Islamic University of Technology

Department of Computer Science

Database Management Systems II Lab

Course Code- CSE 4410

Prepared By:
Nazia Krim Khan Oishee
ID: 200042137

Date: January 28, 2023

Contents

1 Introduction

In our third lab of DBMS II our task was to implement procedures and functions through PL/SQL to acquire a better understanding of those.

2 PL/SQL

PL/SQL stands for Procedural Language for Structured Query Language. The language allows us to write code and deploy it in the database nearest to data. PL/SQL simplifies application development, optimize execution, and improve resource utilization in the database. The language is case-insensitive programming language, like SQL.

PL/SQL is a blocked programming language. Program units can be named or unnamed blocks. Unnamed blocks are known as anonymous blocks. Procedures and functions fall in the category of named block.

3 Procedure in PL/SQL

A procedure is a module that performs one or more actions. Procedures are key building blocks of modular code, which allow us to reuse and manage our program logic. Procedures have their associate formal parameter list through which we move data out and into procedure. The parameters cab be of IN, OUT or IN OUT type.

4 Function in PL/SQL

A function is a module that returns data through its RETURN clause, rather than in an OUT or IN OUT parameter.

Both procedures and functions have a name, can take parameters, return values and be called from other blocks. The difference is that a function must return a value, but in a procedure it is optional.

5 Tasks

We were given total 5 tasks in the lab based on the schema.

Task1: 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.

Solution

```
CREATE OR REPLACE
PROCEDURE Show_time( name IN MOVIE.MOV_TITLE%TYPE )
AS
duration number;
div integer;
time number;
BEGIN
SELECT MOV_TIME INTO duration
```

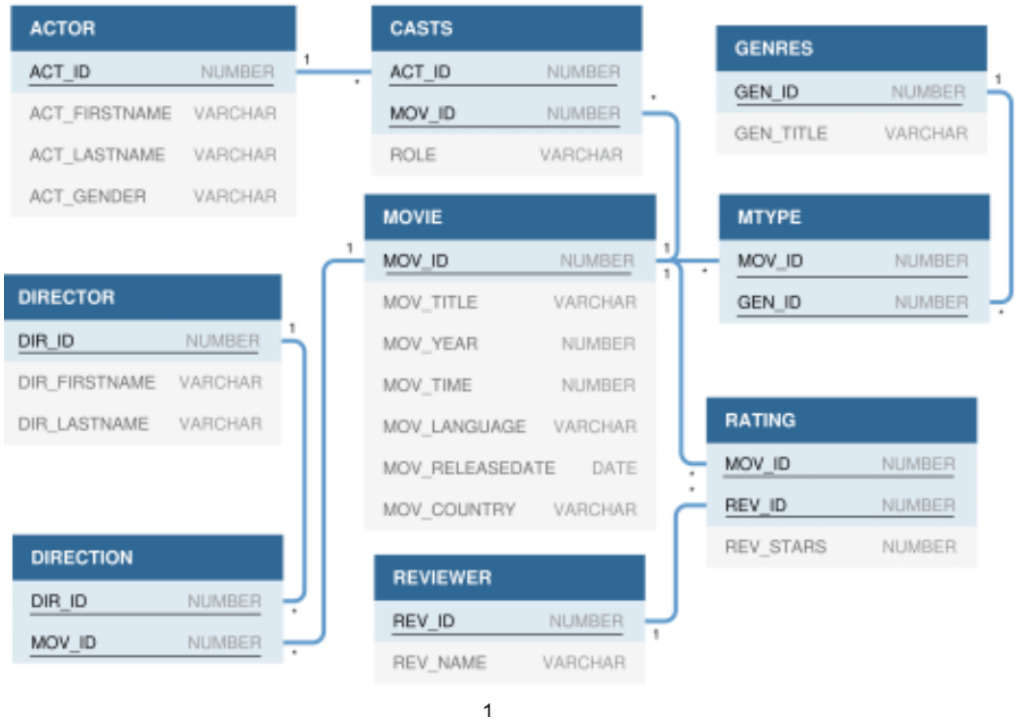

Figure 5.1: Movie Schema

```
FROM MOVIE
WHERE  MOV_TITLE=name ;
div := MOD(duration,70);
IF (div>30) THEN
time:=duration+ Floor((duration/70))*15;
ELSE
time:=duration+ (Floor((duration/70))-1)*15;
END IF;
DBMS_OUTPUT.PUT_LINE (Floor(time/60) || ' hour ' || MOD(time,60)||' minutes
    ');
END ;
/
-- Call it from an anonymous block
SET SERVEROUTPUT ON SIZE 1000000
SET VERIFY OFF
DECLARE
Movie_Name MOVIE.MOV_TITLE%TYPE;
BEGIN
Movie_Name := '&moviename';
Show_time(Movie_Name);
END ;
/
```

Explanation

In this task, I created a procedure named Show_time which takes a movie name as IN parameter, which means the parameter can be read only.

