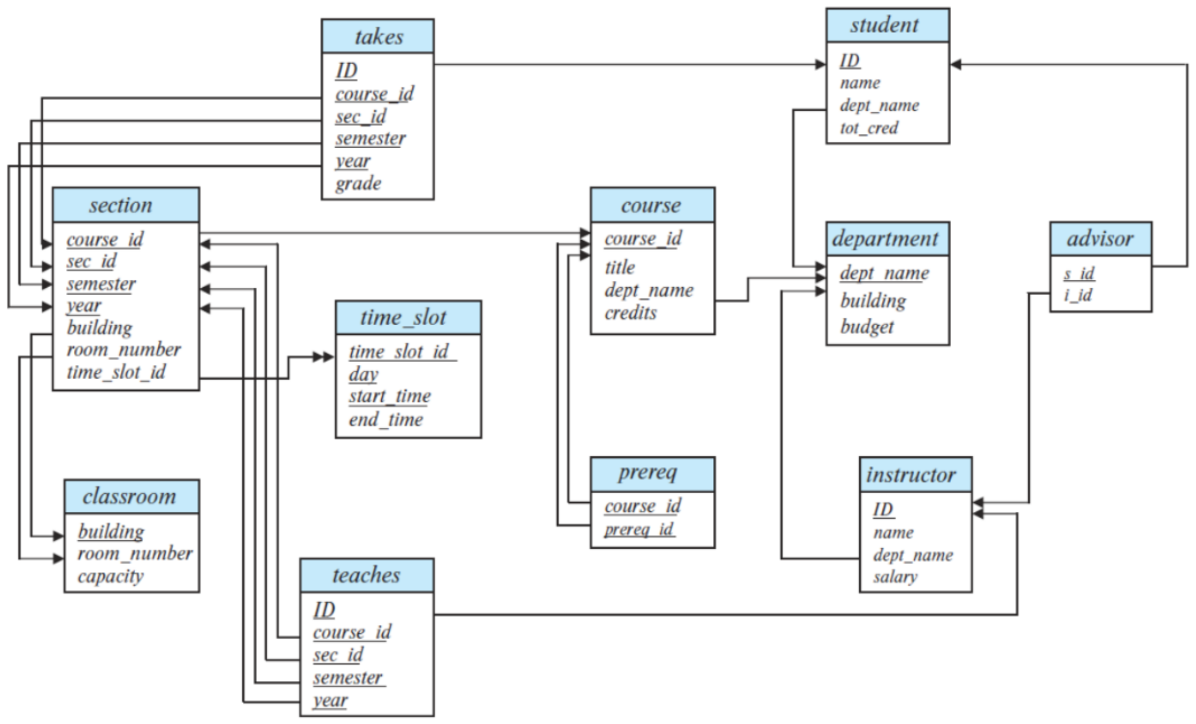
Lab 10 Report

Name	: Mashrur Ahsan
ID	: 200042115
Program	: SWE
Department	: CSE
Course	: CSE 4308

The Scenario:

Consider the following schema of an University Database:



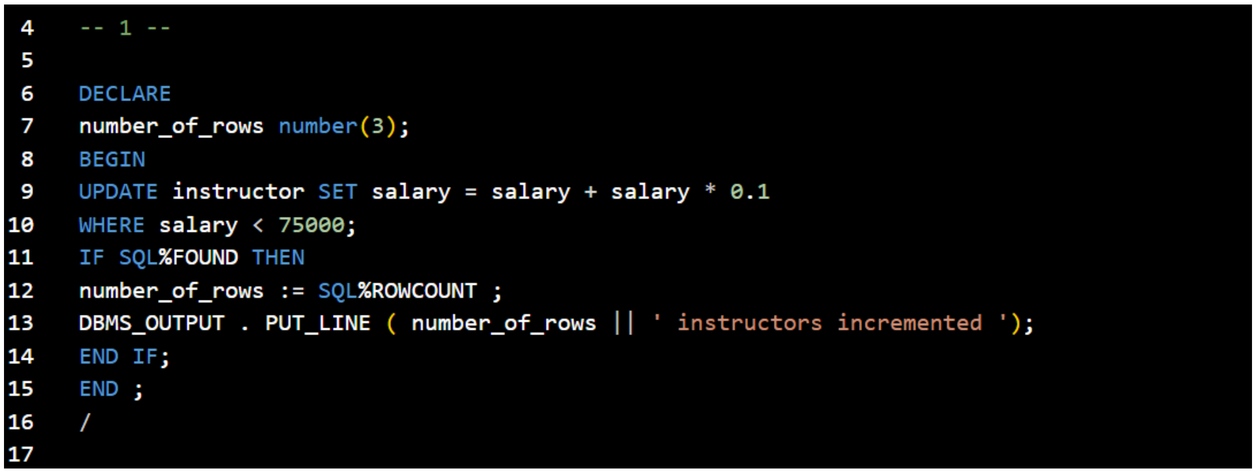
Problem Statements:

