ORACLE UNIVERSITY

Integrated Cloud Applications & Platform Services

Oracle Database 12c R2: SQL Workshop II

Activity Guide

D80194GC20

Edition 2.0 | November 2016 | D98639

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

ORACLE

Author

Apoorva Srinivas

Technical Contributors and Reviewers

Nancy Greenberg

Suresh Rajan

Bryan Roberts

Sharath Bhujani

Graphic Designer

Kavya Bellur

Publishers

Veena Narasimhan

Asief Baig

Table of Contents

Course Practice Environment: Security Credentials

Chapter I

Course Practice Environment: Security Credentials

For OS usernames and passwords, see the following:

- If you are attending a classroom-based or live virtual class, ask your instructor or LVC producer for OS credential information.
- If you are using a self-study format, refer to the communication that you received from Oracle University for this course.

For connection-specific credentials used in this course, see the following table:

Connection-Specific Credentials		
Connection_Name	**Username**	**Password**
myconnection	`ora21`	`ora21`
`ora22`	`ora22`	`ora22`

Practices for Lesson 1: Introduction

Chapter 1

Practices for Lesson 1: Overview

Practice Overview

In this practice, you 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 a SQL script by opening the file in SQL Developer.

Practice 1-1: Using SQL Developer

Tasks

1. Start SQL Developer by using the desktop icon.
2. Create a database connection by using the following information:
   - Connection Name: `myconnection`
   - Username: `ora21`
   - Password: Enter the password from the “Course Practice Environment: Security Credentials” document.
   - Hostname: `localhost`
   - Port: `1521`
   - Service Name: `PDBORCL`
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 in 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 Run 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 the last name, job ID, and commission for all employees who are entitled to receive a commission.
6. Set your script pathing preference to `/home/oracle/labs/sql2`.
   a. Select Tools > Preferences > Database > Worksheet.
   b. Enter the value in the “Select default path to look for scripts” field.
7. Enter the following in the Enter SQL Statement box:

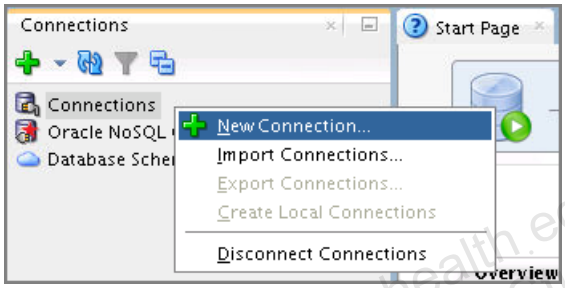
```
SELECT employee_id, first_name, last_name
       FROM   employees;
```

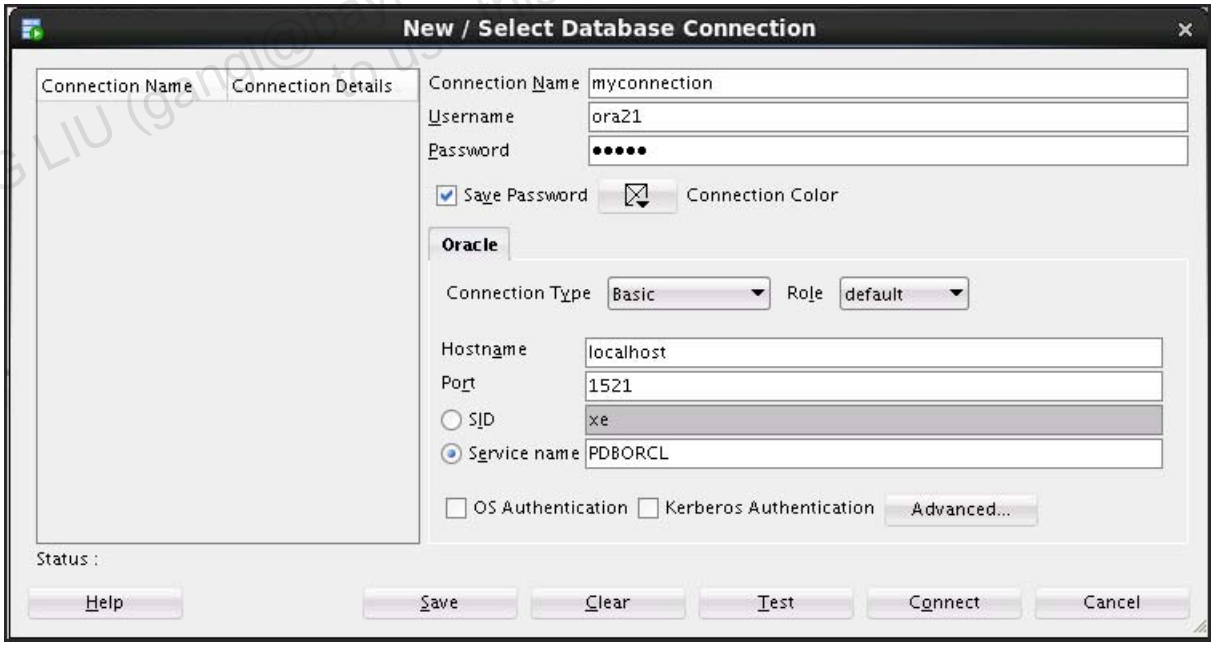
8. Save the SQL statement to a script file by using the File > Save menu item.
   a. Select File > Save.
   b. Name the file `intro_test.sql`.
   c. Place the file in your `/home/oracle/labs/sql2/labs` folder.
9. Open and run `confidence.sql` from your `/home/oracle/labs/sql2/labs` folder, and observe the output.

Solution 1-1: Using SQL Developer

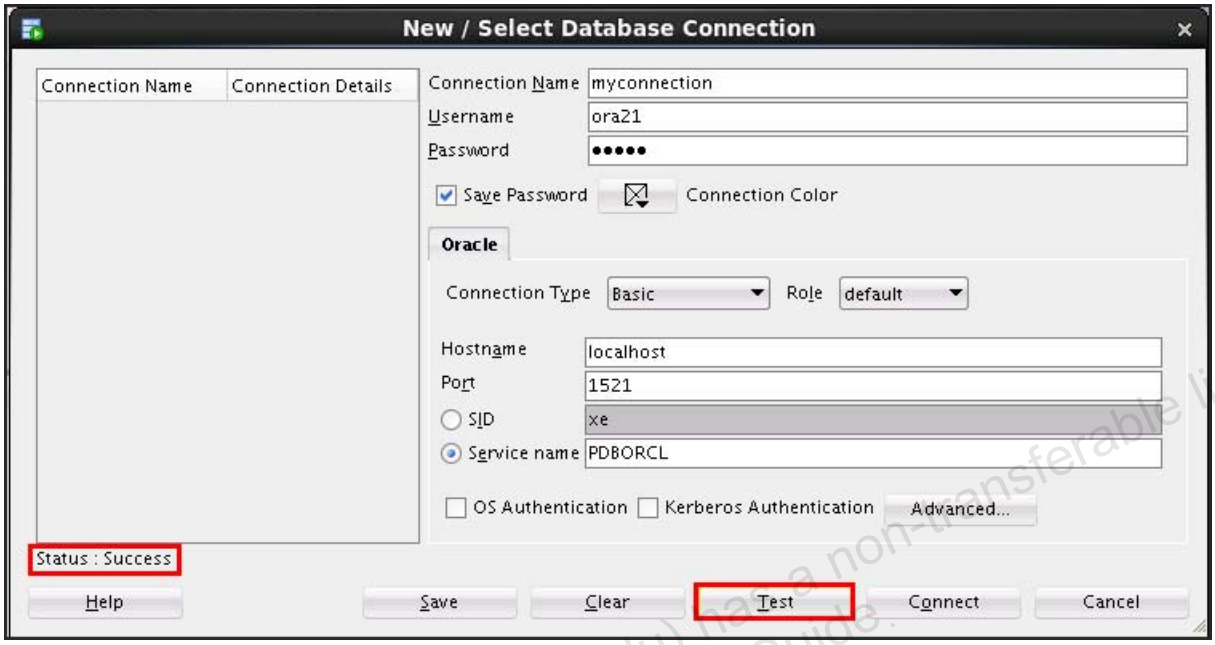
1. Start SQL Developer by using the desktop icon.

2. Create a database connection by using the following information:
   - Connection Name: `myconnection`
   - Username: `ora21`
   - Password: Enter the password from the “Course Practice Environment: Security Credentials” document.
   - Hostname: `localhost`
   - Port: `1521`
   - Service Name: `PDBORCL`

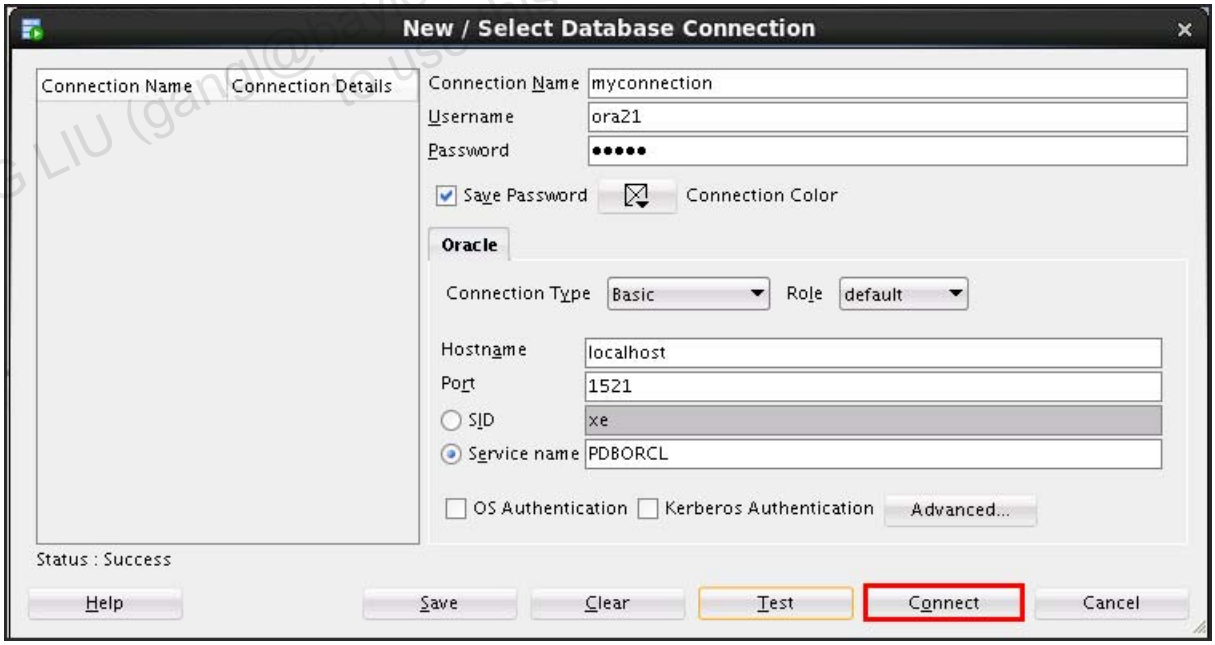




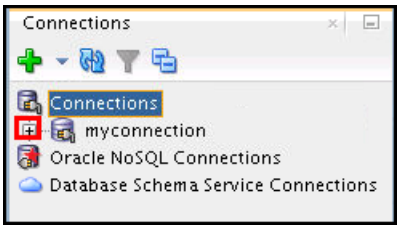
3. Test the new connection. If the status is Success, connect to the database by using this new connection.
   a. Click the Test button in the New/Select Database Connection window.



