**\*\*\*\*\*\*\*\*\*\* EXPERIMENT: 08 \*\*\*\*\*\*\*\*\*\***

**\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\***

**Problem Statement:**

Using the relation schemata established in Experiments - 02, 03, and 05, create and execute SQL programs that allow enforcement of business rules with database triggers.

**Author : Mehul Y Khandhadiya**

**Roll no : 55**

**Date : 22-Oct-2020**

**Queries Set**

**\*\*\*\*\*\*\*\*\*\*\*\***

**\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* QUERY – 01 \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\***

**Write SQL code to compile and execute a stored procedure - SHOW\_EMPLOYEE, to list employee details for the input variable ENO holding employee number. (Use EMPP Table)**

**\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\***

**CREATE OR REPLACE TRIGGER UPDATE\_CUST\_BALANCE\_TRG**

**AFTER INSERT ON LINE**

**FOR EACH ROW**

**BEGIN**

**UPDATE CUSTOMER**

**SET BALANCE =: NEW.L\_PRICE \*: NEW.L\_UNITS**

**WHERE C\_CODE = (SELECT C\_CODE FROM INVOICE**

**WHERE INV\_NUM =: NEW.INV\_NUM);**

**END;**

**SELECT \* FROM LINE WHERE INV\_NUM = 1006;**

INV\_NUM L\_NUM P\_COD L\_UNITS L\_PRICE

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

1006 1 MC001 3 6.99

1006 2 JB012 1 109.92

1006 3 CH10X 1 9.95

1006 4 HC100 1 256.99

**SELECT \* FROM INVOICE WHERE INV\_NUM = 1006;**

INV\_NUM C\_CODE INV\_DATE

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

1006 10014 17-JAN-20

**SELECT \* FROM CUSTOMER WHERE C\_CODE = 10014;**

C\_CODE LNAME FNAME C\_AREA C\_PHONE BALANCE

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

10014 Johnson Bill 615 2455533 0

**INSERT INTO LINE VALUES (1006, 5, 'PP101', 10, 5.87);**

1 row created.

**SELECT \* FROM CUSTOMER WHERE C\_CODE = 10014;**

C\_CODE LNAME FNAME C\_AREA C\_PHONE BALANCE

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

10014 Johnson Bill 615 2455533 58.7

**\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* QUERY – 02 \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\***

**Write SQL code to compile and execute a stored procedure - ADD\_EMPLOYEE, to add a record to EMPP table. Check the existence of the created procedure using USER\_OBJECTS view. Use this procedure to insert following records.**

