LAB REPORT

CSE 4410

DATABASE MANAGEMENT SYSTEMS II LAB

LAB_03: PL/SQL

NAME: CHOWDHURY ASHFAQ

STUDENT ID: 200042123

PROGRAM: SWE

GROUP: 1A

DATE: 30/01/23

Tasks:

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 of 15 minutes 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

Analysis of the problem:

Functions and Procedures are an important part of PL/SQL. We were given a scenario and based on that scenario we had to write 2 Procedures and 3 functions.

Solution:

```
CREATE OR REPLACE PROCEDURE get_time (movie_title IN VARCHAR2, hour OUT NUMBER, minute OUT NUMBER)
AS
    time NUMBER;
    intermission NUMBER;
begin
  SELECT MOV_TIME INTO time FROM MOVIE WHERE MOV_TITLE = movie_title;
  intermission := TRUNC (time/ (70+15));
  time:= time+(intermission*15);
  hour := TRUNC(time/60);
  minute:= time-(hour*60);
    DBMS_OUTPUT.PUT_LINE('Movie Title: '|| movie_title || ' and RUNNING TIME: ' || hour || ' hours ' || minute || ' minutes');
end;
/


DECLARE
hour NUMBER;
minute NUMBER;

begin
  get_time('Beyond the Sea',hour,minute);
end;
/
```

```
CREATE OR REPLACE PROCEDURE get_movies(rating IN NUMBER)
AS
    CURSOR top_rated_cursor IS
        SELECT MOV_TITLE, AVG(REV_STARS) AS STARS
        FROM MOVIE, RATING
        WHERE MOVIE.MOV_ID=RATING.MOV_ID
        GROUP BY MOV_TITLE
        ORDER BY STARS DESC;

    mov_title top_rated_cursor%ROWTYPE;
    cnt NUMBER;

begin
  cnt:=0;

  OPEN top_rated_cursor;

  loop
    FETCH top_rated_cursor INTO mov_title;
    EXIT WHEN top_rated_cursor%NOTFOUND;
    DBMS_OUTPUT.PUT_LINE(mov_title.mov_title);
    cnt := cnt+1;
    EXIT WHEN cnt = rating;
  end loop;

    CLOSE top_rated_cursor;

    IF cnt<rating THEN
        DBMS_OUTPUT.PUT_LINE('Error: N is greater than number of movies');
    END IF;
end;
/

begin
  get_movies(4);
end;
/
```

```
CREATE OR REPLACE FUNCTION get_yearly_earning(mov_id NUMBER, current_date DATE, release_date DATE)
RETURN NUMBER
AS
  rev_stars NUMBER;
  total_earning NUMBER;
BEGIN
  SELECT AVG(rev_stars) INTO rev_stars
  FROM REVIEW
  WHERE MOV_ID = mov_id;

  IF rev_stars >= 6 THEN
    total_earning := (current_date - release_date) * 10;
  ELSE
    total_earning := 0;
  END IF;

  RETURN total_earning / (current_date - release_date);
END;
/

begin
DBMS_OUTPUT.PUT_LINE(get_yearly_earning(902,1962-02-19,1962-12-11));
end;
/
```

```
CREATE OR REPLACE FUNCTION get_genre_status(g_id IN NUMBER) RETURN VARCHAR2
AS

    status VARCHAR2(20);
    review_cnt NUMBER;
    avg_rating NUMBER;

begin
  SELECT GEN_TITLE into status FROM GENRES WHERE gen_id=g_id;
  if status IS NOT NULL THEN
    status:= 'Exists';
  else
    status:= 'Not Exists';
  end if;

  SELECT COUNT(*) AS review_cnt, AVG(REV_STARS) AS avg_stars
  INTO review_cnt, avg_rating
  FROM MOVIE
  JOIN MTYPE on movie.MOV_ID=mtype.MOV_ID
  JOIN RATING on movie.mov_id = rating.mov_id
  where mtype.gen_id=g_id;

  RETURN status || ' | ' || review_cnt || ' | ' || avg_rating;
end;
/

begin
  DBMS_OUTPUT.PUT_LINE(get_genre_status(1));
end;
/
```

```
CREATE OR REPLACE FUNCTION get_count_genre_in_range(start_date DATE, end_date DATE)
RETURN VARCHAR2
AS
  most_frequent_genre VARCHAR2(20);
  movie_count INT;
BEGIN
  SELECT GEN_TITLE, COUNT(*) INTO most_frequent_genre, movie_count
  FROM MOVIE,MTYPE,GENRES
  WHERE MOVIE.MOV_ID=MTYPE.MOV_ID AND MTYPE.GEN_ID=GENRES.GEN_ID AND
  MOV_RELEASEDATE BETWEEN start_date AND end_date
  GROUP BY GEN_TITLE
  ORDER BY COUNT(*) DESC
  FETCH FIRST ROW ONLY;

  RETURN most_frequent_genre || ' (' || movie_count || ')';
END;
/

begin
  DBMS_OUTPUT.PUT_LINE(get_count_genre_in_range(1962-02-19,1962-12-11));
end;
/
```

Explanation:

1)
get_time is a procedure which gives the playing time of a movie as OUT parameter based on movie_title. The procedure starts by using a SELECT statement to retrieve the time of the movie from the MOVIE table. The intermission is calculated as per the format (70 + 15) minutes. Finally, the hour and minute are calculated by dividing the time by 60 and storing the result in hour and subtracting the result of hour * 60 from time and storing it in minute.

2)
The procedure get_movies() retrieves a list of top rated movies from the MOVIE and RATING tables, based on the average rating (REV_STARS) of each movie. The procedure accepts a parameter 'rating' which represents the number of top rated movies that the user wants to retrieve.
The procedure has a cursor 'top_rated_cursor' which retrieves the movie title and the average rating of each movie. The cursor is opened and looped through to retrieve each movie title. The loop will fetch each row from the cursor and store it in the mov_title record. The loop continues until the cursor reaches the end of the data or until the

number of rows fetched is equal to the value of 'rating' specified by the user. The cursor is closed when the loop completes.

```
DBMS_OUTPUT.PUT_LINE(mov_title.mov_title);
```

This line generates the required output.

3)
The function "get_yearly_earning" calculates the yearly earning of a movie based on its review stars. The function first retrieves the release date and current date of the movie. It then calculates the number of years between the current date and the release date by subtracting the year part of the release date from the year part of the current date. The function then calculates the total earning of the movie by multiplying the number of years by $10, assuming that for each review star greater than or equal to 6, the movie will receive $10. The function returns the yearly earning of the movie by dividing the total earning by the number of years.

4)
The function first checks if the genre exists by querying the GENRES table using the input g_id. If the genre exists, the status is set to 'Exists'. If not, the status is set to 'Not Exists'.

Then, the function calculates the count of movie reviews and the average rating for movies of the genre represented by the input g_id by joining the MOVIE, MTYPE, and RATING tables. It returns genre status, review count and average rating of that genre.

5)
The function starts by selecting the GEN_TITLE (genre title) and the count of movies for that genre from the MOVIE, MTYPE, and GENRES tables. The join is done on the MOV_ID column in all three tables to obtain the data from all three tables.

Then, the query filters the results to only include the movies that were released between the start_date and end_date.

Finally, the query groups the data by GEN_TITLE, orders the count of movies in each genre in descending order, and fetches only the first row of the result set. This first row represents the most frequently occurring

genre in the given date range. In the end the function returns most frequent genre along with the count of movies under that genre.

Problems Faced:

Question was a bit complex to understand like the third one.