Islamic University of Technology
Lab Task 03
CSE 4410 : DBMS-II Lab

Submitted To :

Dr. Abu Raihan Mostofa Kamal
Professor, CSE Department
Islamic University of Technology

Zannatun Naim Sristy
Lecturer, CSE Department
Islamic University of Technology

Submitted By :

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

Report :

Analysis : In this task, we need to write appropriate PL/SQL procedure/function solving the given questions.

Working code :

```
-->1<--
create or replace Procedure
required_time(title in varchar)
as
    time number;
    hour number;
    minute number;
    intermission number;
begin
    select mov_time into time
    from movie
    where mov_title = title;

    intermission := 0;
    hour := 0;
    minute := 0;

    intermission:=floor(time/70);
    intermission:=intermission*15;
    time := time + intermission;
    hour:=floor(time/60);
    minute:=time-hour*60;

    DBMS_OUTPUT.PUT_LINE( 'HOUR: ' || hour || ' || Minute: ' || Minute ||
' || Intermission(mins): ' ||intermission);
end;
/

begin
    required_time('Vertigo');
end;
/
```

```
-->2<--
create or replace Procedure
top_rated(n in number)
as
    c number;
    c2 number;
    average number;
begin
    select avg(rev_stars) into average
    from movie natural join rating natural join reviewer;

    c := 0;
    c2 := 0;

    for row in (select mov_title,rev_stars from movie natural join rating
natural join reviewer) loop
        if(row.rev_stars>average)then
            c := c + 1;
        end if;
    end loop;
    if(c<n) then
        DBMS_OUTPUT.PUT_LINE('Error : Not enough movies!');
    else
        for row in (select mov_title,rev_stars from movie natural join
rating natural join reviewer where rev_stars>average) loop
            c2 := c2 + 1;
            if(c2<=n)then
                DBMS_OUTPUT.PUT_LINE(row.mov_title);
            end if;
        end loop;
    end if;
end;
/

begin
    top_rated(15);
end;
/

-->3<--
```

```
create or replace
function yearly_earnings(movieid number)
return number
is
    earnings number;
    release_date date;
    n number;
    yearly number;
begin
    earnings := 0;

    select mov_releasedate into release_date
    from  movie
    where mov_id = movieid;


    for row in (select rev_stars from movie natural join rating natural
join reviewer where mov_id = movieid and rev_stars>=6) loop
        earnings := earnings + (10*(10-row.rev_stars));
    end loop;


    yearly:=earnings/YEAR(sysdate-release_date);

    return yearly;
end;
/

BEGIN
    DBMS_OUTPUT.PUT_LINE(yearly_earnings(901));
END;
/


-->4<--

create or replace
function genre_info(genid number)
return varchar
```

```
is
    rev_count number;
    v_avg number;
    v_status varchar2(200);
    total_average number;
    total_count number;
    result varchar2(1000);
begin
    v_status := 'nice genre';

    select count(rev_id) into rev_count
    from genres natural join mtype natural join rating
    where gen_id = genid;

    select avg(rev_stars) into v_avg
    from genres natural join mtype natural join rating
    where gen_id = genid;

    select avg(rev_stars) into total_average
    from genres natural join mtype natural join rating;


    select avg(rev_count) into total_count
    from(
            select count(rev_id) as rev_count
            from genres natural join mtype natural join rating
            group by gen_title
        );


    if(rev_count > total_average AND v_avg < total_average) THEN
        v_status := 'Widely Watched';
    elsif(rev_count < total_average AND v_avg > total_average) THEN
        v_status := 'Highly Rated';
    elsif(rev_count > total_average AND v_avg > total_average) THEN
        v_status := 'Peoples favorite';
    else
        v_status := 'So so';
    end if;
```

```
    result := 'Genre status : ' || v_status || ' || Review count : ' ||
rev_count || ' || Average rating : ' || v_avg;

    return result;
end;
/

DECLARE
    result varchar2(1000);
begin
    result := genre_info(1002);
    DBMS_OUTPUT.PUT_LINE(result);
end;
/

-->5<--

create or replace
function freq_genre(start_date movie.mov_releasedate%type, end_date
movie.mov_releasedate%type )
return varchar2
IS
    genre_id number;
    cnt_mov number;
    genre_name varchar2(100);
    result varchar2(1000);
BEGIN

    SELECT id INTO genre_id
    FROM (
            SELECT mtype.gen_id AS id , count(mov_range.MOV_ID) AS
mov_count
            FROM (
                    SELECT *
                    FROM MOVIE
                    WHERE MOVIE.MOV_RELEASEDATE BETWEEN START_DATE AND
END_DATE
                  ) mov_range , mtype
            WHERE mov_range.MOV_ID = mtype.MOV_ID
```

```
            GROUP BY mtype.gen_id
            ORDER BY mov_count DESC
        )popular_gen
    WHERE ROWNUM<=1;

    select gen_title into genre_name
    from genres
    where gen_id = genre_id;

    select count(movie.mov_id) into cnt_mov
    from movie natural join mtype
    where gen_id = genre_id and movie.release_date between start_date and
end_date;


    result := genre_name || ' Count of movies : ' || cnt_mov;

    return result;

end;
/


DECLARE
    start_date movie.mov_releasedate%type;
    end_date movie.mov_releasedate%type;
BEGIN
    start_date := '&Start date : ';
    end_date := '&End date : ';
    DBMS_OUTPUT.PUT_LINE(freq_genre(to_date(start_date),
to_date(end_date));
end;
/
```

