Islamic University of Technology (IUT)

Report on Lab 10

Submitted By

Shanta Maria (ID:200042172)

CSE 4308 Database Management Systems Lab

Submitted To

Md. Bakhtiar Hasan

Lecturer, Department of CSE

Zannatun Naim Srsity

Lecturer, Department of CSE

November 13, 2022

Introduction

In the lab class, we were given five tasks to solve using PL/SQL to understand the use of procedures and triggers.

1 Task 1

1. Decrease the budget of the departments having a budget more than 99999 by 10%. Show the number of departments that did not get affected.

1.1 Solution

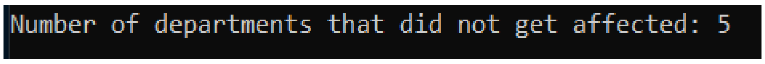
```
-----Task 1-----
declare
    total_rows number(2);
    rows_updated number(2);
begin
    update department
        set budget=budget - (0.10*budget)
            where budget>99999;

    rows_updated := sql%rowcount;

    select count(*) into total_rows from department;

    dbms_output.put_line('Number of departments that did not get affected: '
    || to_char(total_rows-rows_updated));
end;
/
```

1.2 Analysis and Explanation

First two variables were declared for the total number of rows and the number of rows being updated. Then after `begin` statement, I wrote the update `SQL` statement which decreases the budget by 10% for those above 99999. The next two lines of code store the number of rows where budget is updated and then the total number of rows from the table `department` in the variables declared before. Lastly, the difference between the two variables is printed as output for the number of unaffected rows.

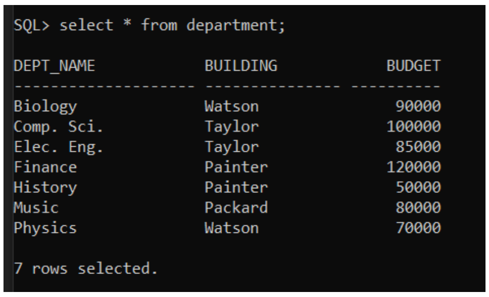
1.3 Difficulties

I faced a lot of errors and had to do many trials and errors before getting the result I wanted. The `sql%rowcount` code returned wrong results if it was not written right after the update statement.



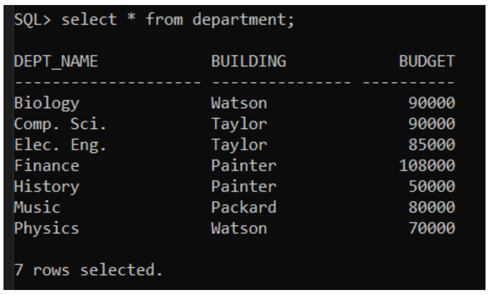
```
Number of departments that did not get affected: 5
```

Figure 1: Task 1



```
SQL> select * from department;

DEPT_NAME            BUILDING             BUDGET
-------------------- --------------- ----------
Biology              Watson                90000
Comp. Sci.           Taylor               100000
Elec. Eng.           Taylor                85000
Finance              Painter              120000
History              Painter               50000
Music                Packard               80000
Physics              Watson                70000

7 rows selected.
```

Figure 2: Before Update



```
SQL> select * from department;

DEPT_NAME            BUILDING             BUDGET
-------------------- --------------- ----------
Biology              Watson                90000
Comp. Sci.           Taylor                90000
Elec. Eng.           Taylor                85000
Finance              Painter              108000
History              Painter               50000
Music                Packard               80000
Physics              Watson                70000

7 rows selected.
```

Figure 3: After Update

2 Task 2

2. Write a procedure that takes the day of the week and starting hour and ending hour as input and prints which instructors should be in the class during that time.

