Islamic University of Technology (IUT)

Report on Lab 3

Submitted By

Shanta Maria (ID:200042172)

CSE 4410 Database Management Systems II Lab

Submitted To

Dr. Abu Raihan Mostofa Kamal

Professor, Department of CSE

Zannatun Naim Srsity

Lecturer, Department of CSE

Md. Rafid Haque

Lecturer, Department of CSE

January 30, 2023

Introduction

These lab tasks were based on the use of PL/SQL.

Question

1. Write a procedure to that will take a mov_title and show the require time (–hour –minute) to play that movie in a cinema hall. Let say, there will be an intermission after each 70 minutes only if the remaining time of the movie is greater than 30 minutes.
2. Write a procedure to find the N top-rated movies (average rev_stars of a movie is higher than other movies). The procedure will take N as input and print the mov_title upto N movies. If N is greater then the number of movies, then it will print an error message.
3. Suppose, there is a scheme that for each rev_stars greater than or equal to 6, a movie will receive $10. Now write a function to calculate the yearly earning (total earning /year in between current date and release date) of a movie that is obtained from user review.

Table 1: Movie Category Table for Question 4.

Genre Status	Review Count	Average Rating [avg of rev_stars]
Widely Watched	>avg review count of different genres	<avg rating of different genres
Highly Rated	<avg review count of different genres	>avg rating of different genres
People's Favorite	>avg review count of different genres	>avg rating of different genres
So So	otherwise	

4. Write a function, that given a genre (gen_id) will return genre status, additionally the review count and average rating of that genre.
5. Write a function, that given two dates will return the most frequent genre of that time (according to movie count) along with the count of movies under that genre which had been released in the given time range .

1 Task 1

1.1 Analysis and Explanation

For this task, I made a procedure that took the movie title as the input. Inside the procedure, a cursor is declared which returns the movie time of the input movie title from the database.

```
------1------
-----procedure-----
create or replace
procedure cinema_time(m_title in movie.mov_title%type)
is
    mov_time movie.mov_time%type;
    count_hours number;
    count_mins number;
    count_int number;

    cursor find_movie(m_title movie.mov_title%type)
    is
        select mov_time
        from movie
        where mov_title = m_title;

begin
    open find_movie(m_title);
    loop
        fetch find_movie into mov_time;
        exit when find_movie%notfound;

            count_int := floor(mov_time/70);
            mov_time := mov_time + (count_int * 15);
            count_hours := floor(mov_time/60);
            count_mins := mov_time - (60*count_hours);

            dbms_output.put_line('Cinema Time: ' || to_char(count_hour
s) ||
            ' hours and ' || to_char(count_mins) || ' minutes.');

    end loop;
    close find_movie;
end;
/
```

```
SQL> declare
  2      m_title varchar(50);
  3  begin
  4      m_title := '& m_title';
  5      cinema_time(m_title);
  6  end;
  7  /
Enter value for m_title: Spirited Away
Cinema Time: 2 hours and 20 minutes.

PL/SQL procedure successfully completed.
```

I then fetched the cursor result into a variable after opening it. To find the total cinema time, firstly I calculated the total number of intervals for the movie by dividing the time by 70 and getting the floor value of the result using the floor() function. Next, I added the extra 15 minutes of interval time to the movie time for all the possible intervals combined. Then I broke down the time into hours and minutes and printed the output time.

1.2 Difficulties

It was challenging to understand whether to run a loop to calculate the number of hours for the movie time since floor() was an unknown function. After understanding the use of floor() function in SQL, it was easier to break down the time. Other than that, I did not face any mentionable issues.