Explanation of the code :

1) The required_time procedure here takes in a varchar parameter which holds the title of the movie from which we need to find the total required time to play the movie in a theater. The query gives the time of the aforementioned movie into a number variable. The time is in minutes. We divide the time by 70 and apply floor function to find the number of intermissions. After that we add the intermission with the time variable. Now we calculate the hour and minute from the time variable and show it in the console using DBMS.OUTPUT_LINE function.
2) The procedure top_rated takes in a number variable n, which represents the number of top rated movies to be shown. The query returns the average rating of all movies that have a rating. The first loop counts the number of movies which have a higher rating than the average rating. The if condition handles the error that if the provided number n is greater than the count of higher than average movies. Otherwise another for loop shows those movies up to n.
3) The yearly_earnings function takes in a number parameter which represents a movie id and returns the yearly earning of that movie based on reviewer ratings. The query puts the release date of the movie into a date variable. The loop iterates through all the ratings the movie has gotten above or equal to 6. The ratings are multiplied by 10 for each rating above 6 and incremented into a number variable earnings. Afterwards the earnings are divided by the year difference between current date and the release date of the movie. Year function is used to do that. Finally the calculated value is returned.
4) The genre_info function takes in a number variable of genid which represents the genre id and returns a varchar. There are a number of queries. The first query finds the total number of reviews/ratings of that specific genre and puts it in the rev_id variable. The second query finds the average ratings of all movies in the genre and puts it in the v_avg variable. The third query finds the total average rating of all genres and puts it in the total_average variable. The last query uses nested query to find the average count of reviews across all genres and puts it in the total_count variable. The if statements implement given conditions using the aforementioned variables for genre status. Finally all of the results of the operations performed are properly put in a varchar variable and returned.
5) The freq_genre function takes two parameters of type date which represent the start time and end time provided by the user. There are 3 queries. The first query

uses nested queries to figure out the genre id of the most frequent genre of the given time range. The innermost nested query shows only those movies which are within the time range. The outer nested query counts the number of movies within that range and groups them according to their genre id and sorts them in descending order. Finally the query only shows the topmost genre id using the rownum clause. The second query uses the genre id from query 1 to find the title of the genre and the third query finds the count of movies under the genre within the time range. Finally the genre title and movie count is put in a varchar2 variable and returned.

Problems :

- There were some issues declaring variables in functions. After putting appropriate size constraints, the issue was resolved.
- Figuring out the proper nested queries proved to be quite a challenge.