Execute the given `DDL+drop.sql` and `smallRelationsInsertFile.sql` files. Then, write PL/SQL statements to perform the following tasks:

1. Provide 10% increment to the instructors that get salary less than 75000. Show the number of instructors that got increment.
2. Write a procedure for printing time_slot of every teacher.
3. Write a procedure to find the N advisers and their details who has highest number of students under their advising.
4. Create a trigger that automatically generates IDs for students when we insert data into `STUDENT` table.
5. Create a trigger that will automatically assign a advisor to a newly admitted student of his/her own department.

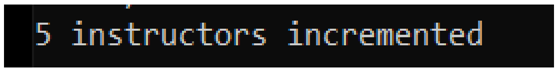
Write anonymous blocks to illustrate your programs, if needed.

1. Provide 10% increment to the instructors that get salary less than 75000. Show the number of instructors that got increment.



```
4    -- 1 --
5
6    DECLARE
7    number_of_rows number(3);
8    BEGIN
9    UPDATE instructor SET salary = salary + salary * 0.1
10   WHERE salary < 75000;
11   IF SQL%FOUND THEN
12   number_of_rows := SQL%ROWCOUNT ;
13   DBMS_OUTPUT . PUT_LINE ( number_of_rows || ' instructors incremented ');
14   END IF;
15   END ;
16   /
17
```

The output:



```
5 instructors incremented
```

Explanation:

Line 9: Updating salary.

Line 11: Use of keywords.

Line 12: Assigning the number of rows affected into a variable.

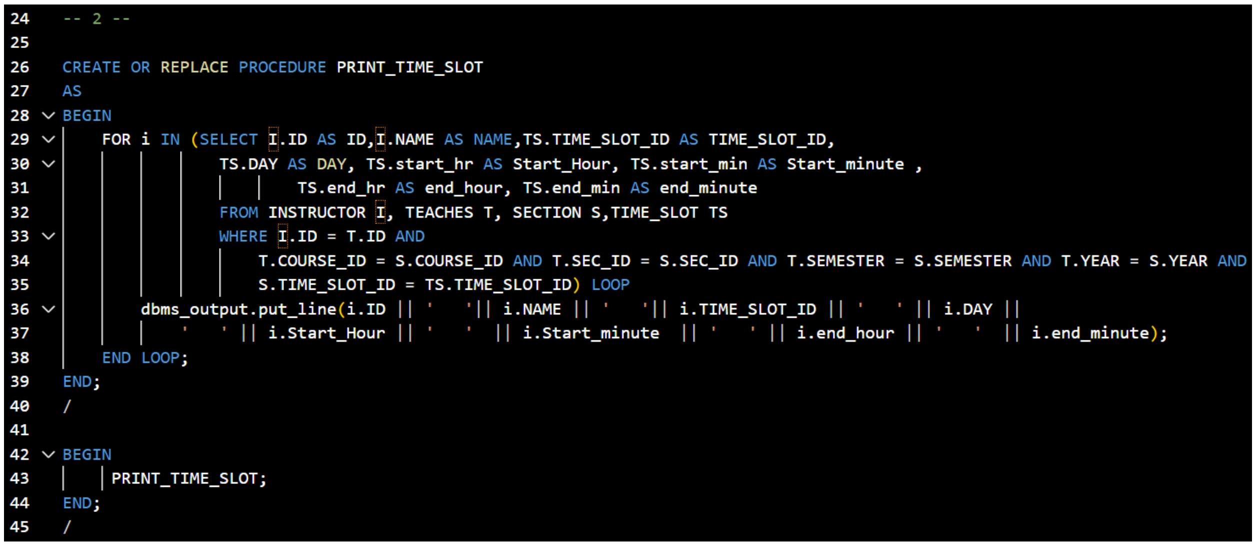
Findings:

