
         calendar_year+6 year,
         SUM(amount_sold) sale, COUNT(amount_sold) cnt
  FROM   sh.sales, sh.times, sh.customers, sh.countries,
         sh.products
 WHERE  sales.time_id = times.time_id AND
        sales.prod_id = products.prod_id
 AND    sales.cust_id = customers.cust_id
 AND    customers.country_id = countries.country_id
 GROUP BY country_name, prod_name, calendar_year;
```

Connect to the `SH` schema by using the `SH` schema database connection. Open the `lab_08_01.sql` file from the `/home/oracle/labs/labs` directory. Click the Run Script (F5) icon on the SQL Worksheet.

- b. Verify that your `SALES_VIEW` has 3,219 rows.
Enter the following code in the SQL Worksheet area:

```
SELECT COUNT(*)
FROM sales_view ;
```

```
COUNT(*)
-----
3219

1 rows selected
```

2. As an initial example of models, examine the following statement:

```
SELECT SUBSTR(country,1,20) country,
       SUBSTR(prod,1,15) prod, year, sales
```

```

FROM sales_view
WHERE country IN ('Japan','Singapore')
MODEL RETURN UPDATED ROWS
  PARTITION BY (country)
  DIMENSION BY (prod, year)
  MEASURES (sale sales)
  RULES (
    sales['Bounce', 2007] = sales['Bounce', 2006] +
                          sales['Bounce', 2005],
    sales['Y Box', 2007] = sales['Y Box', 2006],
    sales['2_Products', 2007] = sales['Bounce', 2007] +
    sales['Y Box', 2007])
ORDER BY country, prod, year;

```

The above statement partitions by country, so the rules are applied to the data of one country at a time. Note that the data ends with 2007, so any rules defining values for 2008 or later will insert new cells. The first rule defines the sales of Bounce in 2007 as the sum of sales in 2006 and 2005. The second rule defines the sales for Y Box in 2007 as being the same value as they were for 2006. The third rule defines a category called 2_Products, which is the sum of adding the 2007 Bounce and Y Box values. Note that the values for 2_Products are derived from the results of the two former rules, so those rules must be executed before the 2_Products rule. Execute the statement, or you can run the lab_08_02.sql script file. Examine the result set.

Open the lab_08_02.sql file from the /home/oracle/labs/labs directory. Click the Run Script (F5) icon on the SQL Worksheet. Examine the following result set:

COUNTRY	PROD	YEAR	SALES
Japan	2_Products	2007	53785.49
Japan	Bounce	2007	8094.83
Japan	Y Box	2007	45690.66
Singapore	2_Products	2007	17222.76
Singapore	Bounce	2007	3066.51
Singapore	Y Box	2007	14156.25
6 rows selected			

3. You want to view the `SALES` value for the product `Bounce` in the year 2004, in Canada, and set it to 10. To accomplish this, use a “positional cell reference.” The value for the cell reference is matched to the appropriate dimension based on its position in the expression. The `DIMENSION BY` clause of the model determines the position assigned to each dimension—in this case, the first position is product (`PROD`) and the second position is `YEAR`. Your result set should be as follows:

COUNTRY	PROD	YEAR	SALES
Canada	Bounce	2004	10
1 rows selected			

Open the `/home/oracle/labs/solns/sol_08.sql` script. Uncomment and select the code under Task 3. Click the Run Script (F5) icon on the SQL Worksheet toolbar to execute the script. The code is displayed as follows:

```
SELECT SUBSTR(country,1,20) country,
       SUBSTR(prod,1,15) prod, year, sales
FROM   sales_view
WHERE  country='Canada'
MODEL  RETURN UPDATED ROWS
      PARTITION BY (country)
      DIMENSION BY (prod, year)
      MEASURES (sale sales)
      RULES (
        sales['Bounce', 2004] = 10 )
ORDER BY country, prod, year;
```

4. You want to create a forecast value of `SALES` for the product `Bounce` in the year 2008, in Spain, and set it to 15. Use a rule in the `SELECT` statement that sets the year value to 2008 and thus creates a new cell in the array.

Your result set should be as follows:

COUNTRY	PROD	YEAR	SALES
Spain	Bounce	2008	15
1 rows selected			

Uncomment and select the code under Task 4. Click the Run Script (F5) icon on the SQL Worksheet toolbar to execute the script. The code is displayed as follows:

