



**Islamic University of Technology**  
**Lab Task 10**  
**CSE 4308 - DBMS Lab**

**Submitted To :**

MD Bakhtiar Hasan  
Ast. Professor, CSE Department  
Islamic University of Technology

Zannatun Naim Sristy  
Lecturer, CSE Department  
Islamic University of Technology

**Submitted By :**

Sian Ashsad  
ID : 200042151  
Prog. : SWE  
Dept. : CSE

## **Task 1 :**

### **Working code :**

```
DECLARE
    TOTAL_ROWS NUMBER (2);
BEGIN
    UPDATE INSTRUCTOR
        SET SALARY = SALARY + (SALARY * 0.1)
        WHERE SALARY < 75000 ;
    IF SQL % NOTFOUND THEN
        DBMS_OUTPUT . PUT_LINE ( 'No instructor satisfied the condition ');
    ELSIF SQL % FOUND THEN
        TOTAL_ROWS := SQL % ROWCOUNT ;
        DBMS_OUTPUT . PUT_LINE ( TOTAL_ROWS || ' instructors incremented ');
    END IF;
END ;
/
```

### **Report :**

**Analysis :** In this task we need to write PL/SQL statements to provide a 10% increment to the instructors that get a salary less than 75000 and then show the number of instructors that got increment.

**Explanation of solution :** A NUMBER variable TOTAL\_ROWS is declared at the very beginning of the code. An SQL query is performed which updates the Instructor table and sets the salary by incrementing it by 10% if Salary is less than 75000.

Afterwards IF ELSIF conditionals are set to check if such instructors are found or not. This is done through the NOTFOUND and FOUND implicit cursors. If the SQL query affects at least one row, then the ROWCOUNT function is used to see how many were affected. The count is stored in the TOTAL\_ROWS variable and output in the console. This is the number of instructors that got incremented.

**Findings :** The method of finding if any rows were affected or not was new to me. The usage of the ROWCOUNT function made counting affected rows simple.

**Problems :** At first, I implemented a query which counted the number of instructors whose salary was less than 75000 and showed that to the console. This required using two SQL queries instead of one. The order of these SQL queries could also affect the result. Therefore using cursors instead is the wiser implementation choice.

## **Task 2 :**

### **Working code :**

```
CREATE OR REPLACE
PROCEDURE PRINT_TIME_SLOT
AS
BEGIN

    FOR i IN (SELECT T.TIME_SLOT_ID, T.DAY, T.start_hr, T.start_min ,T.end_hr,
T.end_min FROM INSTRUCTOR I, TEACHES E, SECTION S,TIME_SLOT T WHERE I.ID = E.ID
AND

                E.COURSE_ID = S.COURSE_ID AND E.SEC_ID = S.SEC_ID AND
E.SEMESTER = S.SEMESTER AND E.YEAR = S.YEAR AND
                S.TIME_SLOT_ID = T.TIME_SLOT_ID) LOOP
        DBMS_OUTPUT . PUT_LINE (i.TIME_SLOT_ID || ' ' || i.DAY || ' ' ||
i.start_hr || ' ' || i.end_hr);
    END LOOP;

END;
/
```

### **Report :**

**Analysis :** In this task we need to write PL/SQL statements to create a procedure for printing time\_slot of every teacher.

**Explanation of solution :** The procedure performs an SQL query between four tables. The relations between these tables are self-explanatory. The query is passed through a loop which iterates through each row provided by the query. Then required output is shown in the console.

**Findings :** The same procedure can be made using cursors. However I opted to use simple loop iterations instead.

**Problems :** There was a problem regarding the relationship between the given tables. Natural join was not working. The issue was that the table was different from the diagram provided in the lab task. After finding out appropriate relations the problem was resolved.