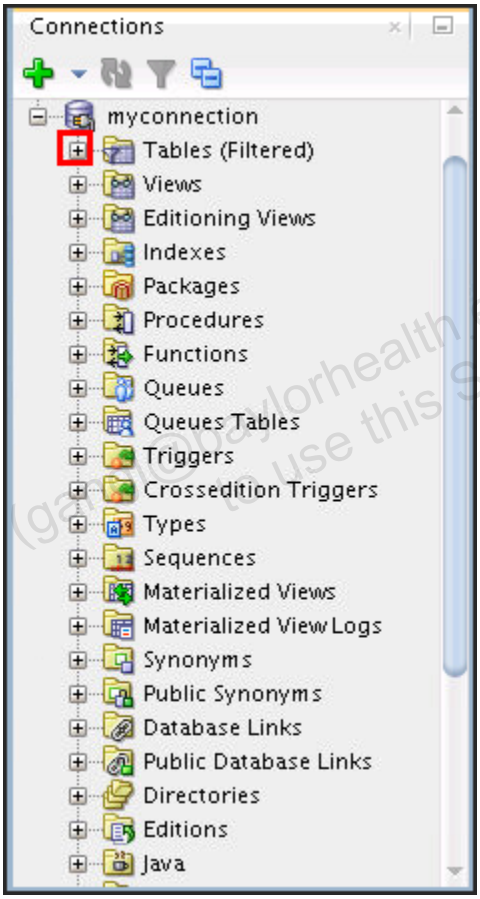
   b. If the status is Success, click the Connect button.

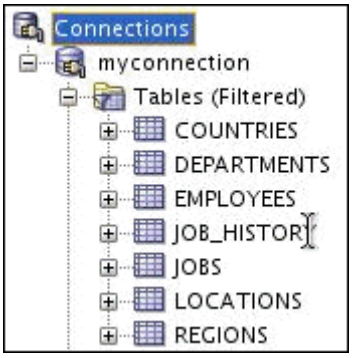


4. Browse the structure of the `EMPLOYEES` table and display its data.
   a. Expand the myconnection connection by clicking the plus sign next to it.



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





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

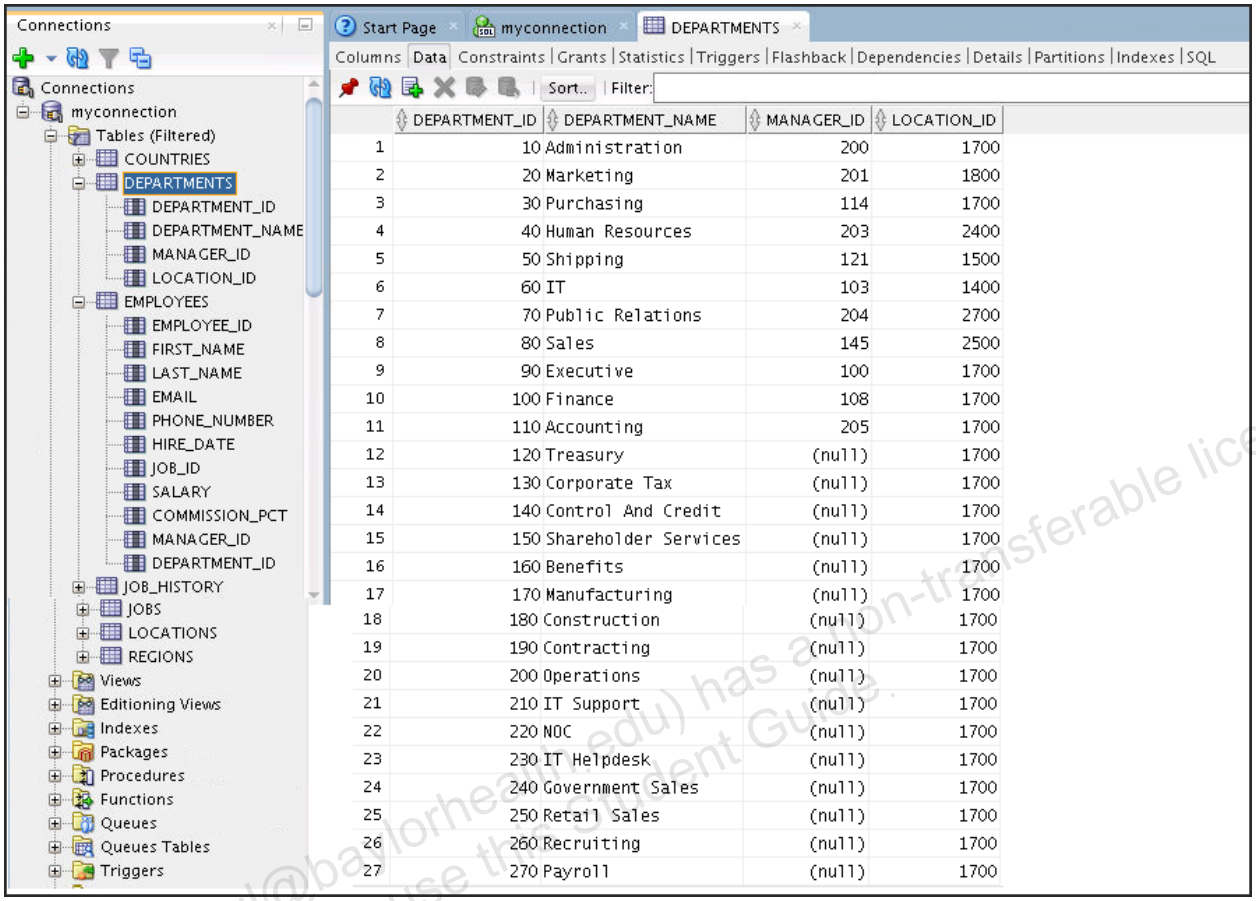
Columns | Data | Model | Constraints | Grants | Statistics | Triggers | Flashback | Dependencies | Details | Partitions | Indexes | SQL

Actions...

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

d. View the data in the `DEPARTMENTS` table.

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



	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
9	90	Executive	100	1700
10	100	Finance	108	1700
11	110	Accounting	205	1700
12	120	Treasury	(null)	1700
13	130	Corporate Tax	(null)	1700
14	140	Control And Credit	(null)	1700
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

5. Execute some basic `SELECT` statements to query the data in the `EMPLOYEES` table in the SQL Worksheet area. Use both the Run 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;
```

	LAST_NAME	SALARY
1	Baida	2900
2	Tobias	2800
3	Himuro	2600
4	Colmenares	2500
5	Mikkilineni	2700
6	Landry	2400
7	Markle	2200
8	Atkinson	2800
9	Marlow	2500
10	Olson	2100
11	Rogers	2900
12	Gee	2400

...

b. Write a query to display the last name, job `ID`, and commission for all employees who are entitled to receive a commission.