- Understood how to use implicit cursors.

Problems:

- Nothing to significant.

2. Write a procedure for printing time_slot of every teacher.



```
24   -- 2 --
25
26   CREATE OR REPLACE PROCEDURE PRINT_TIME_SLOT
27   AS
28   BEGIN
29       FOR i IN (SELECT I.ID AS ID,I.NAME AS NAME,TS.TIME_SLOT_ID AS TIME_SLOT_ID,
30                   TS.DAY AS DAY, TS.start_hr AS Start_Hour, TS.start_min AS Start_minute ,
31                           TS.end_hr AS end_hour, TS.end_min AS end_minute
32                   FROM INSTRUCTOR I, TEACHES T, SECTION S,TIME_SLOT TS
33                   WHERE I.ID = T.ID AND
34                       T.COURSE_ID = S.COURSE_ID AND T.SEC_ID = S.SEC_ID AND T.SEMESTER = S.SEMESTER AND T.YEAR = S.YEAR AND
35                       S.TIME_SLOT_ID = TS.TIME_SLOT_ID) LOOP
36           dbms_output.put_line(i.ID || '   '|| i.NAME || '   '|| i.TIME_SLOT_ID || '   ' || i.DAY ||
37               '   ' || i.Start_Hour || '   ' || i.Start_minute  || '   ' || i.end_hour || '   ' || i.end_minute);
38       END LOOP;
39   END;
40   /
41
42   BEGIN
43       PRINT_TIME_SLOT;
44   END;
45   /
```

Explanation:

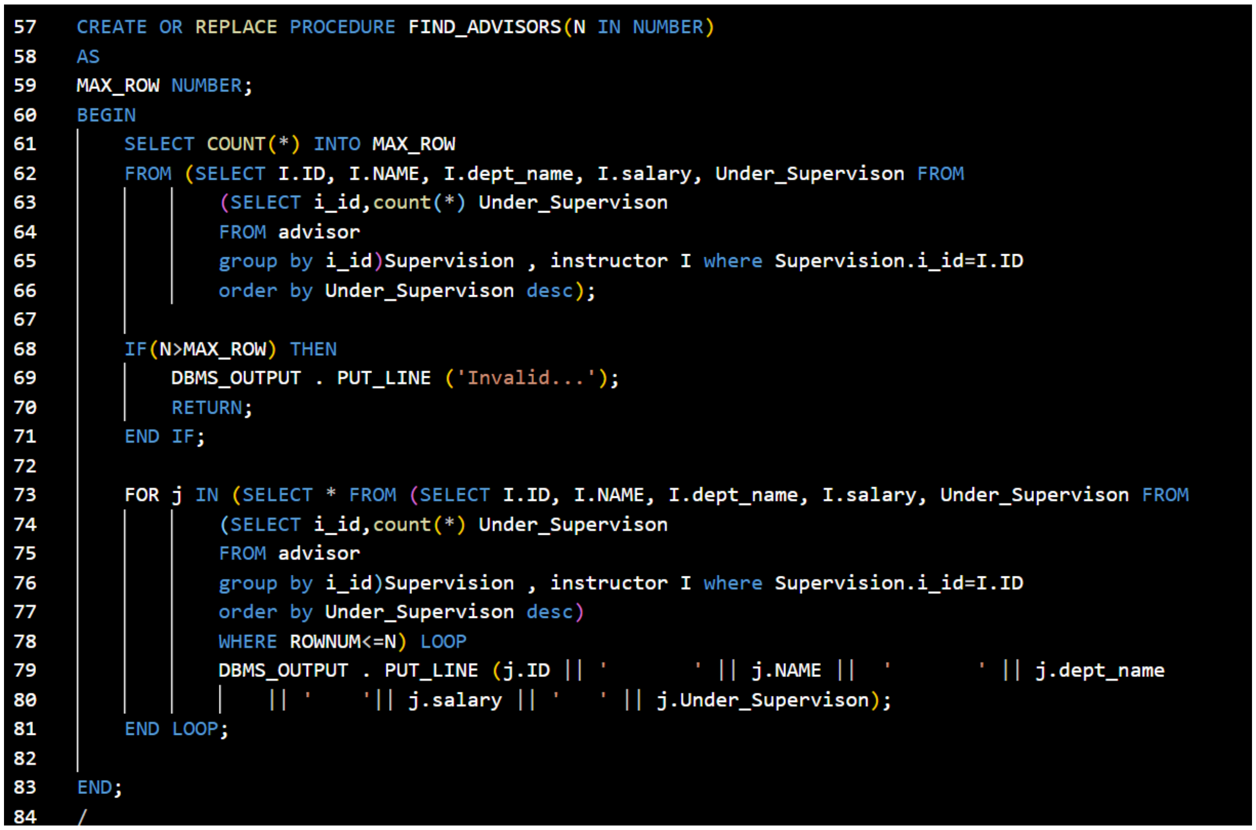
Line 29 to 35: Basically, it's the whole query. Selecting the things we want to show and selecting the tables that we want to show from. Then we are making sure that the foreign key constraints are followed.