2.1 Solution

```
-----Task 2-----
-----procedure-----
create or replace
procedure instructor_schedule
(free_day in time_slot.day%type, s_hr in time_slot.start_hr%type,
e_hr in time_slot.end_hr%type)
is
    ins_name instructor.name%type;
    ins_id instructor.id%type;

    cursor ins(free_day time_slot.day%type, s_hr time_slot.start_hr%type,
    e_hr time_slot.end_hr%type)
    is
        select id, name
        from instructor natural join teaches natural join section
        natural join time_slot
        where day=free_day and start_hr=s_hr and end_hr=e_hr;

begin
    open ins(free_day, s_hr, e_hr);
    loop
        fetch ins into ins_id, ins_name;
        exit when ins%notfound;
        dbms_output.put_line(ins_id || ' ' || ins_name);
    end loop;
    close ins;
end;
```

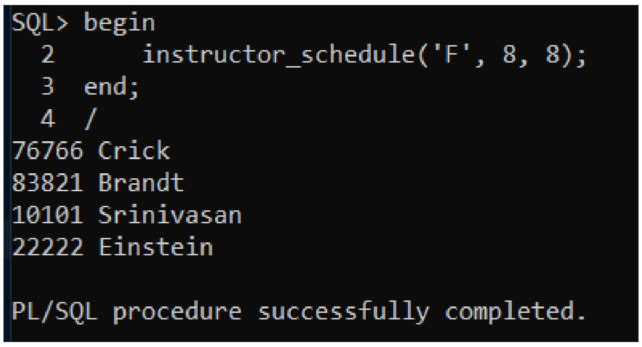
```
/


-----call-----
begin
    instructor_schedule('F', 8, 8);
end;
/
```

2.2 Analysis and Explanation

Firstly, I declared variables to store cursor results later on in the code. Then I made a cursor called `ins` which took the same parameters as the procedure and returned the ID and Name of the instructors who were free during the specified time. This was found by the natural join of four tables as shown in the code. Then, I ran a loop to traverse through the cursor results. I stored each of the column values into the variables declared before and printed the results using `dbms_output.put_line()`. The loop was ended when there were no more results/rows returned by the cursor. To demonstrate if the procedure worked correctly, I called the procedure using the inputs `'F', 8, 8` which showed the result shown below.

2.3 Difficulties

I found this task very difficult. It took me a long while to complete this task because it was confusing for me to understand that the question asked for the teachers who should be teaching class in the time period specified and not the teachers who were free during that time slot and should be assigned a class to fit that. It was also difficult to debug my code for the procedure even with `show error`.



```
SQL> begin
  2      instructor_schedule('F', 8, 8);
  3  end;
  4  /
76766 Crick
83821 Brandt
10101 Srinivasan
22222 Einstein

PL/SQL procedure successfully completed.
```

Figure 4: Task 2

3 Task 3

3. Write a procedure to find the top *N* students based on the number of courses they are enrolled in. The procedure should take *N* as input and print the ID, name, department name, and the number of courses taken by the student.

3.1 Solution

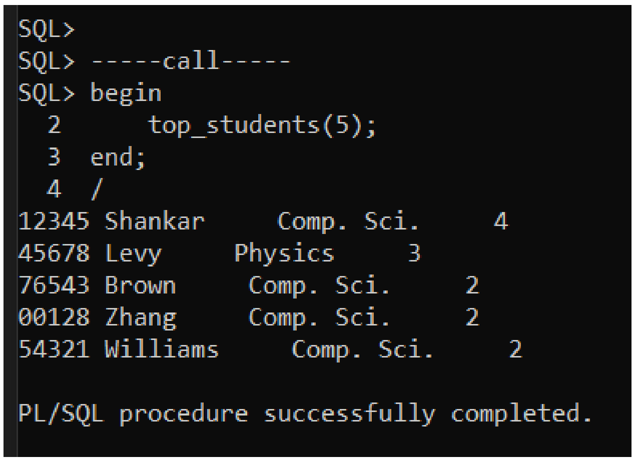
```
-----Task 3-----

-----procedure-----

create or replace

procedure top_students(n in int)

is

    std_id student.id%type;

    std_name student.name%type;

    std_dept_name student.dept_name%type;

    std_course_num int;


    cursor find_std

    is

        select id, max(name) as name, max(dept_name) as dept_name,

        count(course_id) as no_of_courses

        from student natural join takes
        group by id
        order by no_of_courses desc;

begin
    open find_std;
    loop
        fetch find_std into std_id, std_name, std_dept_name,
        std_course_num;
        exit when find_std%rowcount>n;
        dbms_output.put_line(std_id || ' ' || std_name || '     '
        || std_dept_name || '     ' || std_course_num);
    end loop;
    close find_std;
end;
/

-----call-----
begin
    top_students(5);
end;
/
```