```
SELECT last_name, job_id, commission_pct
FROM   employees
WHERE  commission_pct IS NOT NULL;
```

	LAST_NAME	JOB_ID	COMMISSION_PCT
1	Russell	SA_MAN	0.4
2	Partners	SA_MAN	0.3
3	Errazuriz	SA_MAN	0.3
4	Cambrault	SA_MAN	0.3
5	Zlotkey	SA_MAN	0.2
6	Tucker	SA_REP	0.3
7	Bernstein	SA_REP	0.25
8	Hall	SA_REP	0.25
9	Olsen	SA_REP	0.2
10	Cambrault	SA_REP	0.2

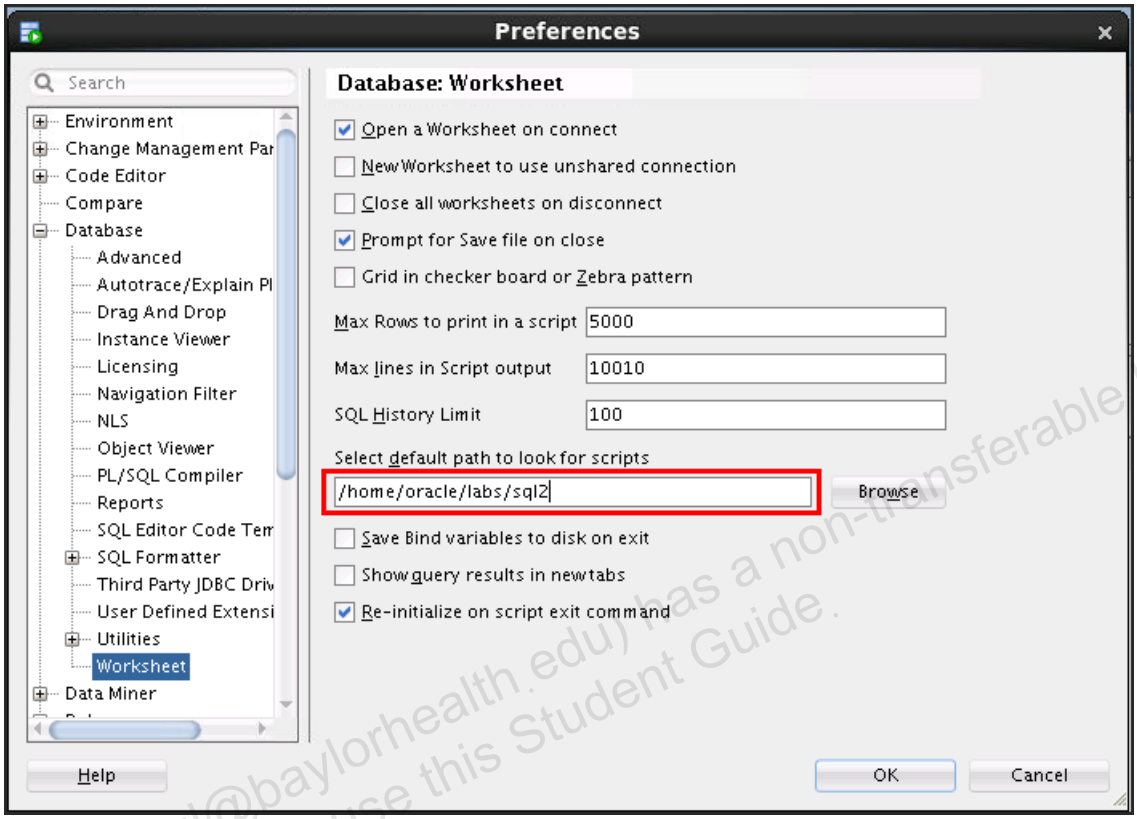
...

6. Set your script pathing preference to `/home/oracle/labs/sql2`.
   a. Select Tools > Preferences > Database > Worksheet.

b. Enter the value in the “Select default path to look for scripts” field. Then click OK.

Note: To view the number of rows selected, enable the feedback option and set it to `1`.

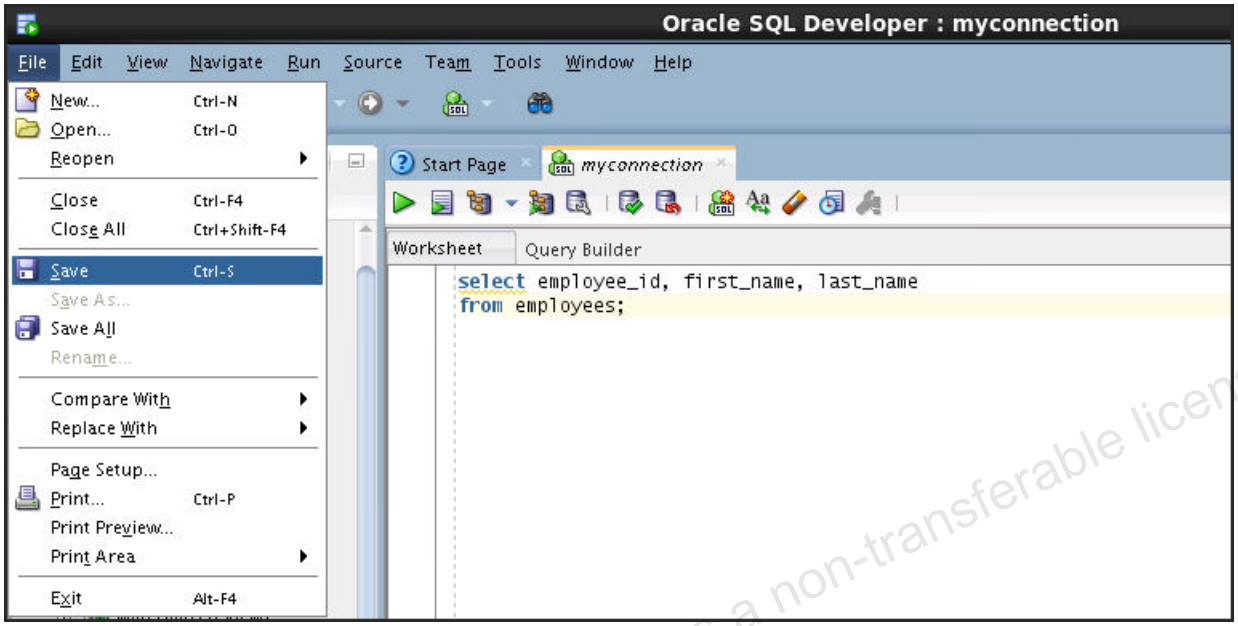
```
set feedback on;
set feedback 1;
```



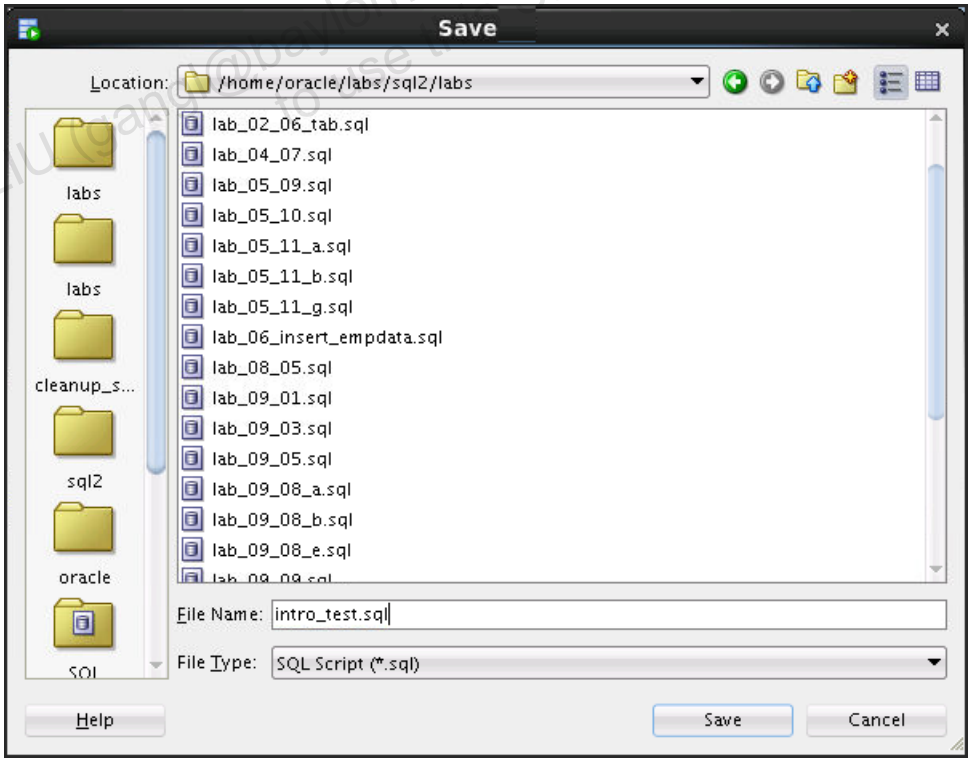
7. Enter the following SQL statement:

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

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



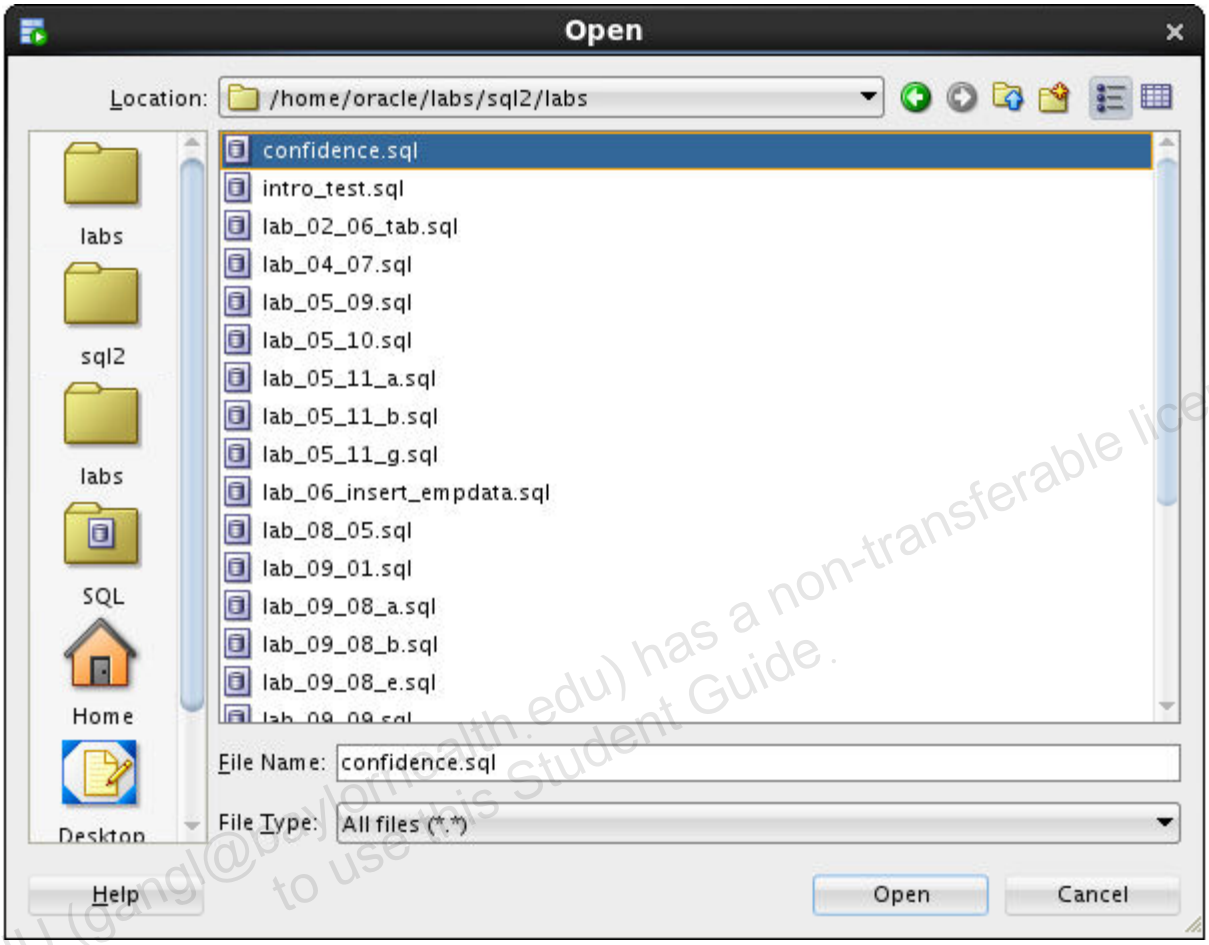
   b. Name the file **`intro_test.sql`**.
      Enter `intro_test.sql` in the File_name text box.
   c. Place the file in the **`/home/oracle/labs/sql2/labs`** folder.



Then click Save.

9. Open and run `confidence.sql` from your `/home/oracle/labs/sql2/labs` folder and observe the output.

   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

COUNT(*)
--------
     107

COUNT(*)
--------
      25

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