**7118, Your Name, 07-Jul-2020, Teaching Asst., 25000**

**7119, Atulya Bharat, 03-Aug-2005, Professor, 162000**

**\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\***

**CREATE TABLE SALARY\_CHANGES(**

**OP\_TYPE VARCHAR2(10) NOT NULL,**

**OP\_DATE DATE DEFAULT SYSDATE,**

**OP\_TIME VARCHAR2(9) DEFAULT TO\_CHAR(SYSTIMESTAMP,'HH:MI:SS'),**

**OLD\_SAL NUMBER(8,2),**

**NEW\_SAL NUMBER(8,2),**

**ENO NUMBER(4) NOT NULL**

**);**

**CREATE OR REPLACE TRIGGER SALARY\_CHANGE\_TRG**

**AFTER DELETE OR INSERT OR UPDATE ON EMPP**

**FOR EACH ROW**

**BEGIN**

**IF DELETING THEN**

**INSERT INTO SALARY\_CHANGES(OP\_TYPE,OP\_DATE,OP\_TIME,OLD\_SAL,ENO)**

**VALUES ('DELETE',SYSDATE,TO\_CHAR(SYSTIMESTAMP,'HH:MI:SS'),:OLD.SALARY,:OLD.EID);**

**DBMS\_OUTPUT.PUT\_LINE('THE DELETE ENTRY IS LOGGED IN SALARY\_CHANGES TABLE');**

**ELSIF INSERTING THEN**

**INSERT INTO SALARY\_CHANGES(OP\_TYPE,OP\_DATE,OP\_TIME,NEW\_SAL,ENO)**

**VALUES ('INSERT',SYSDATE,TO\_CHAR(SYSTIMESTAMP,'HH:MI:SS'),:NEW.SALARY,:NEW.EID);**

**DBMS\_OUTPUT.PUT\_LINE('THE INSERT ENTRY IS LOGGED IN SALARY\_CHANGES TABLE');**

**ELSE**

**INSERT INTO SALARY\_CHANGES(OP\_TYPE,OP\_DATE,OP\_TIME,OLD\_SAL,NEW\_SAL,ENO)**

**VALUES ('UPDATE',SYSDATE,TO\_CHAR(SYSTIMESTAMP,'HH:MI:SS'),:OLD.SALARY,:NEW.SALARY,:OLD.EID);**

**DBMS\_OUTPUT.PUT\_LINE('THE UPDATE ENTRY IS LOGGED IN SALARY\_CHANGES TABLE');**

**END IF;**

**END;**

Trigger created**.**

**INSERT INTO EMPP**

**VALUES(7121, 'Melody Malvankar', SYSDATE, 'Asst. professor',8000);**

THE INSERT ENTRY IS LOGGED IN SALARY\_CHANGES TABLE

**1 row created.**

**INSERT INTO EMPP**

**VALUES(7122,'Kalpak Gundappa',SYSDATE,'Research Asst.',45000);**

**THE INSERT ENTRY IS LOGGED IN SALARY\_CHANGES TABLE**

**1 row created.**

**UPDATE EMPP SET SALARY=SALARY + 2500 WHERE EID >= 7121;**

THE UPDATE ENTRY IS LOGGED IN SALARY\_CHANGES TABLE

THE UPDATE ENTRY IS LOGGED IN SALARY\_CHANGES TABLE

2 rows updated.

**DELETE FROM EMPP WHERE EID = 7122;**

THE DELETE ENTRY IS LOGGED IN SALARY\_CHANGES TABLE

1 row deleted.

**SELECT COUNT(\*) FROM EMPP;**

COUNT(\*)

----------

20

**SELECT COUNT(\*) FROM SALARY\_CHANGES;**

COUNT(\*)

----------

5

**SELECT \* FROM SALARY\_CHANGES;**

OP\_TYPE OP\_DATE OP\_TIME OLD\_SAL NEW\_SAL ENO

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

INSERT 22-OCT-20 07:48:55 8000 7121

INSERT 22-OCT-20 07:50:48 45000 7122

UPDATE 22-OCT-20 07:51:42 8000 10500 7121

UPDATE 22-OCT-20 07:51:42 45000 47500 7122

DELETE 22-OCT-20 07:52:06 47500 7122

5 rows selected.

**\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* QUERY – 03 \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\***

**Write SQL code to compile and execute the stored procedure - REMOVE EMPLOYEE, which will remove the employee record(s) from EMPP table when supplied with an input name phrase (entered always as lower case) indicating employee name (use EMPP table). If the matching employee is not found, an appropriate exception should be raised.**

**\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\***

**CREATE TABLE EMP\_SALARY**

**AS(SELECT EID,SALARY AS TOT\_SAL FROM EMPP WHERE 1=2);**

Table created.

**SELECT \* FROM EMP\_SALARY;**

no rows selected

**DESC EMP\_SALARY;**

Name Null? Type

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

EID NOT NULL NUMBER(4)

TOT\_SAL NOT NULL NUMBER(8,2)

STATUS VARCHAR2(7)

**INSERT INTO EMP\_SALARY(EID,TOT\_SAL)**

**SELECT EID, (SALARY\*1.25 - 1200)\* 0.90**

**FROM EMPP;**

19 rows created.

**SELECT \* FROM EMP\_SALARY;**

EID TOT\_SAL STATUS

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

7101 167670 ON\_ROLL

7102 163732.5 ON\_ROLL

7103 165420 ON\_ROLL

7104 154620 ON\_ROLL

7105 142245 ON\_ROLL

7106 142245 ON\_ROLL

7107 142245 ON\_ROLL

7108 133582.5 ON\_ROLL

7109 101295 ON\_ROLL

7110 96120 ON\_ROLL

7111 53145 ON\_ROLL

EID TOT\_SAL STATUS

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

7112 49095 ON\_ROLL

7113 38970 ON\_ROLL

7114 35876.25 ON\_ROLL

7115 32670 ON\_ROLL

7116 32670 ON\_ROLL

7117 35145 ON\_ROLL

7118 27045 ON\_ROLL

7119 181170 ON\_ROLL

19 rows selected.