Line 36: Printing the desired output.

Findings and Problems:

- Nothing too significant.

3. Write a procedure to find the N advisers and their details who has highest number of students under their advising.



```
57  CREATE OR REPLACE PROCEDURE FIND_ADVISORS(N IN NUMBER)
58  AS
59  MAX_ROW NUMBER;
60  BEGIN
61      SELECT COUNT(*) INTO MAX_ROW
62      FROM (SELECT I.ID, I.NAME, I.dept_name, I.salary, Under_Supervison FROM
63              (SELECT i_id,count(*) Under_Supervison
64              FROM advisor
65              group by i_id)Supervision , instructor I where Supervision.i_id=I.ID
66              order by Under_Supervison desc);
67
68      IF(N>MAX_ROW) THEN
69          DBMS_OUTPUT . PUT_LINE ('Invalid...');
70          RETURN;
71      END IF;
72
73      FOR j IN (SELECT * FROM (SELECT I.ID, I.NAME, I.dept_name, I.salary, Under_Supervison FROM
74              (SELECT i_id,count(*) Under_Supervison
75              FROM advisor
76              group by i_id)Supervision , instructor I where Supervision.i_id=I.ID
77              order by Under_Supervison desc)
78              WHERE ROWNUM<=N) LOOP
79              DBMS_OUTPUT . PUT_LINE (j.ID || '        ' || j.NAME ||  '        ' || j.dept_name
80                  || '    '|| j.salary || '   ' || j.Under_Supervison);
81      END LOOP;
82
83  END;
84  /
```

Explanation:

Line 62 to 66: This is the main query. The query gives us the details of every instructor that is acting as a supervisor for any number of students.

Line 68: If the query produces less rows than the input N, then it will print out “Invalid”.

Line 73 to 81: Iterating a loop where it is printing out details until we reach N.