```
SELECT SUBSTR(country,1,20) country,
       SUBSTR(prod,1,15) prod, year, sales
FROM   sales_view
WHERE  country = 'Spain'
MODEL  RETURN UPDATED ROWS
      PARTITION BY (country)
      DIMENSION BY (prod, year)
      MEASURES (sale sales)
      RULES (
        sales['Bounce', 2008] = 15 )
ORDER BY country, prod, year;
```

If you have time, complete the following steps:

5. You want to forecast the sales of Bounce in France for the year 2008 to be 100 more than the maximum sales in the period 2002 to 2005. To do so, you need to use the **BETWEEN** clause to specify multiple cells on the right of the rule, and these are aggregated to a single value with the **MAX()** function.

Your result set should be as follows:

COUNTRY	PROD	YEAR	SALES
France	Bounce	2008	2082.11
1 rows selected			

Uncomment and select the code under Task 5. Click the Run Script (F5) icon on the SQL Worksheet toolbar to execute the script. The code is displayed as follows:

```
SELECT SUBSTR(country,1,20) country,
       SUBSTR(prod,1,15) prod, year, sales
FROM   sales_view
WHERE  country='France'
MODEL  RETURN UPDATED ROWS
      PARTITION BY (country)
      DIMENSION BY (prod, year)
      MEASURES (sale sales)
      RULES (
        sales['Bounce', 2008] =
        100 + max(sales)['Bounce', year BETWEEN 2002 AND 2005] )
ORDER BY country, prod, year;
```


6. You want to update the sales values for Bounce in Germany for multiple years, using a rule where each year's sales is the sum of Mouse Pad sales for that year, plus 20 percent of the Y Box sales for that year. For the baseline year, use 2004–2007. Use the `CV()` function to help obtain the results. By subtracting from the `CV(year)` value, you can refer to other rows in the data set.

Your result set should be as follows:

COUNTRY	PROD	YEAR	SALES
Germany	Bounce	2005	13636.056
Germany	Bounce	2006	19051.414
Germany	Bounce	2007	32123.778
3 rows selected			

Uncomment and select the code under Task 6. Click the Run Script (F5) icon on the SQL Worksheet toolbar to execute the script. The code is displayed as follows:

```
SELECT SUBSTR(country,1,20) country,
       SUBSTR(prod,1,15) prod, year, sales
FROM sales_view
WHERE country='Germany'
MODEL RETURN UPDATED ROWS
  PARTITION BY (country)
  DIMENSION BY (prod, year)
  MEASURES (sale sales)
  RULES (
    sales['Bounce', year BETWEEN 2004 AND 2007] =
    sales['Mouse Pad', cv(year)] +
    0.2 * sales['Y Box', cv(year)])
ORDER BY country, prod, year;
```

7. You want to specify projection for the United Kingdom sales values for mouse pads for the years 2008 to 2012 so that they are equal to 120 percent of the value in 2006.

Your result set should be as follows:

COUNTRY	PROD	YEAR	SALES
United Kingdom	Mouse Pad	2008	7860.456
United Kingdom	Mouse Pad	2009	7860.456
United Kingdom	Mouse Pad	2010	7860.456
United Kingdom	Mouse Pad	2011	7860.456
United Kingdom	Mouse Pad	2012	7860.456
5 rows selected			

Uncomment and select the code under Task 7. Click the Run Script (F5) icon on the SQL Worksheet toolbar to execute the script. The code is displayed as follows:

```
SELECT SUBSTR(country,1,20) country,
       SUBSTR(prod,1,15) prod, year, sales
FROM sales_view
WHERE country='United Kingdom'
MODEL RETURN UPDATED ROWS
  PARTITION BY (country)
  DIMENSION BY (prod, year)
  MEASURES (sale sales)
  RULES (
    sales['Mouse Pad', FOR year FROM 2008 TO 2012 INCREMENT 1]
    = 1.2 * sales[cv(prod), 2005] )
ORDER BY country, prod, year;
```

8. Examine this query and its results. Execute the lab_08_08.sql script.