**CREATE OR REPLACE TRIGGER UPDATE\_TOT\_SAL\_TRG**

**AFTER INSERT OR UPDATE OR DELETE ON EMPP**

**FOR EACH ROW**

**BEGIN**

**IF INSERTING THEN**

**INSERT INTO EMP\_SALARY(EID,TOT\_SAL) VALUES(:NEW.EID,(:NEW.SALARY + (:NEW.SALARY\*0.25)-1200)-(:NEW.SALARY + (:NEW.SALARY\*0.25)-1200)\*0.10);**

**ELSIF DELETING THEN**

**UPDATE EMP\_SALARY SET STATUS = 'RETIRED' WHERE EID = :OLD.EID;**

**ELSE**

**UPDATE EMP\_SALARY SET TOT\_SAL = ((:NEW.SALARY + (:NEW.SALARY\*0.25) - 1200)-(:NEW.SALARY + (:NEW.SALARY\*0.25)-1200)\*0.10) WHERE EID = :OLD.EID;**

**END IF;**

**END;**

Trigger created.

**ALTER TRIGGER UPDATE\_TOT\_SAL\_TRG ENABLE;**

Trigger altered.

**COMMIT;**

Commit complete.

SELECT COUNT(\*) FROM EMPP;

COUNT(\*)

----------

19

**SELECT COUNT(\*) FROM EMP\_SALARY;**

COUNT(\*)

----------

19

**INSERT INTO EMPP VALUES(7121,'Melody malvankar',SYSDATE,'Asst. professor',80000);**

1 row created.

**INSERT INTO EMPP VALUES(7122,'kalpak Gundappa',SYSDATE,'Research Asst.',45000);**

1 row created.

**UPDATE EMPP SET SALARY=SALARY + 2500 WHERE EID>=7121;**

2 rows updated.

**DELETE FROM EMPP WHERE ENO = 7122;**

DELETE FROM EMPP WHERE ENO = 7122

\*

ERROR at line 1:

ORA-00904: "ENO": invalid identifier

**DELETE FROM EMPP WHERE EID = 7122;**

**1 row deleted.**

**SQL> SELECT COUNT(\*) FROM EMP\_SALARY;**

**COUNT(\*)**

**----------**

**21**

**SQL> SELECT COUNT(\*) FROM EMPP;**

**COUNT(\*)**

**----------**

**20**

**SQL> SELECT \* FROM EMP\_SALARY;**

**EID TOT\_SAL STATUS**

**---------- ---------- -------**

**7101 167670 ON\_ROLL**

**7102 163732.5 ON\_ROLL**

**7103 165420 ON\_ROLL**

**7104 154620 ON\_ROLL**

**7105 142245 ON\_ROLL**

**7106 142245 ON\_ROLL**

**7107 142245 ON\_ROLL**

**7108 133582.5 ON\_ROLL**

**7109 101295 ON\_ROLL**

**7110 96120 ON\_ROLL**

**7111 53145 ON\_ROLL**

**EID TOT\_SAL STATUS**

**---------- ---------- -------**

**7112 49095 ON\_ROLL**

**7113 38970 ON\_ROLL**

**7114 35876.25 ON\_ROLL**

**7115 32670 ON\_ROLL**

**7116 32670 ON\_ROLL**

**7117 35145 ON\_ROLL**

**7118 27045 ON\_ROLL**

**7119 181170 ON\_ROLL**

**7121 91732.5 ON\_ROLL**

**7122 52357.5 RETIRED**

**21 rows selected.**

**\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* QUERY – 04 \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\***

**Write SQL code to compile and execute the stored function - CHECK\_ITEM that will report status as 1 if items with mentioned P\_CODE are present in the inventory, otherwise reports status as 0. No exceptions to be handled.**

**\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\***

**\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* QUERY – 05 \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\***

**Write a SQL code to compile and execute the stored procedure - ADDITEM, that will insert an item in ITEMS table with given particulars - item code, item description, invoice date, quantity of purchase, minimum quantity, item price and supplier code.**

**\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\***

**\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* QUERY – 06 \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\***

**Write a SQL code to compile and execute the stored procedure - UPDATE\_ITEM, that will update particulars (quantity and/or cost) for an item in ITEMS table with given particulars - item code, quantity of purchase, and item price.**

**Report an error when the said item (to be updated) does not exist in ITEMS table (the NO\_DATA\_FOUND exception). Use the CHECK\_ITEM function created earlier.**

**\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\***

**\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* QUERY – 07 \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\***

**Modify procedure in Query-06, as UPDATE\_ITEM\_ADD\_WHEN\_NOT\_FOUND such that when the mentioned item is not present in ITEMS, an item is entered into ITEMS with available particulars supplied in the procedure call.**