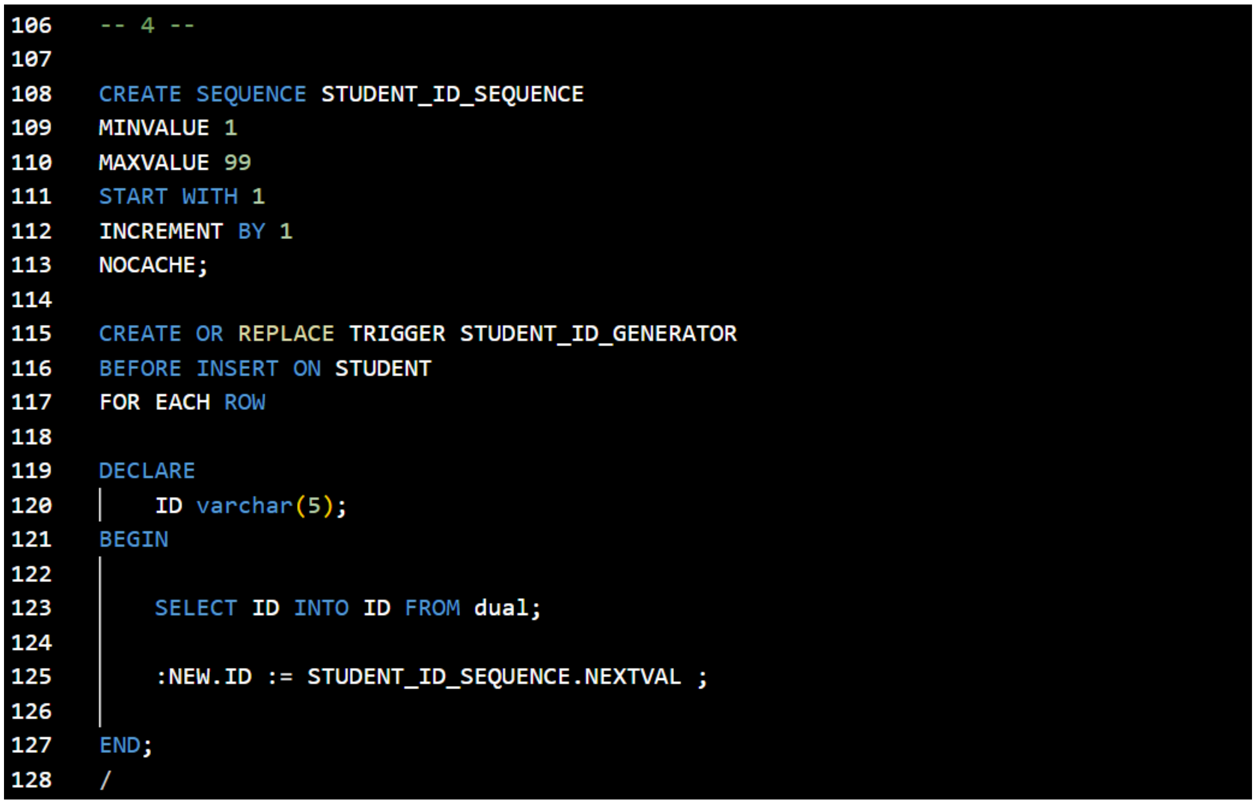
Findings:

- How to iterate a loop for a massive query that has multiple columns of information.
- How to fetch a single row from a massive query into a variable.

Problems:

- The resulting query was a bit difficult to figure out.

4. Create a trigger that automatically generates IDs for students when we insert data into STUDENT table.



```
106    -- 4 --
107
108    CREATE SEQUENCE STUDENT_ID_SEQUENCE
109    MINVALUE 1
110    MAXVALUE 99
111    START WITH 1
112    INCREMENT BY 1
113    NOCACHE;
114
115    CREATE OR REPLACE TRIGGER STUDENT_ID_GENERATOR
116    BEFORE INSERT ON STUDENT
117    FOR EACH ROW
118
119    DECLARE
120        ID varchar(5);
121    BEGIN
122
123        SELECT ID INTO ID FROM dual;
124
125        :NEW.ID := STUDENT_ID_SEQUENCE.NEXTVAL ;
126
127    END;
128    /
```

Explanation:

Line 123: Fetched information and stored it in a variable.

Line 125: Assigning the new value of ID according to the 'SEQUENCE' that was created earlier.