Then using SELECT statement I selected the duration of the movie and then calculated the required time to show the movie using conditional statement.The procedure displays the show time.

I called the procedure from an anonymous block. The block takes a movie name as an in-

put from the user and passes it to the procedure.

I used SERVEROUTPUT to show the result in the console.

Task2: 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.

Solution

```
CREATE OR REPLACE
PROCEDURE Show_Top_N_Movies( N IN NUMBER )
AS
REVIEW RATING.REV_STARS%TYPE;
TITLE  MOVIE.MOV_TITLE%TYPE;
mov_num number;
BEGIN
SELECT COUNT(MOV_ID) into mov_num FROM MOVIE;
IF(mov_num<N) THEN
DBMS_OUTPUT.PUT_LINE('ERROR');
RETURN;
END IF;
FOR i IN (SELECT TITLE FROM (SELECT avg(REV_STARS) AS REVIEW,MOV_TITLE AS
   TITLE FROM RATING natural join MOVIE GROUP BY MOV_TITLE ORDER BY REVIEW
   DESC) WHERE ROWNUM<=N)  LOOP
DBMS_OUTPUT.PUT_LINE(i.TITLE);
 END LOOP;
END ;
/
-- Call it from an anonymous block
SET SERVEROUTPUT ON SIZE 1000000
SET VERIFY OFF
DECLARE
N NUMBER;
BEGIN
N:='&N';
Show_Top_N_Movies(N);
END ;
/
```

Explanation

In this task, I created a procedure which takes a number as input.

Then I selected the total number of movies in the movie schema using COUNT aggregate function. If the input is greater than the total count, then the procedure shows an error message and terminates the procedure. Else the procedure selects the top rated movies by joining RATING and MOVIE tables.And to accomplish this task, I used explicit cursor in FOR loop.

Task3: 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 per year in between current date and release date) of a movie that is obtained from user review.

Solution

```

SET SERVEROUTPUT ON SIZE 1000000
```

```
CREATE OR REPLACE FUNCTION GET_YEARLY_INCOME(movie_title IN MOVIE.MOV_TITLE%
   TYPE)
RETURN NUMBER
AS
Yearly_Income NUMBER;
Release_Year NUMBER;
Rev_Count NUMBER;
Years_Passed NUMBER;
Current_Year NUMBER;
BEGIN
  SELECT EXTRACT(YEAR FROM (MOV_RELEASEDATE)) INTO Release_Year FROM MOVIE
   WHERE MOV_TITLE=movie_title;
  SELECT COUNT(*) INTO Rev_Count FROM MOVIE natural join RATING WHERE
   MOV_TITLE=movie_title AND RATING.REV_STARS>6;
  SELECT EXTRACT(Year FROM SYSDATE) INTO Current_Year FROM Dual;
  Years_Passed:=Current_Year-Release_Year;
  RETURN TRUNC((Rev_Count*10)/Years_Passed,4);
END;
/

DECLARE
    movie_title movie.mov_title%TYPE;
BEGIN
    movie_title:='&movie_title';
    DBMS_OUTPUT.PUT_LINE('Yearly Income ' || GET_YEARLY_INCOME(movie_title))
   ;
END;
/
```

Explanation

I created a function which takes a movie name as input and returns the yearly income as output.

At first I extracted the year from the RELEASEDATE. Then I subtracted it from current year to calculate the number of passed years. I counted the number of reviews that movie got where rating were more than 6 by joining RATING and MOVIE tables.

Then I performed the instructed operation in the task to calculate yearly income.

The function GET_YEAR LY_INCOME returns this calculated yearly income.

Task4: Write a function, that given a genre (gen_id) will return genre status, additionally the review count and average rating of that genre.

