COEN 178 **Intro to Database Systems** **Winter 2021**

Lab 7 (100 pts)

Objectives: Learn

- How to use SQL Views
- How to generate a formatted report using SQL commands.
- How to fix “mutating table errors” caused by triggers.

NOTE: If you don’t’ complete all the exercises in Part 3 within the Lab time, please complete them and submit to the TA before Lab 8.

Part 1 (40 pts)

In this section, you will learn to define SQL Views and use them.

A View is a **virtual table** based on the result-set of an SQL statement on a base table (a base table is a table that actually exists in the database. Eg. All the tables you have been using are base tables). You can think of a view as a “Saved Select statement” with a name. A view can be created from one or many base tables. A view can be defined on another view. Once you create a view, you can issue SQL queries against the view (just like you perform queries on a ase table).

Views can be used to hide a complex queries and make it easy to issue simple queries against the view. They canbe used to restrict access to data and create summaries of reports on data from various tables. Please keep in mind that updating through a view that is defined on multiple tables may not be permitted.

The syntax for defining a view is

```
CREATE Or Replace VIEW view_name AS
SELECT column1, column2.....
FROM table_name
WHERE [condition];
```

To do the exercises on views, let us create a few base tables as follows:

- Create a table called **EMP7**(empid varchar(5) Primary Key, empName varchar(20), deptId varchar(5),salary NUMBER (10,2));
- Create a table called **Project7** (projectID varchar(5) Primary Key, title varchar(20), budget NUMBER(10,2),startDate Date,endDate Date, managerId varchar(5));
- Create a table called **EMP_Project** (projectID varchar(5),empid varchar(5), commission NUMBER(10,2));
- Insert the following data into the tables. NOTE: You must add at least 2 more rows of additional data in to the given tables.

```
insert into EMP7 values ('e1','J.Smith','d1',100000);

insert into EMP7 values ('e2','M.Jones','d1',120000);

insert into EMP7 values ('e3','D.Clark','d2',200000);

insert into  Project7 values ('pj1','Research', 1000000, '10-Jan-
2019','20-Feb-2020','e1');

insert into  Project7 values ('pj2','Research', 100000, '10-Feb-
2019','20-Apr-2019','e2');

insert into  EMP_Project values ('pj2','e2',10000);
```

Question 1 (40 pts)

This question has parts a to g.

a) Create a view (with a create or Replace …) called **CurrentProjects** on **Project7** by selecting the *projectid, title* and *managerid* where the end date comes later than the sysdate.

b) Create a view (with a create or Replace …) called **PastProjects** on **Project7**, by selecting the *title* and *managerid* where the end date is earlier than the sysdate.

c) Do a Select * on the views you have created in a and b. Do they work?

d) Insert the following data

```
insert into CurrentProjects values ('p99','Testing','e2');
```

Now, do a Select * on Project7 and show the row that is inserted. What was inserted for budget,

startdate and enddate?

e) Insert the following data

```
insert into PastProjects values ('Testing','e2');
```

Did it succeed? If not, give reasons.

f) Now, create a view (with a create or Replace …) called **ManagedProjects** on **EMP7** and **Project7**, by selecting the *projectID*, *title (rename it as Project_Title)* and *empName (rename it as ManagerName)* where the EMP7.empid = Project7.managerId.

g) Do a Select * from ManagedProjects.

Example Output(This is an example, you should have the same layout):

```
 PROJECTID   PROJECT_TITLE        MANAGERNAME

-----          -------------------- --------------------

pj1            Research                 J.Smith

p99            Testing                  M.Jones

pj2            Research                 M.Jones
```

h) Now, try to insert the following:

```
insert into ManagedProjects values ('p88','Design','Mary Mason');
```

Did it succeed? If not, why not?

Ref:

Create View:
https://docs.oracle.com/cd/B19306_01/server.102/b14200/statements_8004.htm
Oracle View:
http://www.oracletutorial.com/oracle-view/

Part 2 (35 pts)

In SQLPlus, we can design and generate a report by formatting column headings, variable values etc.

Note: You have to create a report in your Final project.

Read (or refer to) the document, Creating_A_Report.doc.

SQL Report Generation Links:

SQL Report Generation: http://docs.oracle.com/cd/A87860_01/doc/server.817/a82950/ch4.htm

Commands:

Break : https://docs.oracle.com/cd/B19306_01/server.102/b14357/ch12009.htm

Question 2 (10 pts)

Examine the file, **CreateEmpReport.sql**. It is an example file that includes commands (with comments) to format a report in SQLPLUS. Execute the **CreateEmpReport.sql.** We have used the EMP7 table you have created in Part 1 to provide the values for the report. Your output should be in the spooled file, **EmpReport.txt,** which should have the formatted output.