```
COUNT(*)
--------
      23

COUNT(*)
--------
      27

COUNT(*)
--------
      19

COUNT(*)
--------
      10
```

Practices for Lesson 2: Introduction to Data Dictionary Views

Chapter 2

Practices for Lesson 2: Overview

Practice overview

This practice covers the following topics:

- Querying the dictionary views for table and column information
- Querying the dictionary views for constraint information
- Adding a comment to a table and querying the dictionary views for comment information

Practice 2-1: Introduction to Data Dictionary Views

Overview

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

Tasks

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

...

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.

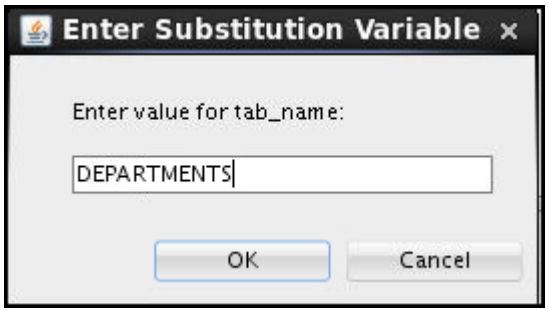
   Note: Your list may not exactly match the following list:

	TABLE_NAME	OWNER
1	DUAL	SYS
2	SYSTEM_PRIVILEGE_MAP	SYS
3	TABLE_PRIVILEGE_MAP	SYS
4	USER_PRIVILEGE_MAP	SYS
5	STMT_AUDIT_OPTION_MAP	SYS
6	AUDIT_ACTIONS	SYS
7	WRR$_REPLAY_CALL_FILTER	SYS
8	HS_BULKLOAD_VIEW_OBJ	SYS
9	HS$_PARALLEL_METADATA	SYS
10	HS_PARTITION_COL_NAME	SYS
11	HS_PARTITION_COL_TYPE	SYS

…

98	SDO_TOPO_DATA$	MDSYS
99	SDO_GR_MOSAIC_0	MDSYS
100	SDO_GR_MOSAIC_1	MDSYS
101	SDO_GR_MOSAIC_2	MDSYS
102	SDO_GR_MOSAIC_3	MDSYS
103	SDO_GR_PARALLEL	MDSYS
104	SDO_GR_RDT_1	MDSYS
105	SDO_WFS_LOCAL_TXNS	MDSYS

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_02_03.sql`.
For example, if the user enters `DEPARTMENTS`, the following output results:



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

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_02_04.sql`.
For example, if the user enters `DEPARTMENTS`, the following output results:

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

5. 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.

6. Run the `lab_02_06_tab.sql` script as a prerequisite for exercises 6 through 9. Alternatively, open the script file to copy the code and paste it into your SQL Worksheet. Then execute the script. This script:
   - Drops the existing `DEPT2` and `EMP2` tables
   - Creates the `DEPT2` and `EMP2` tables

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

	TABLE_NAME
1	DEPT2
2	EMP2

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

	CONSTRAINT_NAME	CONSTRAINT_TYPE
1	MY_EMP_DEPT_ID_FK	R
2	MY_DEPT_ID_PK	P
3	MY_EMP_ID_PK	P

9. Display the object names and types from the `USER_OBJECTS` data dictionary view for the `EMP2` and `DEPT2` tables.

	OBJECT_NAME	OBJECT_TYPE
1	DEPT2	TABLE
2	EMP2	TABLE

Solution 2-1: Introduction to Data Dictionary Views

Solution

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

```
SELECT table_name
   FROM   user_tables;
```

2. Query the `ALL_TABLES` data 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 <>'ORA21';
```

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_02_03.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_02_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.

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. Run the lab_02_06_tab.sql script as a prerequisite for exercises 6 through 9. 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
7. 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');
```

8. 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');
```

9. Display the object names and types from the USER_OBJECTS data dictionary view for the EMP2 and DEPT2 tables.

```
SELECT   object_name, object_type
FROM     user_objects
WHERE    object_name= 'EMP2'
OR       object_name= 'DEPT2';
```

Practices for Lesson 3: Creating Sequences, Synonyms, and Indexes

Chapter 3

Practices for Lesson 3: Overview

Practices Overview

This practice covers the following topics:

- Creating sequences
- Using sequences
- Querying the dictionary views for sequence information
- Creating synonyms
- Querying the dictionary views for synonyms information
- Creating indexes
- Querying the dictionary views for indexes information

Note: Before starting this practice, execute the `/home/oracle/labs/sql2/code_ex/cleanup_scripts/cleanup_03.sql` script.

Practice 3-1: Creating Sequences, Synonyms, and Indexes

Overview

This practice provides you with a variety of exercises in creating and using a sequence, an index, and a synonym.

Note: Execute the `cleanup_03.sql` script from `/home/oracle/labs/sql2/code_ex/cleanup_scripts/` before performing the following tasks.

Tasks

1. Create the `DEPT` table based on the following table instance chart. Confirm that the table is created.

Column Name	`ID`	`NAME`
Key Type	`Primary key`	
Null/Unique		
FK Table		
FK Column		
Data Type	`NUMBER`	`VARCHAR2`
Length	`7`	`25`

2. You need a sequence that can be used with the `PRIMARY KEY` column of the `DEPT` table. The sequence should start at 200 and have a maximum value of 1,000. Have your sequence increment by 10. Name the sequence `DEPT_ID_SEQ`.
3. To test your sequence, write a script to insert two rows in the `DEPT` table. Name your script `lab_03_03.sql`. Be sure to use the sequence that you created for the `ID` column. Add two departments: Education and Administration. Confirm your additions. Run the commands in your script.
4. 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_04.sql`. Run the statement in your script.

	SEQUENCE_NAME	MAX_VALUE	INCREMENT_BY	LAST_NUMBER
1	DEPARTMENTS_SEQ	9990	10	280
2	DEPT_ID_SEQ	1000	10	400
3	EMPLOYEES_SEQ	9999999999999999999999999999	1	207
4	LOCATIONS_SEQ	9900	100	3300

5. Create a synonym for your `EMPLOYEES` table. Call it `EMP1`. Then find the names of all synonyms that are in your schema.

	SYNONYM_NAME	TABLE_OWNER	TABLE_NAME	DB_LINK	ORIGIN_CON_ID
1	EMP1	ORA21	EMPLOYEES	(null)	3

6. Drop the `EMP1` synonym.
7. Create a nonunique index on the `NAME` column in the `DEPT` table.

8. 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.

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

9. Drop the tables and sequences created in this practice.

Solution 3-1: Creating Sequences, Synonyms, and Indexes

1. Create the DEPT table based on the following table instance chart. Confirm that the table is created.

Column Name	ID	NAME
Key Type	Primary key	
Null/Unique		
FK Table		
FK Column		
Data Type	NUMBER	VARCHAR2
Length	7	25

```
CREATE TABLE dept
 (id   NUMBER(7)CONSTRAINT department_id_pk PRIMARY KEY,
  name VARCHAR2(25));
```

To confirm that the table was created and to view its structure, issue the following command:

```
DESCRIBE dept;
```

2. You need a sequence that can be used with the primary key column of the DEPT table. The sequence should start at 200 and have a maximum value of 1,000. Have your sequence increment by 10. Name the sequence DEPT_ID_SEQ.

```
CREATE SEQUENCE dept_id_seq
  START WITH 200
  INCREMENT BY 10
  MAXVALUE 1000;
```

3. To test your sequence, write a script to insert two rows in the DEPT table. Name your script lab_03_03.sql. Be sure to use the sequence that you created for the ID column. Add two departments: Education and Administration. Confirm your additions. Run the commands in your script.

```
INSERT INTO dept
VALUES (dept_id_seq.nextval, 'Education');
INSERT INTO dept
VALUES (dept_id_seq.nextval, 'Administration');
```

4. 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_04.sql`. Run the statement in your script.

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

5. Create a synonym for your `EMPLOYEES` table. Call it `EMP1`. Then find the names of all synonyms that are in your schema.

```
CREATE SYNONYM emp1 FOR EMPLOYEES;
SELECT *
FROM   user_synonyms;
```

6. Drop the `EMP1` synonym.

```
DROP SYNONYM emp1;
```

7. Create a nonunique index on the `NAME` column in the `DEPT` table.

```
CREATE INDEX dept_name_idx ON dept (name);
```

8. 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.

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';
```

9. Drop the tables and sequences created in this practice.

```
DROP TABLE DEPT;
DROP TABLE SALES_DEPT;
DROP SEQUENCE dept_id_seq;
```

Practices for Lesson 4: Creating Views

Chapter 4

Practices for Lesson 4: Overview

Practices Overview

This practice covers the following topics:

- Creating a simple view
- Creating a complex view
- Creating a view with a check constraint
- Attempting to modify data in the view
- Querying the dictionary views for view information
- Removing views

Practice 4-1: Creating Views

Overview:

In this practice, you create and use views, query data dictionary views for view information, and remove views.

Tasks:

1. The staff in the `HR` department wants to hide some of the data in the `EMPLOYEES` table. Create a view called `EMPLOYEES_VU` based on the employee numbers, employee last names, and department numbers from the `EMPLOYEES` table. The heading for the employee name should be `EMPLOYEE`.
2. Confirm that the view works. Display the contents of the `EMPLOYEES_VU` view.

	EMPLOYEE_ID	EMPLOYEE	DEPARTMENT_ID
1	100	King	90
2	101	Kochhar	90
3	102	De Haan	90
4	103	Hunold	60
5	104	Ernst	60
6	105	Austin	60
7	106	Pataballa	60
8	107	Lorentz	60
9	108	Greenberg	100
10	109	Faviet	100
11	110	Chen	100
12	111	Sciarra	100
13	112	Urman	100

...

3. Using your `EMPLOYEES_VU` view, write a query for the `HR` department to display all employee names and department numbers.

	EMPLOYEE	DEPARTMENT_ID
1	King	90
2	Kochhar	90
3	De Haan	90
4	Hunold	60
5	Ernst	60
6	Austin	60
7	Pataballa	60
8	Lorentz	60
9	Greenberg	100
10	Faviet	100
11	Chen	100

...

4. Department 80 needs access to its employee data. Create a view named `dept80` that contains the employee numbers, employee last names, and department numbers for all employees in department 80. You have been asked to label the view columns `EMPNO`, `EMPLOYEE`, and `DEPTNO`. For security purposes, do not allow an employee to be reassigned to another department through the view.
5. Display the structure and contents of the `DEPT80` view.

```
DESCRIBE dept80
Name     Null     Type
-------- -------- ------------
EMPNO    NOT NULL NUMBER(6)
EMPLOYEE NOT NULL VARCHAR2(25)
DEPTNO            NUMBER(4)
```

	EMPNO	EMPLOYEE	DEPTNO
1	145	Russell	80
2	146	Partners	80
3	147	Errazuriz	80
4	148	Cambrault	80
5	149	Zlotkey	80
6	150	Tucker	80
7	151	Bernstein	80
8	152	Hall	80
9	153	Olsen	80
10	154	Cambrault	80
11	155	Tuvault	80

...

6. Test your view. Attempt to reassign Abel to department 50.