3.2 Analysis and Explanation

This task was similar to task 2 so it was easier to solve. As before, I wrote the cursor which fetched the results using SQL query. The cursor was then traversed using a loop. The column results were then stored in separate variables and printed. The loop was exited when the number of rows traversed from the cursor was greater than the number of students specified by the procedure input. Lastly, I tested the procedure by calling it. The result is shown below.

3.3 Difficulties

I found this task easy to solve since this was similar to Task 2. I still found it a bit difficult to debug my code even using `show error` statement.



```
SQL>
SQL> -----call-----
SQL> begin
  2      top_students(5);
  3  end;
  4  /
12345 Shankar     Comp. Sci.     4
45678 Levy     Physics     3
76543 Brown     Comp. Sci.     2
00128 Zhang     Comp. Sci.     2
54321 Williams     Comp. Sci.     2

PL/SQL procedure successfully completed.
```

Figure 5: Task 3

4 Task 4

4. Create a trigger that automatically generates IDs for instructors when we insert data into `INSTRUCTOR` table.

4.1 Solution

```
-----Task 4-----
-----trigger-----
drop sequence ins_seq;
create sequence ins_seq
minvalue 10000
maxvalue 99999
start with 10000
```

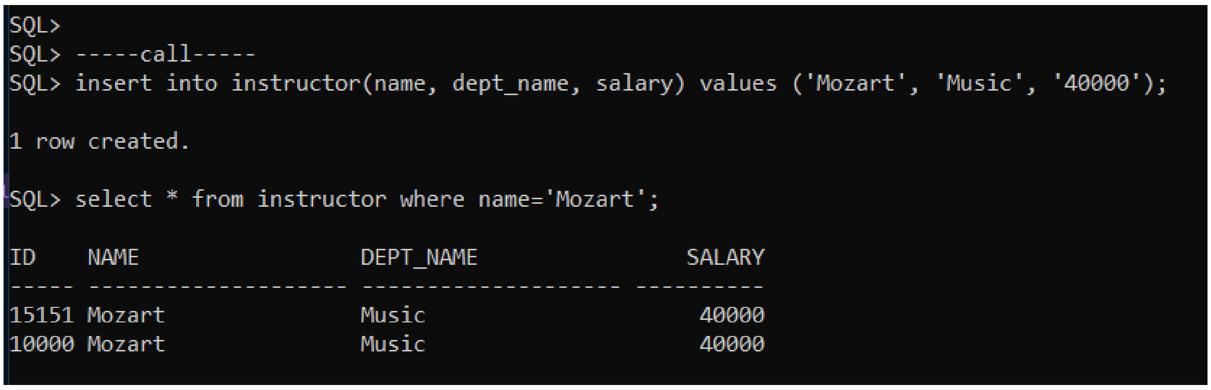
```
increment by 1
cache 500;

drop trigger trigger_generate_id;
create or replace
trigger trigger_generate_id
before insert on instructor
for each row
begin
    select ins_seq.nextval
    into :new.id
    from dual;
end;
/

-----call-----
insert into instructor(name, dept_name, salary)
values ('Mozart', 'Music', '40000');
```

4.2 Analysis and Explanation

This task was completed easily by following the instructions in the PDF document. At first I made a sequence to auto generate the instructor ID starting from 10000 and incrementing by 1 until 99999. Then I created a trigger which was executed everytime a new instructor was inserted. It was set to trigger before the insertion so that the ID of the new instructor could be set using the trigger using the sequence values. Here `:new.id` refers to the `id` attribute of the new values being inserted into the instructor table. To test the trigger, a new instructor was inserted into the table. The result is shown below. The second row shows the new instructor with the auto generated ID that is 10000 here.