**The default values for item description, vendor code and minimum quantity as 'NEW ITEM ...', NULL and (quantity / 8) truncated respectively. Use ADD\_ITEM procedure created earlier.**

**You need not catch the NO\_DATA\_FOUND exception.**

**\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\***

**\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* QUERY – 08 \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\***

**Write a SQL code to compile and execute the stored procedure - SHOW ITEM that will list the item particulars for an item in ITEMS table when the item code is supplied as input.**

**Report an error when the said item to be updated does not exist in ITEMS. Use the CHECK\_ITEM function created earlier.**

**\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\***

**\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* QUERY – 09 \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\***

**Modify the procedure in Query-08 as SHOW\_ITEM\_TMR\_E which will handle TOO\_MANY\_ROWS exception in SELECT query.**

**In addition to exceptions in Query-06 (NO\_DATA\_FOUND and OTHERS) the TOO\_MANY\_ROWS exception should be caught when a call to the procedure call -EXEC ADD\_ITEM( HH15P', 'NEW ITEM-2' ,150,NULL,25); fetches more than one row in the result set.**

**\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\***

**\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* QUERY – 10 \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\***

**Now extend the procedure in Query-09 as SHOW\_ITEM\_TMR\_HANDL ED to print the rows returned by the SELECT query after catching the appropriate exception.number from console. You should only report the violations.**

**\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\***

**\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* VIVA-VOCE \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\***

**Q1 – Differentiate b/w a statement level trigger and a row**

**level trigger.**

**Statement level triggers** executes only once for each single transaction. Used for enforcing all additional security on the transactions performed on the table. “FOR EACH ROW” clause is omitted in CREATE TRIGGER command.

Example: If 1500 rows are to be inserted into a table, the statement level trigger would execute only once.

**Row level triggers** executes once for each and every row in the transaction. Specifically used for data auditing purpose. “FOR EACH ROW” clause is present in CREATE TRIGGER command.

Example: If 1500 rows are to be inserted into a table, the row level trigger would execute 1500 times.

**Q2 – How many triggers a table can have? List all of these.**

We can define 6 types of triggers for each table:

1. **AFTER INSERT** activated after data is inserted into the table.
2. **AFTER UPDATE:** activated after data in the table is modified.
3. **AFTER DELETE:** activated after data is deleted/removed from the table.
4. **BEFORE INSERT:** activated before data is inserted into the table.
5. **BEFORE UPDATE:** activated before data in the table is modified.
6. **BEFORE DELETE:** activated before data is deleted/removed from the table.

**Q3 – What are cascading triggers?**

A Trigger that contains statements which cause invoking of other Triggers are known as cascading triggers. Here’s the order of execution of statements in case of cascading triggers:

* Execute all BEFORE statement triggers that apply to the current statement.
* Loop for each row affected statement.
* Execute all BEFORE row triggers that apply to the current statement in the loop.
* Lock and change row, perform integrity constraints check; release lock.
* Execute all AFTER row triggers that apply to the current statement.
* Execute all AFTER statement triggers that apply to the current statement.

**Q4 - Why COMMIT and ROLLBACK cannot be used in triggers? Can a**

**trigger call a stored function or procedures that perform a**

**COMMIT or a ROLLBACK?**

A transaction is everything that happens between one COMMIT/ROLLBACK and the next. Every DDL issues two implicit commands. Doing this in a trigger would break the transaction into nested transactions which doesn't make sense. Autonomous transactions are a way of implementing nested transactions but they should be used with care.

You cannot include COMMIT or ROLLBACK statements in a stored procedure if any of the following conditions are true:

* The stored procedure is nested within a trigger or user-defined function.
* The stored procedure is called by a client that uses two-phase commit processing.

**Q4 – Is it possible to create a trigger that will fire when a row**

**is reading a query?**

Row-level triggers fires once for each row affected by the triggering event such as INSERT, UPDATE, or DELETE. A row trigger is fired each time the table is affected by the triggering statement.

Row-level triggers are useful for data-related activities such as data auditing and data validation. A row-level trigger fires each time a row is affected by a triggering event.

**INFERENCE:** From this expt., we learnt about triggers in PL/SQL. We learn about its uses and its implementation. We learnt statement level trigger, row level trigger, etc and their different operations. We implemented these triggers in queries listed and the results were recorded and analysed.

**\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* END \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\***