```
Error report:
SQL Error: ORA-01402: view WITH CHECK OPTION where-clause violation
01402. 00000 -  "view WITH CHECK OPTION where-clause violation"
*Cause:
*Action:
```

7. Run `lab_04_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: `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.

	VIEW_NAME	TEXT
1	DEPT50	SELECT employee_id empno, last_name employee, department_id deptno FROM
2	DEPT80	SELECT employee_id empno, last_name employee, department_id deptno FROM
3	EMPLOYEES_VU	SELECT employee_id, last_name employee, department_id FROM employees
4	EMP_DETAILS_VIEW	SELECT e.employee_id, e.job_id, e.manager_id, e.department_id, d.location_id, l.count

8. Remove the views created in this practice.

Solution 4-1: Creating Views

1. The staff in the HR department wants to hide some of the data in the EMPLOYEES table. Create a view called EMPLOYEES_VU based on the employee numbers, employee last names, and department numbers from the EMPLOYEES table. The heading for the employee name should be EMPLOYEE.

```
CREATE OR REPLACE VIEW employees_vu AS
    SELECT employee_id, last_name employee, department_id
    FROM employees;
```

2. Confirm that the view works. Display the contents of the EMPLOYEES_VU view.

```
SELECT   *
FROM     employees_vu;
```

3. Using your EMPLOYEES_VU view, write a query for the HR department to display all employee names and department numbers.

```
SELECT   employee, department_id
FROM     employees_vu;
```

4. Department 80 needs access to its employee data. Create a view named DEPT80 that contains the employee numbers, employee last names, and department numbers for all employees in department 80. They have requested that you label the view columns EMPNO, EMPLOYEE, and DEPTNO. For security purposes, do not allow an employee to be reassigned to another department through the view.

```
CREATE VIEW dept80 AS
   SELECT   employee_id empno, last_name employee,
            department_id deptno
   FROM     employees
   WHERE    department_id = 80
   WITH CHECK OPTION CONSTRAINT emp_dept_80;
```

5. Display the structure and contents of the DEPT80 view.

```
DESCRIBE dept80

SELECT   *
FROM     dept80;
```

6. Test your view. Attempt to reassign Abel to department 50.

```
UPDATE   dept80
SET      deptno = 50
WHERE    employee = 'Abel';
```

The error is because the `dept80` view has been created with the `WITH CHECK OPTION` constraint. This ensures that the `DEPTNO` column in the view is protected from being changed.

7. Run `lab_04_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;
```

8. Remove the views created in this practice.

```
DROP VIEW employees_vu;
DROP VIEW dept80;
DROP VIEW dept50;
```

Practices for Lesson 5: Managing Schema Objects

Chapter 5

Practices for Lesson 5: Overview

Practice Overview

This practice covers the following topics:

- Adding and dropping constraints
- Deferring constraints
- Creating external tables

Note: Before starting this practice, execute the **`/home/oracle/labs/sql2/code_ex/ /cleanup_scripts/cleanup_05.sql`** script.

Practice 5-1: Managing Schema Objects

Overview

In this practice, you add, drop, and defer constraints. You also create external tables.

Note: Execute the `cleanup_05.sql` script from **`/home/oracle/labs/sql2/code_ex/ /cleanup_scripts/`** before performing the following tasks.

Tasks

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. Populate the `DEPT2` table with data from the `DEPARTMENTS` table. Include only the columns that you need. Confirm that the rows are inserted.

	ID	NAME
1	10	Administration
2	20	Marketing
3	30	Purchasing
4	40	Human Resources
5	50	Shipping
6	60	IT
7	70	Public Relations
8	80	Sales
9	90	Executive
10	100	Finance
11	110	Accounting
12	120	Treasury
13	130	Corporate Tax
14	140	Control And Credit
15	150	Shareholder Services

...

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

```
Name       Null Type
---------- ---- ------------
ID              NUMBER(7)
LAST_NAME       VARCHAR2(25)
FIRST_NAME      VARCHAR2(25)
DEPT_ID         NUMBER(7)
```

4. 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`.
5. Create a `PRIMARY KEY` constraint to the `DEPT2` table by using the `ID` column. The constraint should be named at creation. Name the constraint `my_dept_id_pk`.
6. 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`.
7. 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.
8. Drop the `EMP2` and `DEPT2` tables so that they cannot be restored.
9. Create an external table `library_items_ext`. Use the `ORACLE_LOADER` access driver.

   Note: The `emp_dir` directory object has to be created at first. Refer to the solution on how to create `emp_dir` directory object. Ensure that the external file and the database are on the same machine.

   `library_items.dat` can be found in the `labs/sql2/emp_dir/` folder.

   `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_05_09.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_05_09_soln.sql`. Run the script to create the external table.

   b. Query the `library_items_ext` table.

	CATEGORY_ID	BOOK_ID	BOOK_PRICE	QUANTITY
1	2354	2264	13.21	150
2	2355	2289	46.23	200
3	2355	2264	50	100

10. The `HR` department needs a report of the addresses of all departments. Create an external table as `dept_add_ext` by 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_05_10.sql` file. Observe the code snippet to create the `dept_add_ext` external table. Then replace **<*TODO1*>**, **<*TODO2*>** and **<*TODO3*>** as appropriate and save the script as `lab_05_10_soln.sql`.

    b. Run the `lab_05_10_soln.sql` script to create the external table.

c. Query the `dept_add_ext` table.

	LOCATION_ID	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 America
6	1500	2011 Interiors Blvd	South San Francisco	California	United States of America
7	1600	2007 Zagora St	South Brunswick	New Jersey	United States of America
8	1700	2004 Charade Rd	Seattle	Washington	United States of America
9	1800	147 Spadina Ave	Toronto	Ontario	Canada
10	1900	6092 Boxwood St	Whitehorse	Yukon	Canada

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

11. 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_05_11_a.sql` file to create the `emp_books` table. Observe that the `emp_books_pk` primary key is not created as deferrable.

```
table EMP_BOOKS created.
```

b. Run the `lab_05_11_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.
```

c. 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
```

d. Drop the `emp_books_pk` constraint.

```
table EMP_BOOKS altered.
```

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

```
table EMP_BOOKS altered.
```

f. Set the `emp_books_pk` constraint as deferred.

```
constraint EMP_BOOKS_PK succeeded.
```

g. Run the `lab_05_11_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 starting at line 1 in command:
commit
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.
```

Solution 5-1: Managing Schema Objects

Solution

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. Confirm that the rows are inserted.

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

SELECT * FROM dept2;
```

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

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

DESCRIBE emp2
```

4. 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);
```

5. Create a `PRIMARY KEY` constraint to the `DEPT2` table by 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);
```

6. 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);
```

7. 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);
```

8. Drop the `EMP2` and `DEPT2` tables so that they cannot be restored.

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

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

   Note: The `emp_dir` directory object has to be created at first.

```
CREATE OR REPLACE DIRECTORY emp_dir
AS '/home/oracle/labs/sql2/emp_dir';
GRANT READ ON DIRECTORY emp_dir TO teach_b;
```

   `library_items.dat` can be found in the `labs/sql2/emp_dir/` folder.
   Ensure that the external file and the database are on the same machine.

   `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_05_09.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_05_09_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;
```

   b. Query the `library_items_ext` table.

```
SELECT * FROM library_items_ext;
```

10. The `HR` department needs a report of addresses of all the departments. Create an external table as `dept_add_ext` by 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. Ensure that the external file and the database are on the same machine.

a. Open the `lab_05_10.sql` file. Observe the code snippet to create the `dept_add_ext` external table. Then replace **<*TODO1*>**, **<*TODO2*>** and **<*TODO3*>** as appropriate and save the script as `lab_05_10_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 ('ora21_emp4.exp','ora21_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 `ora21_emp4.exp` and `ora21_emp5.exp` are created in the default directory `emp_dir`.

b. Run the `lab_05_10_soln.sql` script to create the external table.

c. Query the `dept_add_ext` table.

```
SELECT * FROM dept_add_ext;
```

11. 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_05_11_a.sql` script to create the `emp_books` table. Observe that the `emp_books_pk` primary key is not created as deferrable.

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

b. Run the `lab_05_11_b.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;
```

g. Run the `lab_05_11_g.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 by the database at this point, because `COMMIT` failed due to the constraint violation.

Practices for Lesson 6: Retrieving Data by Using Subqueries

Chapter 6

Practices for Lesson 6: Overview

Practice Overview

This practice covers the following topics:

- Creating multiple-column subqueries
- Writing correlated subqueries
- Using the `EXISTS` operator
- Using scalar subqueries
- Using the `WITH` clause

Practice 6-1: Retrieving Data by Using Subqueries

Overview

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

Tasks

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
4	Abel	80	11000
5	Cambrault	80	11000
6	Vishney	80	10500
7	Zlotkey	80	10500
8	Bloom	80	10000
9	King	80	10000
10	Tucker	80	10000
11	Greene	80	9500

...

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

	LAST_NAME	DEPARTMENT_NAME	SALARY
1	King	Executive	24000
2	De Haan	Executive	17000
3	Kochhar	Executive	17000
4	Greenberg	Finance	12008
5	Faviet	Finance	9000
6	Chen	Finance	8200
7	Sciarra	Finance	7700
8	Urman	Finance	7800
9	Popp	Finance	6900
10	Raphaely	Purchasing	11000
11	Khoo	Purchasing	3100

...

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

   Note: Do not display Kochhar in the result set.

	LAST_NAME	HIRE_DATE	SALARY
1	De Haan	13-JAN-09	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 on salary from the highest to the lowest.

	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 the 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	3475.56
2	Chung	3800	50	3475.56
3	Kaufling	7900	50	3475.56
4	Mourgos	5800	50	3475.56
5	Bell	4000	50	3475.56
6	Rajs	3500	50	3475.56
7	Everett	3900	50	3475.56
8	Sarchand	4200	50	3475.56
9	Bull	4100	50	3475.56
10	Vollman	6500	50	3475.56
11	Ladwig	3600	50	3475.56
12	Dilly	3600	50	3475.56
13	Weiss	8000	50	3475.56

...

7. Find all employees who are not supervisors.

   a. First, do this by using the `NOT EXISTS` operator.

	LAST_NAME
1	Abel
2	Ande
3	Atkinson
4	Austin
5	Baer
6	Baida
7	Banda
8	Bates
9	Bell
10	Bernstein
11	Bissot
12	Bloom
13	Bull
14	Cabrio
15	Cambrault

...

   b. Can this be done by using the `NOT IN` operator? How, or why not? If not, try out using another solution.

	LAST_NAME
1	Abel
2	Ande
3	Atkinson
4	Austin
5	Baer
6	Baida
7	Banda
8	Bates
9	Bell
10	Bernstein
11	Bissot
12	Bloom

...

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

	LAST_NAME
1	Chen
2	Sciarra
3	Urman
4	Popp
5	Khoo
6	Baida
7	Tobias
8	Himuro
9	Colmenares
10	Kochhar
11	De Haan
12	Fay
13	Gietz
14	Nayer

...

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.

	LAST_NAME
1	De Haan
2	Austin
3	Lorentz
4	Pataballa
5	Faviet
6	Sciarra
7	Tobias
8	Bell
9	Sarchand
10	Rajs
11	Ladwig
12	Mallin
13	Kaufling

...

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
6	102	De Haan	Executive
7	109	Faviet	Finance
8	108	Greenberg	Finance
9	112	Urman	Finance
10	111	Sciarra	Finance
11	110	Chen	Finance
12	113	Popp	Finance
13	203	Mavris	Human Resources
14	107	Lorentz	IT
15	106	Pataballa	IT

...

102	140	Patel	Shipping
103	141	Rajs	Shipping
104	142	Davies	Shipping
105	143	Matos	Shipping
106	181	Fleaur	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`.

	DEPARTMENT_NAME	DEPT_TOTAL
1	Sales	304500
2	Shipping	156400

Solution 6-1: Retrieving Data by Using Subqueries

Solution

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 `job_ID` match the salary and `job_ID` of any employee located in location `ID` 1700.