Question 3 (25 pts)

In this exercise, you will use the table **MyExpenses** to create a formatted report from a SQL query that selects all entries from the MyExpenses table.

```
CREATE TABLE MyExpenses(category varchar(15), price NUMBER(10,
2));
insert into MyExpenses values ('Groceries', 60.23);
insert into MyExpenses values ('Groceries', 34.83);
insert into MyExpenses values ('Groceries', 94.32);
insert into MyExpenses values ('Groceries', 29.12);
insert into MyExpenses values ('Entertainment', 15.23);
insert into MyExpenses values ('Entertainment', 23.34);
insert into MyExpenses values ('Entertainment', 44.44);
insert into MyExpenses values ('Entertainment', 10.15);
insert into MyExpenses values ('Gas', 30.23);
insert into MyExpenses values ('Gas', 27.34);
insert into MyExpenses values ('Gas', 35.44);
insert into MyExpenses values ('Gas', 40.15);
```

We will be referencing this website to format the report:

http://docs.oracle.com/cd/B10501_01/server.920/a90842/ch7.htm

Create a sql file called **CreateExpenseReport.sql,** with formatting code in your sql file before the select query. Follow the example in CreateEmpReport.sql. Your output should be spooled to ExpenseReport.txt.

Your CreateEmpReport.sql should have the commands to do the following:

- Create a title called "Expense Report". The title should be at the top and should be centered as much as you can make it. In the btitle, include your name. Refer to Example 7-18, 7-19, and 7-20 in the file at the link given above.

- Change the column headings to more meaningfule names (if they are not already). Refer to Example 7-1 in the file at the link given above.

- For every row in the output table add an extra space. Refer to Example 7-11 in the file at the link given above.

- Make the output table display its numerical values in dollar amounts. It should have a dollar sign before each number and should show two decimal places for the cent values. Refer to Example 7-4.

- Display the average, maximum value, and total amount for groceries, entertainment etc (or refer to the categories you have in your MyExpenses table). These amounts should create 3 rows and be placed at the bottom of the output table and line up with their respective columns. Refer to Example 7-14, 7-15, 7-16.

Spool your output into a file called ExpenseReport.txt and submit it to TA.

Part 3 (25 pts)

In this part, you will learn about **mutating table errors** caused by triggers and some solutions.

Mutating table exceptions occur when we try to reference the triggering table in a query from within row-level trigger code. In other words, a mutating table error (*ORA-04091: table name is mutating, trigger/function may not see it*) occurs when a row-level trigger tries to query or change a table that is already undergoing change (via an INSERT, UPDATE, or DELETE statement).

For example, inserting a row in a table triggers an action to calculate the summary of a column in the same table.

Since, it is the row-level triggers that cause the mutating table errors, a row-level trigger may not read or write the table from which it has been fired. However, statement level triggers are free to both read and modify the triggering table.

Question 4 (5 pts)

Step 1: Create a table as shown below:

```
Create table ItemOrder (orderNo VARCHAR(5) Primary key, qty Integer);
```

Step 2: Now define a trigger as shown below (you can copy and paste the code at SQL prompt)

```
CREATE OR REPLACE TRIGGER ItemOrder_after_insert_trig
AFTER INSERT
   ON ItemOrder
   FOR EACH ROW

DECLARE
   v_quantity Integer;

BEGIN
   SELECT qty
   INTO v_quantity
   FROM ItemOrder
   WHERE orderNo = 'o1';

END;
/
Show Errors;
```

Step 3:

Try to insert the following row into ItemOrder.

```
Insert into ItemOrder values ('o1',100);
```

Q: Did you succeed? What did you see?

A:

The solution to the above problem is not to include the query in the trigger. But sometimes, it becomes necessary to include a query statement to enforce a constraint.

Question 5 (10 pts)

Create the tables **COURSE** and **COURSE_PREREQ** as shown below, where COURSE contains the information about a course with a courseNo and name.

The Course_Prereq has the courseNo and its prerequisite (which is a Course).