```
SELECT SUBSTR(country,1,20) country, SUBSTR(prod,1,15)
       prod, year, sales
FROM sales_view
WHERE country='Italy'
MODEL      RETURN UPDATED ROWS
  PARTITION BY (country)
  DIMENSION BY (prod, year)
  MEASURES (sale sales)
  RULES (
    sales['Mouse Pad', 2009] =
    sales['Mouse Pad', 2007] + sales['Mouse Pad', 2008])
ORDER BY country, prod, year;
```

COUNTRY	PROD	YEAR	SALES
Italy	Mouse Pad	2009	
1 rows selected			

Because NULL values cause many rules to return nulls, it may be more useful for you to treat nulls and missing values as 0 values. Convert the query shown above to return a numeric value for sales even though the value for 2008 is missing. Your result set should be as follows:

COUNTRY	PROD	YEAR	SALES
Italy	Mouse Pad	2009	4747.9
1 rows selected			

Uncomment and select the code under Task 8. Click the Run Script (F5) icon on the SQL Worksheet toolbar to execute the script. The code is displayed as follows:

```
SELECT SUBSTR(country,1,20) country,  
       SUBSTR(prod,1,15) prod, year, sales  
FROM   sales_view  
WHERE  country='Italy'  
MODEL  IGNORE NAV RETURN UPDATED ROWS  
  PARTITION BY (country)  
  DIMENSION BY (prod, year)  
  MEASURES (sale sales)  
  RULES  (  
    sales['Mouse Pad', 2009] =  
    sales['Mouse Pad', 2007] + sales['Mouse Pad', 2008])  
ORDER BY country, prod, year;
```

Case Study

Scenario

The ITDB staff use SQL modeling to perform tasks such as calculating the sales difference between two or more products, calculating the net present value (NPV) of a series of periodic cash flows, and so on.

The task below would help the ITDB staff calculate the following:

- The difference of sale in Italy and Spain for each product
- Fourth quarter's sales for the company and for each country

Task

- Connect to the SH schema and create a new view named `sales_view2` to find monthly totals of sales and quantities by product and country. Use a subquery to insert the rows from the `sales`, `times`, `customers`, `countries` and `products` tables into the newly created view. Display all rows from the view. Compare your results with the following:

	COUNTRY	PRODUCT	YEAR	MONTH	SALE	CNT
1	Germany	5MP Telephoto Digital Camera	1998	January	13553.76	11
2	Spain	5MP Telephoto Digital Camera	1998	February	2470.3	2
3	United Kingdom	5MP Telephoto Digital Camera	1998	February	7406.58	6
4	Denmark	5MP Telephoto Digital Camera	1998	March	2465.98	2
5	United States of America	5MP Telephoto Digital Camera	1998	March	28358.77	23
6	Australia	17" LCD w/built-in HDTV Tuner	1998	January	13289.06	11
7	United States of America	Envoy 256MB - 40GB	1998	January	70373.3	70
8	Australia	Envoy 256MB - 40GB	1998	February	8020.92	8
9	Germany	Mini DV Camcorder with 3.5" Swivel LCD	1998	January	20085.45	13
10	Spain	Mini DV Camcorder with 3.5" Swivel LCD	1998	January	6209.39	4
11	United Kingdom	Mini DV Camcorder with 3.5" Swivel LCD	1998	January	20145.15	13

...

Connect to the SH schema by using the SH schema database connection. Open the `/home/oracle/labs/solns/sol_05.sql` script. Uncomment and select the code under Task 1 of the case study. Click the Run Statement (F9) icon on the SQL Worksheet toolbar to execute the script. The code is displayed as follows:

```
DROP VIEW sales_view2;

CREATE VIEW sales_view2 AS
SELECT country_name country, prod_name product, calendar_year year,
       calendar_month_name month, SUM(amount_sold) sale,
       COUNT(amount_sold) cnt
FROM sales, times, customers, countries, products
WHERE sales.time_id = times.time_id AND
       sales.prod_id = products.prod_id AND
       sales.cust_id = customers.cust_id AND
       customers.country_id = countries.country_id
```