```
SELECT e.last_name, d.department_name, e.salary
FROM   employees e JOIN departments d
ON  e.department_id = d.department_id
AND  (salary, job_id) IN
                         (SELECT e.salary, e.job_id
                           FROM   employees e JOIN
departments d
                            ON   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 `manager_ID` as Kochhar.

   Note: Do not display Kochhar in the result set.

```
SELECT last_name, hire_date, salary
FROM   employees
WHERE  (salary, manager_id) IN
             (SELECT salary, manager_id
              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;
```

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 the 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.

```
SELECT e.last_name ename, e.salary salary,
       e.department_id deptno, ROUND(AVG(a.salary),2)
dept_avg
  FROM   employees e JOIN employees a
  ON  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);
```

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`. A much better solution would be a subquery like the following:

```
SELECT last_name
FROM employees
WHERE employee_id NOT IN (SELECT manager_id
                          FROM employees WHERE manager_id IS NOT
NULL);
```

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);
```

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 JOIN departments d
  ON 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 for Lesson 7: Manipulating Data by Using Subqueries

Chapter 7

Practices for Lesson 7: Overview

Practices Overview

This practice covers the following topics:

- Using subqueries to manipulate data
- Inserting values by using a subquery as a target
- Using the `WITH CHECK OPTION` keyword on DML statements
- Using correlated subqueries to update and delete rows

Practice 7-1: Manipulating Data by Using Subqueries

Overview

In this practice, you test your knowledge about using subqueries to manipulate data, the `WITH CHECK OPTION` keyword on DML statements, and correlated subqueries to update and delete rows.

Tasks

1. Which of the following statements are true?
   a. Subqueries are used to retrieve data by using an inline view.
   b. Subqueries cannot be used to copy data from one table to another.
   c. Subqueries update data in one table based on the values of another table.
   d. Subqueries delete rows from one table based on rows in another table.

2. Fill in the blanks:
   a. You can use a subquery in place of the table name in the _____ clause of the `INSERT` statement.

   Options:
   1) `FROM`
   2) `INTO`
   3) `FOR UPDATE`
   4) `VALUES`

3. The `WITH CHECK OPTION` keyword prohibits you from changing rows that are not in the subquery.
   a. `TRUE`
   b. `FALSE`

4. The `SELECT` list of a subquery must have the same number of columns as the column list of the `VALUES` clause.
   a. `TRUE`
   b. `FALSE`

5. You can use a correlated subquery to delete only those rows that also exist in another table.
   a. `TRUE`
   b. `FALSE`

6. To understand the concepts of `WITH CHECK OPTION` and correlated subqueries, run the demo files for this practice.

Solution 7-1: Manipulating Data by Using Subqueries

1. Which of the following statements are true?
   a. Subqueries are used to retrieve data by using an inline view.
   b. Subqueries cannot be used to copy data from one table to another.
   c. Subqueries update data in one table based on the values of another table.
   d. Subqueries delete rows from one table based on rows in another table.

   Answer: a, c, and d

2. Fill in the blanks:
   a. You can use a subquery in place of the table name in the _____ clause of the `INSERT` statement.

   Options:
   1) `FROM`
   2) `INTO`
   3) `FOR UPDATE`
   4) `VALUES`

   Answer: 2

3. The `WITH CHECK OPTION` keyword prohibits you from changing rows that are not in the subquery.
   a. `TRUE`
   b. `FALSE`

   Answer: a

4. The `SELECT` list of a subquery must have the same number of columns as the column list of the `VALUES` clause.
   a. `TRUE`
   b. `FALSE`

   Answer: a

5. You can use a correlated subquery to delete only those rows that also exist in another table.
   a. `TRUE`
   b. `FALSE`

   Answer: a

6. To understand the concepts of `WITH CHECK OPTION` and correlated subqueries, run the demo files for this practice.

Practices for Lesson 8: Controlling User Access

Chapter 8

Practices for Lesson 8: Overview

Practice Overview:

This practice covers the following topics:

- Granting privileges to other users on your table
- Modifying another user's table through the privileges granted to you

Practice 8-1: Controlling User Access

Overview

You grant query privilege on your table to another user. You learn how to control access to database objects.

Tasks

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 *`ora21`* account and the corresponding password to log on to the database.**
3. **Open another SQL Developer session and connect as *`ora22`*.**

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

   Note: For this exercise, open another SQL Developer session and connect as a different user. For example, if you are currently using `ora21`, open another SQL Developer session and connect as `ora22`.

   a. Grant another user (for example, `ora22`) 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.

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

c. Have the user pass on the query privilege to a third user, `ora23`.

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

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.
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
9	90	Executive	100	1700
10	100	Finance	108	1700
11	110	Accounting	205	1700
12	120	Treasury	(null)	1700
13	130	Corporate Tax	(null)	1700
14	140	Control And Credit	(null)	1700
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

. . .

13. Add a new row to your `DEPARTMENTS` table. `ora21` should add `Education` as department number `500`. `ora22` should add `Human Resources` as department number `510`. Query `ora22`'s table from `ora21` and vice versa.
14. Create a synonym for the `ora22`'s `DEPARTMENTS` table from `ora21` and vice versa.
15. Query all the rows in the `ora22`'s `DEPARTMENTS` table by using your synonym and vice versa.

`ora21` `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)

`ora22` `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 `ora22` and vice versa.
17. Remove the row that you inserted into the `DEPARTMENTS` table in step 13 and save the changes.
18. Drop the synonyms you created in Step 14.

Solution 8-1: Controlling User Access

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 WHERE GRANT OPTION;
```

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

   Note: For this exercise, open another SQL Developer session and connect as a different user. For example, if you are currently using `ora21`, open another SQL Developer session and connect as `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.

      `ora21` executes this statement:

```
GRANT select
ON regions
TO ora22 WITH GRANT OPTION;
```

   b. Have the user query your `REGIONS` table.

      `ora22` executes this statement:

```
SELECT * FROM ora21.regions;
```

   c. Have the user pass on the query privilege to a third user, `ora23`.

      `ora22` executes this statement.

```
GRANT select
ON ora21.regions
TO ora23;
```

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

      `ora21` executes this statement.

```
REVOKE select
ON regions
FROM ora22;
```

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.

   `ora21` executes this statement.

```
GRANT select, update, insert
ON COUNTRIES
TO ora22;
```

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

    `ora21` executes this statement.

```
REVOKE select, update, insert ON COUNTRIES FROM ora22;
```

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

    a. `ora22` executes the `GRANT` statement.

```
GRANT select
ON departments
TO ora21;
```

b. ora21 executes the GRANT statement.

```
GRANT select
ON departments
TO ora22;
```

12. Query all the rows in your DEPARTMENTS table.

```
SELECT  *
FROM    departments;
```

13. Add a new row to your DEPARTMENTS table. ora21 should add Education as department number 500. ora22 should add Human Resources as department number 510. Query ora22's table from ora21 and vice versa.

a. ora21 executes this INSERT statement.

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

b. ora22 executes this INSERT statement.

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

c. ora21 executes this SELECT statement.

```
SELECT * FROM ora22.DEPARTMENTS;
```

d. ora22 executes this SELECT statement.

```
SELECT * FROM ora21.DEPARTMENTS;
```

14. Create a synonym for the ora22's DEPARTMENTS table from ora21 and vice versa.

a. ora21 creates a synonym named user2.

```
CREATE SYNONYM user2
        FOR ora22.DEPARTMENTS;
```

b. ora22 creates a synonym named user1.

```
CREATE SYNONYM user1
        FOR ora21.DEPARTMENTS;
```

15. Query all the rows in the ora22's DEPARTMENTS table by using your synonym and vice versa.

a. ora21 executes this SELECT statement.

```
SELECT  *
   FROM    user2;
```

b. ora22 executes this SELECT statement.

```
SELECT  *
    FROM    user1;
```

16. Revoke the `SELECT` privilege from `ora22` and vice versa.

    a. `ora21` revokes the privilege.

```
REVOKE select
     ON departments
     FROM  ora22;
```

    b. `ora22` revokes the privilege.

```
REVOKE select
     ON departments
     FROM ora21;
```

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

    a. `ora21` executes this `DELETE` statement.

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

    b. `ora22` executes this `DELETE` statement.

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

18. Drop the synonyms you created in Step 14.

```
DROP SYNONYM user1;
DROP SYNONYM user2;
```

Practices for Lesson 9: Manipulating Data Using Advanced Queries

Chapter 9

Practices for Lesson 9: Overview

Practice Overview

This practice covers the following topics:

- Performing multitable `INSERT`s
- Performing `MERGE` operations
- Performing flashback operations
- Tracking row versions

Note: Before starting this practice, execute the **`/home/oracle/labs/sql2/code_ex/cleanup_scripts/cleanup_09.sql`** script.

Practice 9-1: Manipulating Data

Overview

In this practice, you perform multitable `INSERT` and `MERGE` operations, and the flashback operation, and track row versions.

Note: Execute the `cleanup_09.sql` script from **`/home/oracle/labs/sql2/code_ex/cleanup_scripts/`** before performing the following tasks.

Tasks

1. Run the `lab_09_01.sql` script in the labs folder to create the `SAL_HISTORY`, `MGR_HISTORY and SPECIAL_SAL` tables.
2. Display the structure of the `SAL_HISTORY` table.

```
Name        Null Type
----------- ---- -----------
EMPLOYEE_ID      NUMBER(6)
HIRE_DATE        DATE
SALARY           NUMBER(8,2)
```

3. Display the structure of the `MGR_HISTORY` table.

```
Name        Null Type
----------- ---- -----------
EMPLOYEE_ID      NUMBER(6)
MANAGER_ID       NUMBER(6)
SALARY           NUMBER(8,2)
```

4. Display the structure of the `SPECIAL_SAL` table.

```
Name        Null Type
----------- ---- -----------
EMPLOYEE_ID      NUMBER(6)
SALARY           NUMBER(8,2)
```

5.
   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.
      - If the salary is less than $20,000:
        - 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-09	17000
2	102	13-JAN-09	17000
3	103	03-JAN-14	9000
4	104	21-MAY-15	6000
5	105	25-JUN-13	4800
6	106	05-FEB-14	4800
7	107	07-FEB-15	4200
8	108	17-AUG-10	12008
9	109	16-AUG-10	9000
10	110	28-SEP-13	8200
11	111	30-SEP-13	7700

...