Solution

```
SET SERVEROUTPUT ON SIZE 1000000
CREATE OR REPLACE FUNCTION GENRE_INFO(id IN mtype.gen_id%TYPE)
RETURN varchar2
AS
  avg_rev_count number;
  avg_rating number;
  gen_count number;
  rev_count number;
  curr_rating number;
BEGIN
  select avg(rev_stars) into avg_rating from mtype,rating where mtype.mov_id
   =rating.mov_id;
```

```
  select count(rev_id) into avg_rev_count from mtype,rating where mtype.
   mov_id=rating.mov_id;
  select count(distinct gen_id) into gen_count from mtype;
  avg_rev_count:=avg_rev_count/gen_count;
  select avg(rev_stars) into curr_rating from mtype,rating where mtype.
   mov_id=rating.mov_id AND mtype.gen_id=id;
  select count(rev_id) into rev_count from mtype,rating where mtype.mov_id=
   rating.mov_id AND mtype.gen_id=id;
  IF(curr_rating<avg_rating AND rev_count>avg_rev_count) THEN
    return 'Widely Watched, Review Count: '||rev_count||', Average Rating:
   '||TRUNC(curr_rating,2);
  ELSIF (curr_rating>avg_rating AND rev_count<avg_rev_count) THEN
    return 'High Rated, Review Count: '||rev_count||', Average Rating: '||
   TRUNC(curr_rating,2);
  ELSIF (curr_rating>avg_rating AND rev_count>avg_rev_count) THEN
    return 'Peoples favorite, Review Count: '||rev_count||', Average Rating:
    '||TRUNC(curr_rating,2);
  ELSE
    return 'So so, Review Count: '||rev_count||', Average Rating: '||TRUNC(
   curr_rating,2);
  END IF;

END;
/

DECLARE
    gen_id mtype.gen_id%TYPE;
BEGIN
    gen_id:='&something';
    DBMS_OUTPUT.PUT_LINE(GENRE_INFO(gen_id));
END;
/
```

Explanation

I created a function that takes genre id, a number as input and return genre status, review count and average rating of that genre.

At first I calculated average rating across all movies by joining MTYPE and RATING tables based on MOV_ID.Then I counted the total number of reviews a movie got. Then I divided the counted number by number of existing genres. Thus we get the average review a movie got across all the genres.

After that I counted the average rating of the given genre as curr_rating and number of reviews that genre got as rev_count. Then I compared the average rating with the curr_rating and average review with the rev_count. Bases on the comparison I allocated a genre, a genre status.

The function returns this genre status along with rev_count and curr_rating.

Task5: Write a function, that given two dates will return the most frequent genre of that time (accord- ing to movie count) along with the count of movies under that genre which had been released in the given time range .

Solution

```
SET SERVEROUTPUT ON SIZE 1000000
```

```
CREATE OR REPLACE FUNCTION MOST_POPULAR_GENRE(START_DATE IN MOVIE.
   MOV_RELEASEDATE%TYPE , END_DATE IN MOVIE.MOV_RELEASEDATE%TYPE)
RETURN varchar2
AS
  count_mov number;
  genre GENRES.GEN_TITLE%TYPE;
  genre_id GENRES.GEN_ID%TYPE;
BEGIN
    SELECT id INTO genre_id FROM (SELECT MTYPE.gen_id AS id,count(
   MOVIES_IN_GivenInterval.MOV_ID) AS movie_count FROM (
    SELECT * FROM MOVIE WHERE MOVIE.MOV_RELEASEDATE BETWEEN START_DATE AND
   END_DATE)MOVIES_IN_GivenInterval,MTYPE
    WHERE MOVIES_IN_GivenInterval.MOV_ID=MTYPE.MOV_ID
    GROUP BY MTYPE.gen_id
    ORDER BY movie_count DESC)
    WHERE ROWNUM<=1;

    SELECT GEN_TITLE INTO genre FROM GENRES WHERE GEN_ID=genre_id;
    SELECT count(MOVIE.MOV_ID) INTO count_mov FROM MOVIE,MTYPE WHERE MOVIE.
   MOV_ID=MTYPE.MOV_ID AND MTYPE.gen_id=genre_id AND MOVIE.MOV_RELEASEDATE
   BETWEEN START_DATE AND END_DATE;
  RETURN genre||', count of movies released: '||count_mov;
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(Most_POPULAR_GENRE(TO_DATE(start_date),TO_DATE(
   end_date)));
END;
/
```

Explanation

I created a function, that takes two dates as input and returns the most frequent genre of that time (accord- ing to movie count) along with the count of movies under that genre which had been released in the given time range.

At first I collected the all the information of the movies that were released in the given interval into MOVIES_IN_GivenInterval. From there I counted the number of movies under each genre. The genre with the highest number of movies is the most frequent genre of that interval.

Then I counted the number of movies under that most frequent genre of that interval from MOVIE and MTYPE.

The function returns the most frequent genre along with the number of movies of that genre within the time interval.

6 Problems I faced

Due to my less knowledge and expertise over PL/SQL, I struggled a bit with the syntax especially that of explicit cursors. It took me a bit of a time to get the logic also. I could complete the first two tasks in the lab hour itself. But I completed the remaining tasks afterwards. I found the last two tasks more difficult than others. It took a lot of time for me to understand and implement queries for those tasks.

I also discovered that I still need to learn how to implement natural join in PL/SQL more efficiently. I found some errors while implementing natural join. I look forward to learn that.