```
GROUP BY country_name, prod_name, calendar_year,
calendar_month_name;

Select * from sales_view2;
```

2. Create a query by using the MODEL clause to show the sales for Italy and Spain and the difference in the sales for each product in the two countries. The difference should be placed in a new row with country = 'Diff Italy-Spain'. Compare your results with the following:

Query Result x

SQL | Fetched 100 rows in 1.838 seconds

R2	PRODUCT	R2	COUNTRY	R2	SALES
1	Model K8822S Cordless Phone Battery		Spain		11385.79
2	Smash up Boxing		Italy		10501.36
3	Comic Book Heroes		Italy		3434.92
4	Model K8822S Cordless Phone Battery		Italy		29385.38
5	Model SM26273 Black Ink Cartridge		Spain		10578.36
6	Envoy External 6X CD-ROM		Italy		26001.09
7	Smash up Boxing		Spain		5121.03
8	Bounce		Italy		11654.77
9	Envoy External 6X CD-ROM		Spain		10807.09
10	Finding Fido		Italy		3260.69
11	Comic Book Heroes		Spain		1894.88
12	Model SM26273 Black Ink Cartridge		Italy		32443.73
13	Bounce		Spain		4521.9
14	Finding Fido		Spain		1446.69
15	Envoy External 6X CD-ROM		DIFF ITALY-SPAIN		15194
16	Model SM26273 Black Ink Cartridge		DIFF ITALY-SPAIN		21865.37
17	Comic Book Heroes		DIFF ITALY-SPAIN		1540.04
18	Smash up Boxing		DIFF ITALY-SPAIN		5380.33
19	Model K8822S Cordless Phone Battery		DIFF ITALY-SPAIN		17999.59
20	Bounce		DIFF ITALY-SPAIN		7132.87
21	Finding Fido		DIFF ITALY-SPAIN		1814
22	CD-R with Jewel Cases, pACK OF 12		Spain		2746.26
23	S27273M Extended Use w/1 Phone Batt.		Spain		10739.9

...

Uncomment and select the code under Task 2 of the case study. Click the Run Statement (F9) icon on the SQL Worksheet toolbar to execute the script. The code is displayed as follows:

```
SELECT product, country, sales
FROM sales_view2
WHERE country IN ('Italy', 'Spain')
```

Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

```

GROUP BY product, country
MODEL
  PARTITION BY (product) DIMENSION BY (country) MEASURES (SUM(sale)
AS sales)
  RULES UPSERT
    (sales['DIFF ITALY-SPAIN'] = sales['Italy'] - sales['Spain']);

```

3. Consider a scenario where the sales for each product in each country grew or declined at the same monthly rate from November 2000 to December 2000 as they did from October 2000 to November 2000. Create a query to calculate the fourth quarter's sales for the company and for an individual country. Compare your result with the following:

Query Result x	
SQL All Rows Fetched: 13 in 0.366 seconds	
COUNTRY	SUM(SALES)
1 Australia	250002.6264434803083381722215327238994274
2 Brazil	551.63
3 Canada	216722.5354572873762047903897394858545358
4 Denmark	118424.1049578372857284030120408203847443
5 France	279346.8885297643485291251215415756814879
6 Germany	571828.9165930267761686378732584917577701
7 Italy	264801.930264380877493880740375931817401
8 Japan	464470.2374536287887634137607851367676686
9 Singapore	361310.0712421328933954956179838843739929
10 Spain	196005.9101636227418939645537893934195918
11 United Kingdom	459559.6019187516464679387065632078659175
12 United States of America	3500592.06804498657014979165985938479813
13	6683616.52106889961313361365747003662067

Uncomment and select the code under Task 3 of the case study. Click the Run Statement (F9) icon on the SQL Worksheet toolbar to execute the script. The code is displayed as follows:

```

SELECT country, SUM(sales)
FROM (SELECT product, country, month, sales
      FROM sales_view2
      WHERE year=2000 AND month IN ('October','November'))
MODEL
  PARTITION BY (product, country) DIMENSION BY (month) MEASURES
(sale sales)
  RULES
    (sales['December']=(sales['November'] /sales['October'])
*sales['November']))
GROUP BY GROUPING SETS (( ), (country));

```