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	108	101	12008
9	109	108	9000
10	110	108	8200
11	111	108	7700
12	112	108	7800
13	113	108	6900

...

6.

a. Run the `lab_09_06_a.sql` script in the lab folder to create the `SALES_WEEK_DATA` table.

b. Run the `lab_09_06_b.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.

```
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)
```

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_09_06_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)
```

g. Write a query to do the `following`:

- Retrieve details such as 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.

h. Display the records from the `EMP_SALES_INFO` table.

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

7. 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 this, first create an external table called `EMP_DATA` using the `emp.dat` source file in the `emp_dir` directory. Use the `lab_09_07.sql` script to do this.
8. Run `the lab_09_08.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
12	Harrison	Bloom	HBLOOM

...

9. 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

10. Drop the `EMP2` table.
11. Query the recycle bin to see whether the table is present.
12. Restore the `EMP2` table to a state before the `DROP` statement.
13. Create the `EMP3` table using the `lab_09_13.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.

	START_DATE	END_DATE	DEPARTMENT_ID
1	31-AUG-16 10.56.49.000000000 PM	(null)	50
2	31-AUG-16 10.56.49.000000000 PM	31-AUG-16 10.56.49.000000000 PM	60
3	(null)	31-AUG-16 10.56.49.000000000 PM	90

14. Drop the `EMP2` and `EMP3` tables so that they cannot be restored. Check in the recycle bin.

Solution 9-1: Manipulating Data

Solution

1. Run the `lab_09_01.sql` script in the lab folder to create the `SAL_HISTORY, MGR_HISTORY and SPECIAL_SAL` tables.
2. Display the structure of the `SAL_HISTORY` table.

```
DESC sal_history
```

3. Display the structure of the `MGR_HISTORY` table.

```
DESC mgr_history
```

4. Display the structure of the `SPECIAL_SAL` table.

```
DESC special_sal
```

5.
   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.
      - If the salary is less than $20,000:
        - 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 > 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;
```

6.

a. Run the lab_09_06_a.sql script in the lab folder to create the SALES_WEEK_DATA table.

b. Run the lab_09_06_b.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;
```

e. Run the lab_09_06_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;
```

7. 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 this, first create an external table called `EMP_DATA` by using the `emp.dat` source file in the `emp_dir` directory. You can use the script in `lab_09_07.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') ) ;
```

8. Run the `lab_09_08.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)
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;
```

9. 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

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

DESCRIBE emp2
```

10. Drop the `EMP2` table.

```
DROP TABLE emp2;
```

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

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

12. Restore the `EMP2` table to a state before the `DROP` statement.

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

13. Create the `EMP3` table by using the `lab_09_13.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';
```

14. Drop the `EMP2` and `EMP3` tables, so that they cannot be restored. Check in the recycle bin.

```
DROP TABLE emp2 PURGE;
DROP TABLE emp3 PURGE;

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

Practices for Lesson 10: Managing Data in Different Time Zones

Chapter 10

Practices for Lesson 10: Overview

Practice Overview

This practice covers using the datetime functions.

Note: Before starting this practice, execute the `/home/oracle/labs/sql2/code_ex/cleanup_scripts/cleanup_10.sql` script.

Practice 10-1: Managing Data in Different Time Zones

Overview

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

Note: Execute the `cleanup_10.sql` script from `/home/oracle/labs/sql2/code_ex/cleanup_scripts/cleanup_10.sql` before performing the following tasks.

Tasks

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

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

      - Singapore

	TZ_OFFSET('SINGAPORE')
1	+08:00

      - Egypt

	TZ_OFFSET('EGYPT')
1	+02:00

   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.

	CURRENT_DATE	CURRENT_TIMESTAMP	LOCALTIMESTAMP
1	31-AUG-2016 16:27:13	31-AUG-16 04.27.13.970275000 PM US/PACIFIC-NEW	31-AUG-16 04.27.13.970275000 PM

   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.

	CURRENT_DATE	CURRENT_TIMESTAMP	LOCALTIMESTAMP
1	01-SEP-2016 07:29:08	01-SEP-16 07.29.08.647709000 AM +08:00	01-SEP-16 07.29.08.647709000 AM

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

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

	DBTIMEZONE	SESSIONTIMEZONE
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.

	LAST_NAME	EXTRACT(YEARFROMHIRE_DATE)
1	Russell	2012
2	Partners	2013
3	Errazuriz	2013
4	Cambrault	2015
5	Zlotkey	2016
6	Tucker	2013
7	Bernstein	2013
8	Hall	2013
9	Olsen	2014
10	Cambrault	2014

...

5. Alter the session to set `NLS_DATE_FORMAT` to `DD-MON-YYYY`.
6. Examine and run the `lab_10_06.sql` script to create the `SAMPLE_DATES` table and populate it.

Note: The screenshot dates will change according to the sysdate.

a. Select from the table and view the data.

	DATE_COL
1	31-AUG-2016

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

	DATE_COL
1	31-AUG-16 11.33.07.000000000 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 2010, 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.

	LAST_NAME	Review
1	De Haan	not this year!
2	Kochhar	not this year!
3	Higgins	Needs Review
4	Gietz	Needs Review
5	Baer	Needs Review
6	Mavris	Needs Review
7	Faviet	Needs Review
8	Greenberg	Needs Review
9	Raphaely	Needs Review

…

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 matches, print `maybe next year!` Sort the results by the `HIRE_DATE` column. Use the `EMPLOYEES` table.
Hint: Use `CASE` expressions and `TO_YMINTERVAL`.

	LAST_NAME	HIRE_DATE	SYSDATE	Awards
1	De Haan	13-JAN-2009	31-AUG-2016	5 years of service
2	Kochhar	21-SEP-2009	31-AUG-2016	5 years of service
3	Higgins	07-JUN-2010	31-AUG-2016	5 years of service
4	Gietz	07-JUN-2010	31-AUG-2016	5 years of service
5	Baer	07-JUN-2010	31-AUG-2016	5 years of service
6	Mavris	07-JUN-2010	31-AUG-2016	5 years of service
7	Faviet	16-AUG-2010	31-AUG-2016	5 years of service
8	Greenberg	17-AUG-2010	31-AUG-2016	5 years of service
9	Raphaely	07-DEC-2010	31-AUG-2016	5 years of service

…

Solution 10-1: Managing Data in Different Time Zones

Solution

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';
OR
ALTER SESSION SET TIME_ZONE = 'US/Pacific-New';
```

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';
OR
ALTER SESSION SET TIME_ZONE = 'Singapore';
```

e. 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;
```

Note: Observe in the 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_10_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;
```

   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.

```
Error report:
SQL Error: ORA-01439: column to be modified must be empty to change datatype
01439. 00000 -  "column to be modified must be empty to change datatype"
*Cause:
*Action:
```

7. Create a query to retrieve last names from the `EMPLOYEES` table and calculate the review status. If the year hired was 2010, 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 2010 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 matches, 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
    ORDER BY hire_date;
```

Additional Practices and Solutions

Chapter 11

Additional Practices and Solutions

Practices Overview

You will be working on extra practices that are based on the following topics:

- Data manipulation language (DML) statements
- Data definition language (DDL) statements
- Datetime functions
- Advanced subqueries

Additional Practices

Overview

The following exercises can be used for extra practice after you have discussed DML and DDL statements in the lessons titled “Managing Schema Objects” and “Manipulating Data Using Advanced Queries.”

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.

Tasks

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-2012	13000
2	202	17-AUG-2013	6000
3	203	07-JUN-2010	6500
4	204	07-JUN-2010	10000
5	205	07-JUN-2010	12008
6	206	07-JUN-2010	8300

```
MGR_HISTORY
```

	EMPLOYEE_ID	MANAGER_ID	SALARY
1	201	100	13000
2	202	201	6000
3	203	101	6500
4	204	101	10000
5	205	101	12008
6	206	205	8300

3. Nita, the DBA, needs you to create a table that 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	Location_id	Location_name
Primary Key	Yes	
Data Type	Number	`VARCHAR2`
Length	4	20

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

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

   Note: The results are based on a different date, and in some cases, they will not match the actual results that the students get. In addition, the time zone offset of the various countries may differ, based on daylight saving time.

   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 may be different based on the date when the command is executed.

	SYSDATE	CURRENT_DATE	CURRENT_TIMESTAMP	LOCALTIMESTAMP
1	31-AUG-2016 23:57:26	01-SEP-2016 09:57:26	01-SEP-16 09.57.26.725658000 AM +10:00	01-SEP-16 09.57.26.725658000 AM

   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. In addition, 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.

	SYSDATE	CURRENT_DATE	CURRENT_TIMESTAMP	LOCALTIMESTAMP
1	31-AUG-2016 23:58:32	31-AUG-2016 17:58:32	31-AUG-16 05.58.32.071102000 PM -06:00	31-AUG-16 05.58.32.071102000 PM

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.

7. The Human Resources department wants a list of employees who are up for review in January, so the department has 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	De Haan	1	13-JAN-2009
2	Hunold	1	03-JAN-2014
3	Landry	1	14-JAN-2015
4	Davies	1	29-JAN-2013
5	Partners	1	05-JAN-2013
6	Zlotkey	1	29-JAN-2016
7	Tucker	1	30-JAN-2013
8	King	1	30-JAN-2012
9	Marvins	1	24-JAN-2016
10	Fox	1	24-JAN-2014
11	Johnson	1	04-JAN-2016
12	Taylor	1	24-JAN-2014
13	Sarchand	1	27-JAN-2012
14	Grant	1	13-JAN-2016

The following exercises can be used for extra practice after you have discussed advanced subqueries.

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.

	LAST_NAME	SALARY
1	King	24000
2	Kochhar	17000
3	De Haan	17000

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.

	EMPLOYEE_ID	LAST_NAME
1	120	Weiss
2	121	Fripp
3	122	Kaufling
4	123	Vollman
5	124	Mourgos
6	125	Nayer
7	126	Mikkilineni
8	127	Landry
9	128	Markle
10	129	Bissot
11	130	Atkinson
12	131	Marlow
13	132	Olson
14	133	Mallin
15	134	Rogers
16	135	Gee
17	136	Philtanker
18	137	Ladwig

…

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.