### Task 3 :

#### Working code :

```
CREATE OR REPLACE
PROCEDURE FIND_ADVISORS(NUM IN NUMBER)
AS
ROW NUMBER(5);
BEGIN
    SELECT MAX(ROWNUM) INTO ROW
    FROM (SELECT I_ID, COUNT(S_ID) AS S_COUNT FROM ADVISOR GROUP BY I_ID ORDER BY
S_COUNT DESC);

    IF(NUM>ROW) THEN
        DBMS_OUTPUT . PUT_LINE ('Input exceeds number of entries');
        RETURN;
    END IF;

    FOR i IN (SELECT * FROM (SELECT I_ID, COUNT(S_ID) AS S_COUNT FROM ADVISOR
GROUP BY I_ID ORDER BY S_COUNT DESC) WHERE ROWNUM<=NUM) LOOP
        DBMS_OUTPUT . PUT_LINE (i.I_ID || ' ' || i.S_COUNT);
    END LOOP;

END;
/
```

#### Report :

**Analysis :** Using PL/SQL statements we need to find the N advisers and their details who has the highest number of students under their advising.

**Explanation of solution :** A procedure `FIND_ADVISORS` is created which takes in a Number variable `NUM`. A number variable `ROW` is declared at the beginning of the procedure. The first SQL query gives the number of Instructors into the `ROW` variable. The query also provides the list of Instructors with the number of students under them in a descending order.

An IF statement checks whether the provided `NUM` variable is bigger than `ROW`. If so then an error message is displayed and the procedure is terminated. Otherwise, a for loop is used to iterate through the previously described SQL query and show the records of `NUM` instructors.

## **Task 4 :**

### **Working code :**

```
CREATE SEQUENCE STUDENT_SEQ
MINVALUE 1
MAXVALUE 9999
START WITH 1
INCREMENT BY 1
CACHE 20;

CREATE OR REPLACE
TRIGGER STUDENT_ID_GENERATOR
BEFORE INSERT ON STUDENT
FOR EACH ROW
BEGIN
    :NEW.ID := STUDENT_SEQ . NEXTVAL ;
END ;
/
```

### **Report :**

**Analysis :** Using PL/SQL we need to create a trigger that automatically generates IDs for students when data is inserted into the `STUDENT` table.

**Explanation of solution :** A sequence is created (**STUDENT\_SEQ**) which has the MINVALUE of 1 and MAXVALUE of 9999. The sequence starts with 1 and increments by 1. It has a cache value of 20.

A trigger **STUDENT\_ID\_GENERATOR** is created which activates before an insert operation on the student table. This trigger replaces the ID value of the insert values with the incremented value of the **STUDENT\_SEQ** sequence.

**Findings :** **:NEW** is used to access the values of the input. In oracle 11g, it is possible to directly put the incremented value of the sequence into the input value.

## **Task 5 :**

### **Working code :**

```
CREATE OR REPLACE
  TRIGGER ADVISOR_ASSIGNER
  AFTER INSERT ON STUDENT
  FOR EACH ROW
DECLARE
  INS_ID INSTRUCTOR.ID% TYPE ;
BEGIN
  SELECT ID INTO INS_ID
  FROM(
    SELECT ID
    FROM INSTRUCTOR I
    WHERE I.DEPT_NAME = :NEW.DEPT_NAME
  )
  WHERE ROWNUM<=1;
  INSERT INTO ADVISOR VALUES(INS_ID, :NEW.ID);

END ;
/
```

### **Report :**

**Analysis :** In this task, we need to create a trigger that will automatically assign an advisor to a newly admitted student of his/her own department.

**Explanation of solution :** A trigger `ADVISOR_ASSIGNER` is created which takes effect after an insertion on the student table. A variable `INS_ID` is taken of the same type as `INSTRUCTOR.ID`.

An SQL query provides the ID of the first instructor found on the Instructor table whose department matches that of the given student's department. `:NEW` is used to determine the newly provided student information. This ID is stored into the `INS_ID` variable. Finally the student id and `INS_ID` is inserted into the ADVISOR table.

**Findings :** The usage of `:NEW` keyword to get the values of newly entered student information is quite useful for the given task. The code can only assign the first instructor of the same department to a new student. Therefore other instructors after the first one of each department are not assigned automatically.

**Problems :** There were no problems faced during this task.