4.3 Difficulties

I did not face any mentionable issues when solving this task.



```
SQL>
SQL> -----call-----
SQL> insert into instructor(name, dept_name, salary) values ('Mozart', 'Music', '40000');

1 row created.

SQL> select * from instructor where name='Mozart';

ID    NAME                 DEPT_NAME                SALARY
----- -------------------- -------------------- ----------
15151 Mozart               Music                     40000
10000 Mozart               Music                     40000
```

Figure 6: Task 4

5 Task 5

5. Create a trigger that will automatically assign an advisor to a newly admitted student of his/her own department. In case there are multiple teachers, the advisor should be selected based on the least number of students advised.

5.1 Solution

```
-----Task 5-----

-----trigger-----

drop trigger assign_advisor;

create or replace

trigger assign_advisor

after insert on student

for each row

declare

    advisor_id instructor.id%type;
```

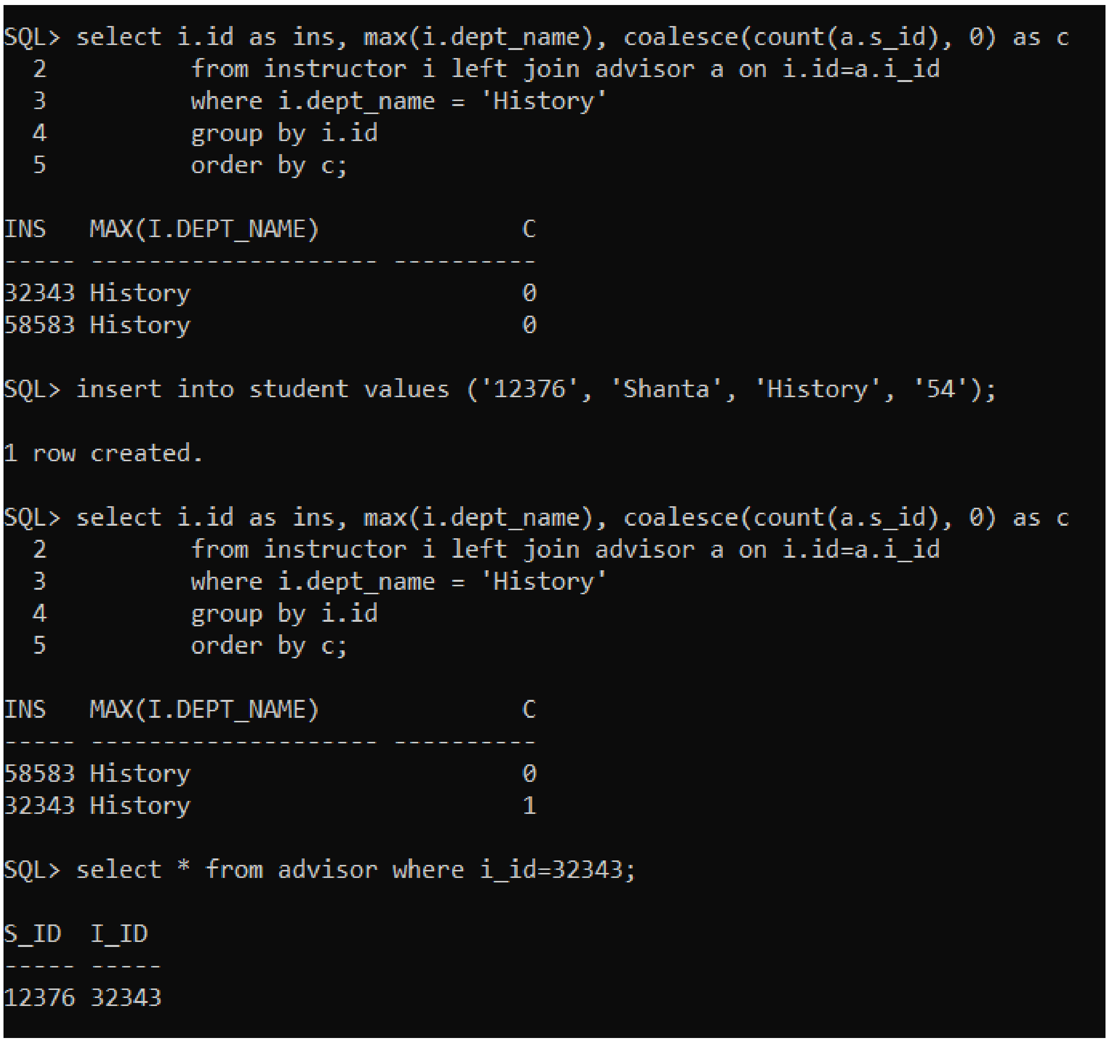
```
begin
    select ins into advisor_id
    from
    (
        select i.id as ins, max(i.dept_name),
        coalesce(count(a.s_id), 0) as c
        from instructor i left join advisor a on i.id=a.i_id
        where i.dept_name = :new.dept_name
        group by i.id
        order by c
    )
    where rownum<=1;

    insert into advisor(i_id,s_id) values(advisor_id,:new.ID);
end;
/

-----call-----
insert into student values ('12376', 'Shanta', 'History', '54');
```