Q: Why do you think the prerequisites are in a separate table and not included in the Course table? (5 pts)

A:

```
CREATE TABLE Course
(
    courseNo   INTEGER PRIMARY KEY,
    courseName VARCHAR(20)
);
CREATE TABLE Course_Prereq
(
    courseNo   INTEGER ,
    prereqNo Integer,
      Foreign Key(prereqNo) references Course (courseNo)
);
```

```
insert into Course values (10,'C++');
insert into Course values (11,'Java');
insert into Course values (12,'Python');
insert into Course values (121,'Web');
insert into Course values (133,'Software Eng');
```

Now, we want to enforce a constraint that *a course cannot have more than 2 prerequsites.*

Write the following trigger to enforce the constraint.

```
CREATE OR REPLACE TRIGGER LimitTest
    BEFORE INSERT OR UPDATE ON Course_Prereq
    FOR EACH ROW  -- A row level trigger
DECLARE
     v_MAX_PREREQS CONSTANT INTEGER := 2;
     v_CurNum INTEGER;
```

```
BEGIN
      BEGIN
            SELECT COUNT(*) INTO v_CurNum FROM Course_Prereq
            WHERE courseNo = :new.CourseNo Group by courseNo;
            EXCEPTION
                  -- Before you enter the first row, no data is found
                  WHEN no_data_found THEN
                   DBMS_OUTPUT.put_line('not found');
                         v_CurNum := 0;
      END;
      if v_curNum > 0 THEN
            IF v_CurNum + 1 > v_MAX_PREREQS THEN
                  RAISE_APPLICATION_ERROR(-20000,'Only  2  prereqs  for
course');
            END IF;
      END IF;
END;
/
SHOW ERRORS;
```

Make sure that the trigger compiles without errors.

Insert the following rows into Course_PreReq table.

```
insert into Course_Prereq values (121,11);

insert into Course_Prereq values (121,10);
```

Using Select, check the data in Course_PreReq table.

Enter the row below (trying to add a third prerequisite for course 121)

```
insert into Course_Prereq values (121,12);
```

Q: Did you successfully add the above row into the table?

A:

Using Select, check the data in Course_PreReq table. How many rows are there?

Enter the row below.

```
insert into Course_Prereq values (133,12);
```

Using Select, check the data in Course_PreReq table. How many rows are there?

Now, do an update as shown below:

```
update COURSE_PREREQ

set courseno = 121 where courseno= 133;
```

Q: What is the result of update above? Did it work? Did you see any mutating table error?

A:

It is possible to insert (single-row inserts), but not update without causing a mutating trigger error.

One solution to this problem is to use Compound Triggers introduced in Oracle 11G.

Question 6 (10 pts)

A compound trigger is a single trigger on a table that enables you to specify actions for each of four timing points:

1.Before the firing statement

2.Before each row that the firing statement affects

3.After each row that the firing statement affects

4.After the firing statement

Let us define a compound trigger to avoid mutating-table error.

Ref: http://www.dbanotes.com/database-development/using-triggers-and-compound-triggers-in-oracle-11g/

Do the following:

Step 1: Delete all rows from Course_Prereq table.

Step 2: Define the following compound trigger.

```
CREATE OR REPLACE TRIGGER LimitTest
FOR INSERT OR UPDATE
ON Course_Prereq
COMPOUND TRIGGER

 /* Declaration Section*/
v_MAX_PREREQS CONSTANT INTEGER := 2;
     v_CurNum INTEGER := 1;
       v_cno INTEGER;

 --ROW level
BEFORE EACH ROW IS
BEGIN
      v_cno := :NEW.COURSENO;
END BEFORE EACH ROW;

 --Statement level
AFTER STATEMENT IS
BEGIN
SELECT COUNT(*) INTO v_CurNum FROM Course_Prereq
            WHERE courseNo = v_cno Group by courseNo;

            IF v_CurNum  > v_MAX_PREREQS THEN
                 RAISE_APPLICATION_ERROR(-20000,'Only 2 prereqs for course');
            END IF;
END AFTER STATEMENT;

 END ;
/
```

```
SHOW ERRORS;
```
Step 3: Compile the error and fix any errors.

Step 4: Insert the following rows.

```
insert into Course_Prereq values (121,11);
insert into Course_Prereq values (121,10);
insert into Course_Prereq values (121,12);
insert into Course_Prereq values (133,12);
```

Step 5: Do a select and display the data in Course_Prereq. Is the constraint, ***a course cannot have more than 2 prerequsites,*** **enforced?**

Step 6: Do an update as shown below.

```
update COURSE_PREREQ

set courseno = 121 where courseno= 133;
```

What is the result? Do a select and display the data in Course_Prereq.

Q: Is the constraint, *a course cannot have more than 2 prerequsites,* enforced?

A: