Islamic University of
Technology

Lab-3 Report: PL/SQL (Functions and Procedures)

Submitted by:

Name : Mashrur Ahsan

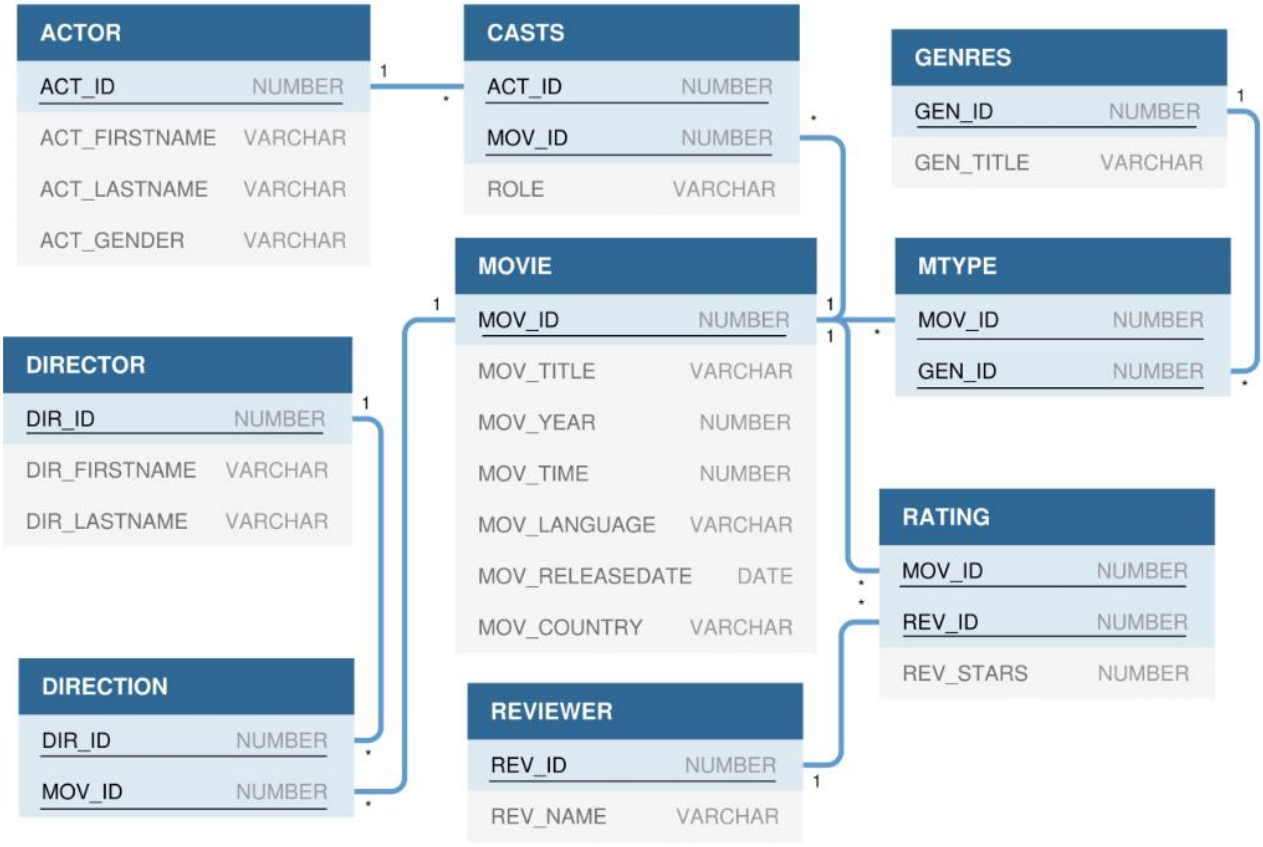
ID 200042115

Program : SWE

Department : CSE

Course Code : CSE 4410

The Schema:



Problem Statements

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.

The Solution:

```
3   -- 1
4   CREATE OR REPLACE PROCEDURE show_movie_time (movie_title IN VARCHAR2)
5   AS
6       movie_time NUMBER;
7       hours NUMBER;
8       minutes NUMBER;
9       intermission NUMBER;
10  BEGIN
11      SELECT MOV_TIME INTO movie_time
12      FROM MOVIE
13      WHERE MOV_TITLE = movie_title;
14
15      intermission := floor(movie_time/70) * 15;
16      minutes := mod(movie_time + intermission, 60);
17      hours := floor((movie_time + intermission) / 60);
18
19      dbms_output.put_line('Movie Title: ' || movie_title);
20      dbms_output.put_line('Running Time: ' || hours || ' hours ' || minutes || ' minutes');
21  END;
22  /
23
24  DECLARE
25      movie VARCHAR2(55);
26  BEGIN
27      movie := '&movie';
28      show_movie_time(movie);
29  END ;
30  /
```

Explanation & Analysis:

Line 11-13: Fetching the desired data (the running time of the film) from the “MOVIE” table into “movie_time” variable.

Line 15-17: Doing necessary calculations.

Line 27-28: Taking user input (the movie name).

Problems Faced:

- Took me a bit of time to figure out the correct logic behind the calculations.

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.

The Solution:

```
34  -- 2
35
36  CREATE OR REPLACE PROCEDURE find_top_movies (n IN NUMBER)
37  AS
38  MaxRows Number;
39  BEGIN
40  |
41  |   Select count(*) into MaxRows from (SELECT COUNT(movie.mov_id)
42  |   FROM movie, rating
43  |   where rating.mov_id = movie.mov_id
44  |   group by movie.mov_title);
45  |
46  |   IF( n > MaxRows) THEN dbms_output.put_line('Error.... Invalid rows');
47  |   ELSE
48  |   |   FOR i IN (select * from (SELECT mov_title, AVG(NVL(rev_stars, 0)) as avgerage_rating
49  |   |   |   |   |   FROM movie m, rating r
50  |   |   |   |   |   WHERE m.mov_id = r.mov_id
51  |   |   |   |   |   GROUP BY m.mov_title
52  |   |   |   |   |   ORDER BY avgerage_rating DESC)
53  |   |   |   |   |   where ROWNUM <= n)
54  |   |   |
55  |   |   LOOP
56  |   |   |   dbms_output.put_line(i.mov_title);
57  |   |   END LOOP;
58  |   End if;
59  END;
60  /
```

```
63  DECLARE
64  |   N number;
65  BEGIN
66  |   N := '&N';
67  |   find_top_movies(N);
68  END ;
69  /
```

Explanation & Analysis:

Line 41-44: The query is calculating the total number of movies that were reviewed by the reviewers. Then the number is getting stored into “MaxRows”.

Line 48-53: First, we are ordering the movies in a descending order (accordingly to their average ratings). Then we are selecting only the top N movies from that.

Line 56: Showing the top N movies one by one.

Problems Faced:

- Took me a bit of time to figure out the query (line 41-44) that will help me to handle the exception (error message).

The queries I needed to solve the problem:

```
72   SELECT mov_title, AVG(NVL(rev_stars, 0)) as avgerage_rating
73   FROM movie m, rating r
74   WHERE m.mov_id = r.mov_id
75   GROUP BY m.mov_title
76   ORDER BY avgerage_rating DESC;
77
78   SELECT COUNT(*) FROM movie m, rating r
79   WHERE m.mov_id = r.mov_id
80   GROUP BY m.mov_title ;
81
82   Select count(*) from (SELECT COUNT(movie.mov_id)
83   FROM movie, rating
84   where rating.mov_id = movie.mov_id
85   group by movie.mov_title);
```

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.

The Solution:

```
89   -- 3
90   CREATE OR REPLACE FUNCTION get_movie_earnings (id IN NUMBER)
91   RETURN NUMBER
92   AS
93   v_total_earnings NUMBER;
94   v_num_reviews NUMBER;
95   v_release_date DATE;
96   v_current_date DATE;
97   v_mov_id NUMBER;
98   BEGIN
99       SELECT COUNT(*) as num_reviews, m.mov_id
100      INTO v_num_reviews, v_mov_id
101      FROM rating r, Movie m
102      WHERE r.mov_id = m.mov_id AND r.mov_id = id AND r.rev_stars >= 6
103      group by m.mov_id;
104
105      SELECT m.mov_releasedate INTO v_release_date
106      FROM movie m
107      WHERE m.mov_id = id;
108
109      v_current_date := SYSDATE;
110
111      v_total_earnings := v_num_reviews * 10;
112
113      RETURN v_num_reviews/((v_current_date - v_release_date)/365);
114  END;
115  /
117  SELECT get_movie_earnings(901) FROM DUAL;         /* Without user input */
118
119
120  /* With user input */
121  DECLARE
122  id number;
123  BEGIN
124  id:= '&id';
125  dbms_output.put_line(get_movie_earnings(id));
126  end;
127  /
```

Explanation & Analysis:

Line 99-103: The query is calculating the total number of reviews that are greater than or equal to 6 for each movie. Then the relevant data is getting stored into the variables.

Line 105-107: Fetching the release date of the movie then storing it into a variable.

Line 109: Getting the current date.

Line 111: Getting the total earnings (according to the question).

Line 113: Returning the yearly income (total earnings ÷ ((total days past) ÷365)).

Problems Faced:

- Didn't know that `(v_current_date - v_release_date)` returned days rather than years.

The queries I needed to solve the problem:

```
130   SELECT COUNT(*) as times , m.mov_id
131   FROM rating r, movie m
132   WHERE m.mov_id = r.mov_id AND r.rev_stars >= 6
133   GROUP BY m.mov_id
134   ORDER BY mov_id DESC;
135
136   SELECT mov_id, mov_title, mov_releasedate from movie Order by mov_releasedate desc;
```

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.

The Solution:

```
140  -- 4
141
142  CREATE OR REPLACE FUNCTION get_genre_status (id IN NUMBER)
143  RETURN VARCHAR2
144  AS
145      gen_title VARCHAR2(20);
146      review_count NUMBER;
147      avg_rating NUMBER(5,3);
148      Genre_Status VARCHAR2(20);
149      avg_reviews number(5,3);
150      avg_rev_stars number(5,3);
151  BEGIN
152
153      /* Calculating the average number of reviews across all genres */
154      Select floor(Sum(total_reviews)/count(total_reviews)) INTO avg_reviews
155      from (SELECT g.GEN_TITLE, COUNT(r.REV_ID) as total_reviews
156      FROM RATING r
157      JOIN MTYPE mt ON r.MOV_ID = mt.MOV_ID
158      JOIN GENRES g ON mt.GEN_ID = g.GEN_ID
159      GROUP BY g.GEN_TITLE);
160
161      /* Calculating the average rating given by the reviewers */
162      SELECT AVG(NVL(REV_STARS, 0)) INTO avg_rev_stars FROM RATING;
```

```
164      /* Calculating the number of reviewes and the average rating for each genre*/
165      SELECT GEN_TITLE, COUNT(RATING.REV_ID), AVG(RATING.REV_STARS)
166      INTO gen_title, review_count, avg_rating
167      FROM GENRES , RATING , MTYPE
168      where GENRES.GEN_ID = MTYPE.GEN_ID AND MTYPE.MOV_ID = RATING.MOV_ID
169          AND GENRES.GEN_ID = MTYPE.gen_id AND id = GENRES.GEN_ID
170      GROUP BY GEN_TITLE;
171
172      IF( review_count > avg_reviews ) THEN
173          IF ( avg_rating < avg_rev_stars ) THEN Genre_Status := 'Widely Watched';
174          ELSIF ( avg_rating > avg_rev_stars ) THEN Genre_Status := 'People''s Favorite';
175          END IF;
176
177      ELSIF ( review_count < avg_reviews  AND  avg_rating > avg_rev_stars )
178      THEN Genre_Status := 'Highly Rated';
179
180      ELSE Genre_Status := 'So So';
181      END IF;
182
183      RETURN 'Genre: ' || gen_title || '   Reivew_Count: ' || review_count ||
184          '   Average_Rating: ' || avg_rating || '    Status: ' || Genre_Status;
185  END;
186  /
```

```
188   SELECT get_genre_status(1001) FROM DUAL;         /* Without user input */
189
190   DECLARE
191   id number;
192   BEGIN
193   id:= '&id';                                       /* With user input */
194   dbms_output.put_line(get_genre_status(id));
195   end;
196   /
197
```

Explanation & Analysis:

Line 154-159: Calculating, on average, how many reviews does one genre get. Basically, we are calculating the average number of reviews across all genres. Then the relevant data is getting stored into the variable.

Line 162: Calculating the average rating a movie gets. Then the relevant data is getting stored into the variable.

Line 165-170: Calculating the number of reviews each genre got as well as the average rating it got. Then the relevant data is getting stored into the variables.

Line 172-181: Setting the “Genre Status” according to the question.

Line 183: Returning the relevant information.

Problems Faced:

- Took me a bit of time to figure out the correct logic to solve the problem.
- Figuring out queries from that new found logic was also a bit time consuming.

The queries I needed to solve the problem:

```
199   SELECT GEN_TITLE, COUNT(RATING.REV_ID), AVG(RATING.REV_STARS)
200   FROM GENRES , RATING , MTYPE
201   where GENRES.GEN_ID = MTYPE.GEN_ID AND MTYPE.MOV_ID = RATING.MOV_ID AND GENRES.GEN_ID = MTYPE.gen_id
202   GROUP BY GEN_TITLE;
203
204
205   Select floor(Sum(total_reviews)/count(total_reviews)) as Average_Reviews
206   from (SELECT g.GEN_TITLE, COUNT(r.REV_ID) as total_reviews
207   FROM RATING r
208   JOIN MTYPE mt ON r.MOV_ID = mt.MOV_ID
209   JOIN GENRES g ON mt.GEN_ID = g.GEN_ID
210   GROUP BY g.GEN_TITLE);
211
212
213   SELECT AVG(NVL(REV_STARS, 0)) FROM RATING;
```

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 .

The Solution:

```
217  -- 5
218
219  CREATE OR REPLACE FUNCTION get_most_frequent_genre (p_start_date DATE, p_end_date DATE)
220  RETURN VARCHAR2
221  AS
222  |  v_genre VARCHAR2(20);
223  |  v_count NUMBER;
224  BEGIN
225  |
226  |    SELECT gen_title, genre_count INTO v_genre, v_count FROM
227  |    (
228  |    |  /* Counting the number of movies that each genre had between the two dates */
229  |    |  SELECT g.gen_title, COUNT(*) as genre_count
230  |    |  FROM movie m, mtype mt, genres g
231  |    |  where  m.mov_id = mt.mov_id AND mt.gen_id = g.gen_id AND m.mov_releasedate
232  |    |  BETWEEN (p_start_date) AND (p_end_date)
233  |    |  GROUP BY g.gen_title
234  |    |  ORDER BY genre_count DESC
235  |    )
236  |    WHERE ROWNUM <= 1;
237  |
238  |  RETURN 'The Genre: ' ||  v_genre || ' (' || v_count || ')';
239  |
240  END;
241  /
242
243  SELECT get_most_frequent_genre(TO_DATE('01-JAN-1940', 'DD-MON-YYYY'),
244  |   |   |   |   |   |   |   |   TO_DATE('01-JAN-2021', 'DD-MON-YYYY'))
245  FROM DUAL;
```

Explanation & Analysis:

Line 229-234: Calculating the number of movies that each genre had between the given dates.

Line 226-236: Selecting the genre with the most number of movies in that given time. Then the relevant data is getting stored into the variables.

Line 238: Returning the relevant information.

Line 243-245: “TO_DATE” function is used to convert the date value specified in any string kind of datatypes such as varchar, varchar2, char, or char2 to the “DATE” data type

Problems Faced:

- Took me a bit of time to figure out the correct logic behind the solution.
- Figuring out queries from that new found logic was also a bit time consuming.

The queries I needed to solve the problem:

```
244  SELECT g.gen_title, COUNT(*) as genre_count
245  FROM movie m, mtype mt, genres g
246  where  m.mov_id = mt.mov_id AND mt.gen_id = g.gen_id AND m.mov_releasedate
247  BETWEEN TO_DATE('01-JAN-1940', 'DD-MON-YYYY') AND TO_DATE('01-JAN-2021', 'DD-MON-YYYY')
248  GROUP BY g.gen_title
249  ORDER BY genre_count DESC;
250
251  SELECT * FROM
252  (
253  |  SELECT g.gen_title, COUNT(*) as genre_count
254  |  FROM movie m, mtype mt, genres g
255  |  where  m.mov_id = mt.mov_id AND mt.gen_id = g.gen_id AND m.mov_releasedate
256  |  BETWEEN TO_DATE('01-JAN-1940', 'DD-MON-YYYY') AND TO_DATE('01-JAN-2021', 'DD-MON-YYYY')
257  |  GROUP BY g.gen_title
258  |  ORDER BY genre_count DESC
259  )
260  where ROWNUM <= 1;
```