5.2 Analysis and Explanation

For this task, I wrote a SQL query which returned all the instructor ID with their respective departments and the number of students they are advising currently in ascending order. The `coalesce()` combined with the left joining of the two tables showed 0 for those instructors with no students to advise. This query was nested within another which returned only the instructor ID and only one row that is the top row from the nested query so that we get the instructor with the least students using `rownum`. The nested query also checks if the instructor is in the same department as the student being inserted or not. The ID returned by the query is stored in the variable `advisor_id` which is later inserted into the advisor table along with the ID of the new student being inserted. A sample student was inserted to test the trigger. The result is shown below.

5.3 Difficulties

I faced a lot of difficulties when doing this task. The SQL query was easy to write however debugging the trigger was difficult even with `show error` statement. I faced errors because of not renaming the column which returned the instructor ID. After renaming the column as `ins` the trigger was created without compilation errors.



```
SQL> select i.id as ins, max(i.dept_name), coalesce(count(a.s_id), 0) as c
  2          from instructor i left join advisor a on i.id=a.i_id
  3          where i.dept_name = 'History'
  4          group by i.id
  5          order by c;

INS   MAX(I.DEPT_NAME)              C
----- -------------------- ----------
32343 History                       0
58583 History                       0

SQL> insert into student values ('12376', 'Shanta', 'History', '54');

1 row created.

SQL> select i.id as ins, max(i.dept_name), coalesce(count(a.s_id), 0) as c
  2          from instructor i left join advisor a on i.id=a.i_id
  3          where i.dept_name = 'History'
  4          group by i.id
  5          order by c;

INS   MAX(I.DEPT_NAME)              C
----- -------------------- ----------
58583 History                       0
32343 History                       1

SQL> select * from advisor where i_id=32343;

S_ID  I_ID
----- -----
12376 32343
```

Figure 7: Task 5

Conclusion

As shown in the report, I have solved and tested the solutions for all five tasks given in the lab. All the commands used were written in VS Code which was then saved with .sql extension. The .sql file was then run through the SQL command line to execute all the commands.