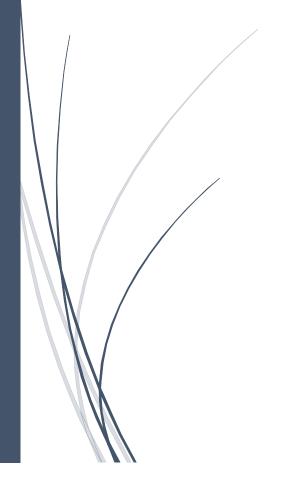
Name: K.M. Tahlil Mahfuz Faruk

Student ID: 200042158

Department: CSE(SWE)

Course: CSE 4410

Database Management System II LAB 3



K.M. Tahlil Mahfuz Faruk Islamic University of Technology

SQL Commands:

In task 1,

Explanation:

- A mov title will be passed in the function parameter.
- Find the title's time. Divide it by 60 and round to find the hour. Then find the minute by subtracting the hour*60 from the time.
- Now divide the time by 70 and round to find the interval time. Interval would be valid if remaining time is greater than 30 mins. Check that condition by (c-1)*70>=30.

Difficulties:

No significant difficulty appeared.

In task 2,

```
topratedmovies(n in number)
   i number;
   average number;
    select avg(rev_stars) into average
   for row in (select mov_title,rev_stars from movie natural join rating natural join reviewer) loop
        if(row.rev_stars>average)then
       DBMS_OUTPUT.PUT_LINE( A: 'Error');
            DBMS_OUTPUT.PUT_LINE( A: row.mov_title);
    topratedmovies( n: 200);
```

Explanation:

- First find the average rev_stars and save it tin average variable.
- Now count the number of movies that has greater review than average.
- If it's greater than n then print error.
- Else print those movies.

Difficulties:

· No such difficulties were faced during this task.

In task 3,

```
function yearlyearnings(movieid number)
earnings number;
yearly number;
rd date;
    earnings:=1;
    select mov_releasedate into rd
    from movie
    where mov_id=movieid;
    select count(mov_id) into n
    from movie natural join rating natural join reviewer
    earnings:=(10*n);
    yearly:=earnings/((sysdate-rd)/365);
   return yearly;
begin
    DBMS_OUTPUT.PUT_LINE( A: 'Yearly Income: ' || yearlyearnings( movieid: 902));
```

Explanation:

- First find the release data of the movie and also find the number of reviews that movie has got which is greater than 6.
- Now multiply by 10 with the number of reviews and then divide it by the year.
- That will provide the yearly income.

Difficulties:

To find the year from the release data was the challenge.

In task 4,

```
function genrestatusshow(genreid in number)
avg_review_of_all_genre number;
       select gen_id,gen_title,avgstars,avg_reviews_per_genre
          from genres natural join rating
          group by genres.gen_id, GENRES.GEN_TITLE
       from genres natural join REVIEWER
       from genres natural join rating
       group by genres.gen_id, GENRES.GEN_TITLE
      DBMS_OUTPUT.PUT_LINE( A: GENRESTATUSSHOW( genreid: 1012));
```

Explanation:

- First find the average review stars. To find it do a subquery.
- Also do a subquery to find the average review count.
- Now natural join both of these queries using the gen_id that will provide the average stars and reviews and save the result in a cursor called dummytable.
- Now Find the average review and stars of all the genres using two queries and save them to variable avg_review_of_all_genres and avg_stars_of_all_genres respectively.
- Now compare according to condition given and return the genre status.

Difficulties:

• To do the subquery and get them to a table was the challenge.

In task 5,

```
type freq_genre_and_movie_count as object
    genre_id number,
function frequent_genre(starting varchar,ending varchar)
        where to_date(starting,'DD-MON-YY')<MOV_RELEASEDATE</pre>
          and MOV_RELEASEDATE<to_date(ending, 'DD-MON-YY') and rownum<=1</pre>
        from MOVIE natural join GENRES
        where to_date(starting, 'DD-MON-YY')<MOV_RELEASEDATE</pre>
          and MOV_RELEASEDATE<to_date(ending, 'DD-MON-YY')</pre>
    data:=frequent_genre( starting: '31-DEC-1940', ending: '31-DEC-1998');
    DBMS_OUTPUT.PUT_LINE( A: data.GENRE_ID || ' ' || data.GENRE_TITLE || ' ' || data.MOV_COUNT);
```

Explanation:

- First create or replace a composite data type that will contain gen_id, gen_title
 and frequent movie counts.
- Now set that data type as the return value of the function.
- Do a subquery of finding the number of movies between the starting date and ending date and sort them according to release data. First row will contain the most frequent genre.
- Now to find the number of movies of that genre, just do a subquery to find the number of movies of each genre and in the range of given date and natural join with the previous subquery.
- Now save them is the composite datatype variable data.
- Return data.

Difficulties:

- The composite data type was showing an error that the "Reference to uninitialized composite".
- To solve this we have to declare the data variable like this.

data:= freq_genre_and_movie_count(genre_id, genre_title, countfreq);