2 Task 2

2.1 Analysis and Explanation

For this task, I made a procedure that took the number n as input and printed the top n movies according to their average rating. For this, a cursor was declared. The cursor consists of a nested sql statement where one statement returns all the movie titles with their average rating using the avg() function which was rounded to 2 decimal places using the round() function and the group by clause. The results were ordered in descending order according to the average rating so that when the outer statement of the nested SQL returned only the top n rows, it would be the top n rated movies. To show an error for exceeding the number of movies, I used the raise_application_error() function and checked the validity using the total number of rows in the movie table. After checking the validity of the input number, the cursor is opened and the results are fetched into variables and then printed.

```
------2------
-----procedure-----
create or replace
procedure top_rated(n in number)
is
    row_num int;
    mov_title movie.mov_title%type;
    rev_stars rating.rev_stars%type;

    cursor find_movies(n number)
    is
        select *
        from
        (
            select mov_title, round(avg(rev_stars), 2) as r_stars
            from movie natural join rating
            group by mov_title
            order by avg(rev_stars) desc
        )
        where rownum<=n;
begin
    for row in (select count(*) as c from movie) loop
        if n>row.c then
            raise_application_error(-2000, 'Number exceeds total number
of movies!');
        else
            open find_movies(n);
            row_num := 1;
            loop
                fetch find_movies into mov_title, rev_stars;
                exit when find_movies%notfound;

                dbms_output.put_line(row_num);
                dbms_output.put_line('Movie Title: ' || mov_title);
                dbms_output.put_line('Average Rev Stars: ' || rev_star
s);
                dbms_output.put_line('-----');

                row_num := row_num + 1;

            end loop;
            close find_movies;
        end if;
    end loop;
end;
/
```

```
SQL> -----call-----
SQL> declare
  2      num int;
  3  begin
  4      num := '& num';
  5      top_rated(num);
  6  end;
  7  /
Enter value for num: 3
1
Movie Title: The Shining
Average Rev Stars: 8.4
-----
2
Movie Title: The Shawshank Redemption
Average Rev Stars: 8.23
-----
3
Movie Title: Braveheart
Average Rev Stars: 7.55
-----

PL/SQL procedure successfully completed.
```

2.2 Difficulties

I faced difficulty in rounding the average rating to 2 decimal places but I was able to solve the problem by using the round() function. Other than that, I did not face any mentionable issues.

3 Task 3

3.1 Analysis and Explanation

For this task, I designed a function that took the movie title as input and returned a string (varchar2) result. Inside the procedure, I made a cursor that returns the release date and the rating of the input movie only if the rating is above 6. Then after opening the cursor and fetching the results into variables, I used a temporary variable to first calculate the total earnings by multiplying 10 with (10-rating) since 10 dollars is multiplied with every rating after 6 not before. The cursor loops until all the rows are traversed so for each rating the earning is added to the previously calculated result. After the loop is completed, I divided the earnings by the

```
------3------
-----function-----
create or replace
function yearly_earning(m_title in movie.mov_title%type)
return varchar2
is
    yearly_earning varchar2(20);
    mov_releasedate movie.mov_releasedate%type;
    temp_calc number;
    r_date number;
    c int;

    cursor find_rev_stars(m_title movie.mov_title%type)
    is
        select mov_releasedate, rev_stars as c
        from movie natural join rating
        where mov_title=m_title and rev_stars>=6;

begin
    temp_calc := 0;
    open find_rev_stars(m_title);

    loop
        fetch find_rev_stars into mov_releasedate, c;
        exit when find_rev_stars%notfound;
            temp_calc := temp_calc + (10*(10-c));
    end loop;

    r_date := extract(year from mov_releasedate);
    temp_calc := round((temp_calc/(2023-r_date)), 2);
    yearly_earning := to_char(temp_calc);

    close find_rev_stars;
return yearly_earning;
end;
/
```

```
SQL> -----call-----
SQL> declare
  2      movie varchar2(25);
  3  begin
  4      movie := '& movie';
  5      dbms_output.put_line(yearly_earning(movie));
  6  end;
  7  /
Enter value for movie: The Shining
3.72

PL/SQL procedure successfully completed.
```

difference of years between the current year and the year in which the movie was released. The result is the yearly earnings which was returned from the function.

3.2 Difficulties

It was challenging to understand the calculation of the yearly earnings. Other than that, I did not face any mentionable issues.