Findings:

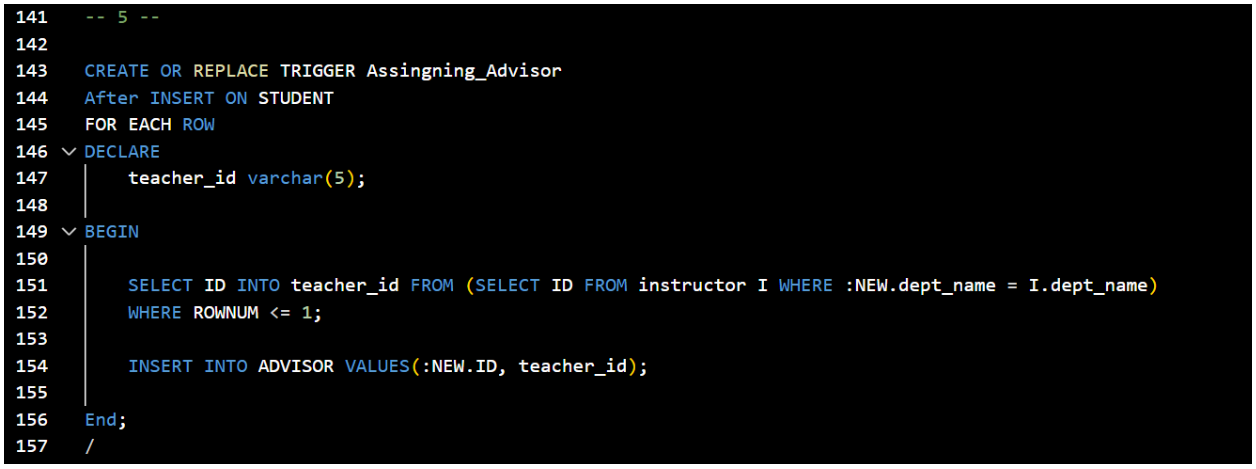
- Understood what a trigger is and understood how a "Before INSERT" trigger works.
- Learned about "SEQUENCE" in pl/sql and what its keywords represent and how it works as a whole.
- What a dual is and how it works.

Problems:

- Had some problem figuring out how the sequence would work with the trigger.
- In general, faced several problems like one would face when one's implementing a trigger for the first time.

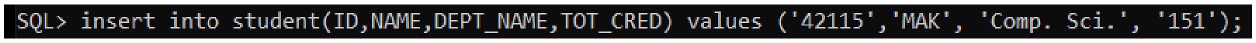
5. Create a trigger that will automatically assign a advisor to a newly admitted student of his/her own department.

Solution:



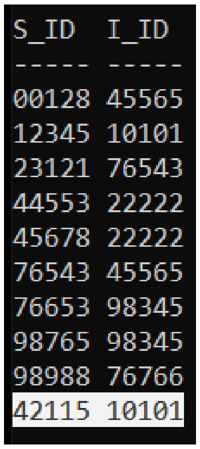
```
141   -- 5 --
142
143   CREATE OR REPLACE TRIGGER Assingning_Advisor
144   After INSERT ON STUDENT
145   FOR EACH ROW
146   DECLARE
147       teacher_id varchar(5);
148
149   BEGIN
150
151       SELECT ID INTO teacher_id FROM (SELECT ID FROM instructor I WHERE :NEW.dept_name = I.dept_name)
152       WHERE ROWNUM <= 1;
153
154       INSERT INTO ADVISOR VALUES(:NEW.ID, teacher_id);
155
156   End;
157   /
```

Adding a new row:



```
SQL> insert into student(ID,NAME,DEPT_NAME,TOT_CRED) values ('42115','MAK', 'Comp. Sci.', '151');
```

After adding the new row:



```
S_ID  I_ID
----- -----
00128 45565
12345 10101
23121 76543
44553 22222
45678 22222
76543 45565
76653 98345
98765 98345
98988 76766
42115 10101
```

Explanation:

Line 151: Fetching the first ID into a variable where the student’s department is the same as that of the instructor.

Line 154: Inserting values into the Advisor table.

Findings:

- Understood how to use After insert trigger.
- Row-level trigger.

Problems:

- Didn’t know how to properly work with new values in case of “AFTER INSERT” triggers.