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
5	Finance Manager	12008
6	Accounting Manager	12008

Additional Practices Solutions

Solutions

The following exercises can be used for extra practice after you have discussed DML and DDL statements in the lessons titled “Managing Schema Objects” and “Manipulating Data Using Advanced Queries.”

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

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, the department has 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 that 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	Location_id	Location_name
Primary Key	Yes	
Data Type	`Number`	`VARCHAR2`
Length	4	20

```
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';
```

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;
```

Note: The results are based on a different date, and in some cases, they will not match the actual results that the students get. In addition, the time zone offset of the various countries may differ, based on daylight saving time.

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. In addition, 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.
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.

7. The Human Resources department wants a list of employees who are up for review in January, so the department has 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.

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);
```

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 = d.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.

```
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:

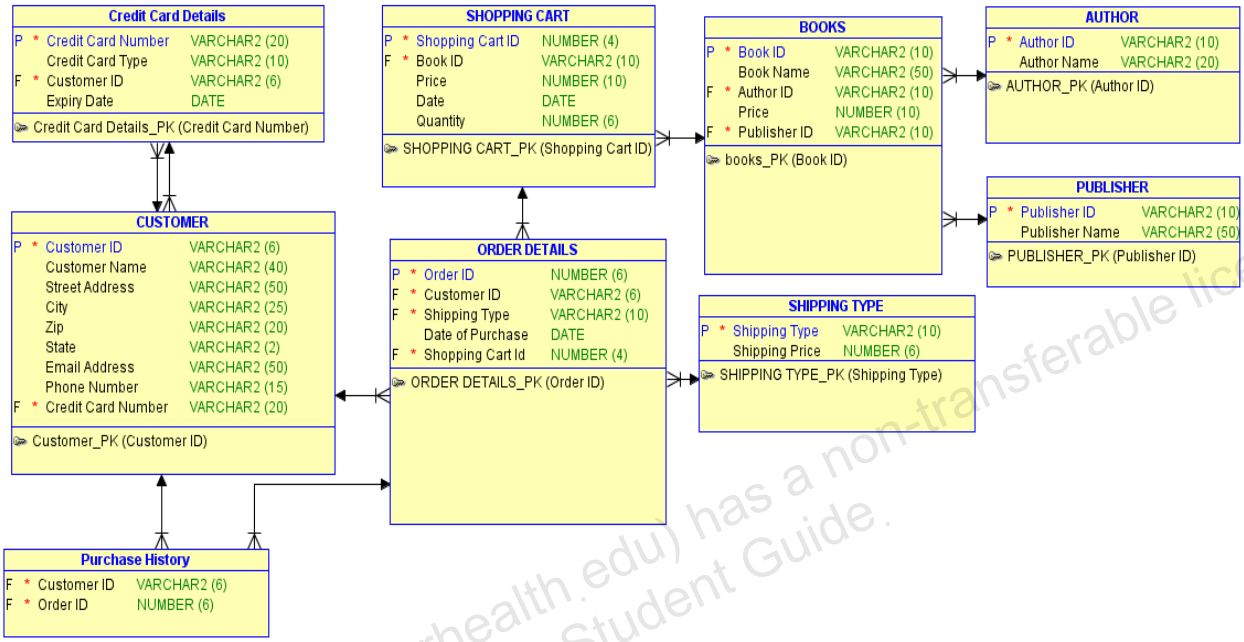
    Write a query to display the job `IDs` of those jobs whose maximum salary is above half the maximum salary in the entire company. Use the `WITH` clause to write this query. Name the query `MAX_SAL_CALC`.

```
 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

In the case study for the *SQL WORKSHOP I* course, you built a set of database tables for an Online Book Store application. In addition, you inserted, updated, and deleted records in an online book 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 `Online_Book_Store_Drop_Tables.sql` script in the labs folder to drop tables if they already exist. Then run the `Online_Book_Store_Populate.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.

	TABLE_NAME	COLUMN_NAME	DATA_TYPE	NULLABLE
1	AUTHOR	AUTHOR_ID	VARCHAR2	N
2	AUTHOR	AUTHOR_NAME	VARCHAR2	Y
3	BOOKS	BOOK_ID	VARCHAR2	N
4	BOOKS	BOOK_NAME	VARCHAR2	Y
5	BOOKS	AUTHOR_ID	VARCHAR2	N
6	BOOKS	PRICE	NUMBER	Y
7	BOOKS	PUBLISHER_ID	VARCHAR2	N
8	CREDIT_CARD_DETAILS	CREDIT_CARD_NUMBER	VARCHAR2	N
9	CREDIT_CARD_DETAILS	CREDIT_CARD_TYPE	VARCHAR2	Y
10	CREDIT_CARD_DETAILS	EXPIRY_DATE	DATE	Y
11	CUSTOMER	CUSTOMER_ID	VARCHAR2	N
12	CUSTOMER	CUSTOMER_NAME	VARCHAR2	Y
13	CUSTOMER	STREET_ADDRESS	VARCHAR2	Y
14	CUSTOMER	CITY	VARCHAR2	Y
15	CUSTOMER	PHONE_NUMBER	VARCHAR2	Y
16	CUSTOMER	CREDIT_CARD_NUMBER	VARCHAR2	N
17	ORDER_DETAILS	ORDER_ID	VARCHAR2	N
18	ORDER_DETAILS	CUSTOMER_ID	VARCHAR2	Y
19	ORDER_DETAILS	SHIPPING_TYPE	VARCHAR2	N
20	ORDER_DETAILS	DATE_OF_PURCHASE	DATE	Y
21	ORDER_DETAILS	SHOPPING_CART_ID	VARCHAR2	N
22	PUBLISHER	PUBLISHER_ID	VARCHAR2	N
23	PUBLISHER	PUBLISHER_NAME	VARCHAR2	Y
24	PURCHASE_HISTORY	CUSTOMER_ID	VARCHAR2	Y
25	PURCHASE_HISTORY	ORDER_ID	VARCHAR2	N
26	SHIPPING_TYPE	SHIPPING_TYPE	VARCHAR2	N
27	SHIPPING_TYPE	SHIPPING_PRICE	NUMBER	Y
28	SHOPPING_CART	SHOPPING_CART_ID	VARCHAR2	N
29	SHOPPING_CART	BOOK_ID	VARCHAR2	N
30	SHOPPING_CART	PRICE	NUMBER	Y
31	SHOPPING_CART	SHOPPING_CART_DATE	DATE	Y
32	SHOPPING_CART	QUANTITY	NUMBER	Y

2. Verify the existence of the `ORDER_ID_SEQ` sequence in the data dictionary.

	SEQUENCE_NAME
1	DEPARTMENTS_SEQ
2	EMPLOYEES_SEQ
3	LOCATIONS_SEQ
4	ORDER_ID_SEQ

3. You want to create some users who have access only to their purchase history. Create a user called Carmen and grant her the privilege to select from the `PURCHASE_HISTORY` table.
4. Add an edition column (varchar2 (6)) to the `BOOKS` table to store the book edition information.
5. Add a `CREDIT_CARD_TYPE` table to store `CREDIT_CARD_TYPE` and `CREDIT_CARD_DESCRIPTION`. The table has a foreign key with the `CREDIT_CARD_TYPE` column in the `CREDIT_CARD_DETAILS` table.
6. Select all the tables from the data dictionary.
7. Create a `SHOPPING_HISTORY` table to store the details of purchase history of the customers.
   Hint: You can copy the `PURCHASE_HISTORY` table.
8. Display the customer details of the first 10 customers who have placed orders in the last month. Order the records based on the customer ID.

	CUSTOMER_ID	ORDER_ID	DATE_OF_PURCHASE	CUSTOMER_NAME
1	CN0001	OD0001	12-JUN-2011	VelasquezCarmen
2	CN0003	OD0003	31-JUL-2014	Nagayama Midori
3	CN0004	OD0004	14-AUG-2016	Quick-To-See Mark
4	CN0009	OD0009	25-NOV-2013	Catchpole Antoinette

9. Show a list of customers who have placed an order more than once.

	CUSTOMER_ID	CUSTOMER_NAME
1	CN0001	VelasquezCarmen
2	CN0003	Nagayama Midori
3	CN0004	Quick-To-See Mark
4	CN0009	Catchpole Antoinette

Additional Practices Solution: Case Study

Solution

First, run the `Online_Book_Store_Drop_Tables.sql` script in the labs folder to drop tables if they already exist. Then run the `Online_Book_Store_Populate.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('CUSTOMER','CREDIT_CARD_DETAILS','SHOPPING_CART',
'ORDER_DETAILS','BOOKS','AUTHOR','PUBLISHER','SHIPPING_TYPE',
'PURCHASE_HISTORY')
ORDER BY table_name;
```

2. Verify the existence of the `ORDER_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 purchase history. Create a user called Carmen and grant her the privilege to select from the `PURCHASE_HISTORY` table.

```
CREATE USER carmen IDENTIFIED BY oracle ;
GRANT select ON purchase_history TO carmen;
```

4. Add an edition column (varchar2 (6)) to the `BOOKS` table to store the book edition information.

```
ALTER TABLE books ADD(edition VARCHAR2(6));
```

5. Add a `CREDIT_CARD_TYPE` table to store `CREDIT_CARD_TYPE` and `CREDIT_CARD_DESCRIPTION`. The table has a foreign key with the `CREDIT_CARD_TYPE` column in the `CREDIT_CARD_DETAILS` table.

```
CREATE TABLE CREDIT_CARD_TYPE
   ( CREDIT_CARD_TYPE VARCHAR2(10) NOT NULL ENABLE,
    CREDIT_CARD_DESCRIPTION VARCHAR2(4000 BYTE),
     CONSTRAINT CREDIT_CARD_TYPE_PK PRIMARY KEY
(CREDIT_CARD_TYPE))
;
```

6. Select all the tables from the data dictionary.

```
SELECT table_name FROM user_tables order by table_name;
```

7. Create a `SHOPPING_HISTORY` table to store the details of a purchase history of customers. **Hint:** You can copy the `PURCHASE_HISTORY` table.

```
CREATE TABLE shopping_history as select * from purchase_history;
```

8. Display the customer details of the first 10 customers who have placed orders in the last month. Order the records based on the customer ID.

```
SELECT o.CUSTOMER_ID, o.ORDER_ID, o.DATE_OF_PURCHASE,
c.CUSTOMER_NAME
FROM ORDER_DETAILS o JOIN PURCHASE_HISTORY p
ON o.CUSTOMER_ID = p.CUSTOMER_ID JOIN CUSTOMER c
ON o.CUSTOMER_ID= c.CUSTOMER_ID
AND rownum < 10
ORDER BY CUSTOMER_ID;
```

9. Show a list of customers who have placed an order more than once.

```
SELECT customer_id, customer_name FROM customer c
WHERE 1 <= (select count(*) from purchase_history where
customer_id = c.customer_id);
```