4 Task 4

4.1 Analysis and Explanation

For this task, I created a function that took the genre ID as input and returned a varchar2 result. I made two cursors inside this function. One cursor returns the details of the input genre. It counts the number of reviews using count() and calculates the average of all the ratings round to 2 decimal places using avg() and round() by grouping the results according to the given genre ID. The second cursor returns the average review count and the average rating of all the different genres. It uses the SQL query of the first cursor without specifying the genre ID and nests it within a from clause. Then it calculates the average review count and rating from the results of the nested query. Both the cursors are opened later and the results are fetched into variables. The results of the specified genre are checked against the results of all the different genres according

```
------4------
-----function-----
create or replace
function genre_status(g_id in number)
return varchar2
is
    result varchar2(100);
    avg_rev number;
    avg_rating number;
    g_status varchar2(25);
    rev_count number;
    rating number;

    -----to find the details of the specified genre-----
    cursor find_gen_details(g_id number)
    is
        select count(rev_id) as rev_count, round(avg(rev_stars), 2) as
 rating
        from genres natural join mtype natural join
        movie natural join rating
        where gen_id = g_id
        group by gen_id;

    -----to find the average details of all genres-----
    cursor diff_gen
    is
        select avg(rev_count) as avg_rev, avg(rating) as avg_rating
        from
            (select count(rev_id) as rev_count, avg(rev_stars) as ratin
g
            from genres natural join mtype natural join
            movie natural join rating
            group by gen_id);

begin
    -----initialising variables-----
    result := 'nothing';
    g_status := 'nothing';
    rev_count := 0;
    rating := 0;
    avg_rating := 0;
    avg_rev := 0;

    open diff_gen;
    fetch diff_gen into avg_rev, avg_rating;

    open find_gen_details(g_id);
    fetch find_gen_details into rev_count, rating;

    if rev_count>avg_rev and rating<avg_rating then
        g_status := 'Widely Watched';
    elsif rev_count<avg_rev and rating>avg_rating then
        g_status := 'Highly Rated';
    elsif rev_count>avg_rev and rating>avg_rating then
        g_status := 'People' || 's Favorite';
    else
        g_status := 'So So';
    end if;

    result := 'Genre Status: ' || g_status || chr(10);
    result := result || 'Review Count: ' || rev_count || chr(10);
    result := result || 'Average Rating: ' || rating || chr(10);

    -----chr(10) = new line character-----

    close find_gen_details;
    close diff_gen;

return result;
end;
/
```

```
SQL> -----call-----
SQL> declare
  2      gen_id genres.gen_id%type;
  3  begin
  4      gen_id := '& gen_id';
  5      dbms_output.put_line(genre_status(gen_id));
  6  end;
  7  /
Enter value for gen_id: 1011
Genre Status: Peoples Favorite
Review Count: 25
Average Rating: 6.78


PL/SQL procedure successfully completed.
```

to the conditions specified in the question and a variable g_status meaning the genre status is set according to the conditions. This condition checking is done using if statements. The wanted outputs are then converted to char and added to a single varchar2 variable which the functions returns.

4.2 Difficulties

I faced a lot of difficulties when doing this task. At first, the function showed errors for char to number conversion since I used + instead of || when concatenating strings. Next, I did not know the char concatenate new line to a variable. After much research, I was able to find the answer. It was also difficult to figure out how to calculate the average rating of different genres as well as the input genre within a single cursor since I was unfamiliar with using multiple cursors.

```
------5------
-----function-----
create or replace
function freq_gen(d1 in date, d2 in date)
return varchar2
is
    result varchar2(100);
    gen_title genres.gen_title%type;
    mov_count number;

    cursor find_freq_gen(d1 date, d2 date)
    is
        select max(gen_title) as gen_title, count(mov_id) as mov_count
        from movie natural join mtype natural join genres
        where mov_releasedate>d1 and mov_releasedate<d2
        group by gen_id
        order by mov_count desc;

begin
    result := 'nothing';

    open find_freq_gen(d1, d2);
    fetch find_freq_gen into gen_title, mov_count;

    result := 'Frequent Genre: ' || gen_title || chr(10);
    result := result || 'Movie Count: ' || mov_count;

    close find_freq_gen;
    return result;
end;
/
```

```
SQL> -----call-----
SQL> declare
  2      d1 varchar2(10);
  3      d2 varchar2(10);
  4  begin
  5      d1 := '& d1';
  6      d2 := '& d2';
  7
  8      dbms_output.put_line(freq_gen(to_date(d1), to_date(d2)));
  9  end;
 10  /
Enter value for d1: 24-MAR-50
Enter value for d2: 27-JAN-90
Frequent Genre: Drama
Movie Count: 2

PL/SQL procedure successfully completed.
```

5 Task 5

5.1 Analysis and Explanation

For this task, I created a function that takes two dates as input. The user input is taken as a string but the to_date() function converts it to date type. A cursor is declared inside the function which returns the genre title with the number of movies of the genre between the input dates. The query checks if the release date is in between the two dates and groups the result by genre ID to count the number of movies for each genre using the count() function. The results were also ordered according to the movie count in descending order. This is because when the cursor is opened, only the first result is fetched and stored into variables since we want only the genre with the highest movie count and this is possible because of how the query results were ordered earlier. The results are then concatenated as strings into a varchar2 variable which is returned by the function at the end.

5.2 Difficulties

I faced difficulty in converting the user input date into date format but other than that, I did not face any mentionable issues.

Conclusion

As shown in the report, I completed all the lab tasks. All the commands used were written in VS Code which was then saved with the .sql extension. The .sql file was then run through the SQL command line to execute all the commands.