Islamic University of Technology (IUT)

Report on Lab 09

Submitted By

Shanta Maria (ID:200042172)

CSE 4308 Database Management Systems Lab

Submitted To

Md. Bakhtiar Hasan

Lecturer, Department of CSE

Zannatun Naim Srsity

Lecturer, Department of CSE

November 9, 2022

Introduction

In the lab class, we were given two main tasks to solve using PL/SQL to understand the basics of the programming language.

1 Task 1

Write PL/SQL statements to perform the following tasks:

1. **Warm-up**:
   (a) Print your name.
   (b) Take your student ID as input and print its length.
   (c) Take two numbers as input and print their product.
   (d) Print the current system time in 12-hour format.
   (e) Take a number as input and print whether it is a whole number or a fraction (with and without CASE statement).
   (f) Write a procedure that takes a number as argument and prints whether it is a composite number or not.

1.1 Solution

```
---Warm-up Tasks 1---
-----1(a)-----
declare
    my_name varchar2(20);
begin
    my_name := '& my_name';
    dbms_output.put_line('My name is ' || my_name);
    exception
    when others then
        dbms_output.put_line(sqlerrm);
end;
```

```
/

------1(b)------

declare

    my_id varchar2(20);

begin

    my_id := '& my_id';

    dbms_output.put_line('Length of ID is ' || length(my_id));

exception

    when others then

        dbms_output.put_line(sqlerrm);

end;

/

-----1(c)-----

declare

    num1 number;

    num2 number;

begin

    num1 := '& num1';

    num2 := '& num2';

    dbms_output.put_line('Product of num1 and num2 is ' || num1*num2);

exception

    when others then

        dbms_output.put_line(sqlerrm);

end;

/

-----1(d)-----
```

```
declare
    curr_date date := sysdate;
begin
    dbms_output.put_line(to_char(curr_date, 'DD-MON-YY HH12:MI:SS PM'));
exception
    when others then
        dbms_output.put_line(sqlerrm);
end;
/

-----1(e)-----
-----with case-----
declare
    num number;
begin
    num := '& num';
    dbms_output.put('The number ' || num || ' is a ' );

    case true
        when round(num, 0)=num then
            dbms_output.put_line('whole number.');
        else
            dbms_output.put_line('fractional number.');
    end case;
end;
/

-----without case-----
declare
```

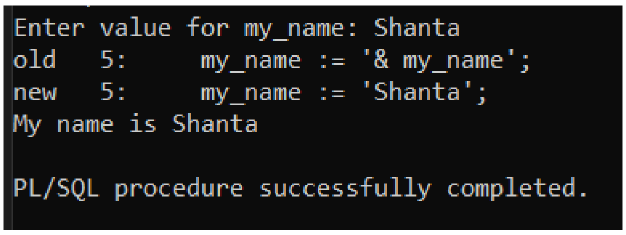
```
    num number;
begin
    num := '& num';
    dbms_output.put('The number ' || num || ' is a ' );

    if(round(num, 0)=num)
        then dbms_output.put_line('whole number.');
    else
        dbms_output.put_line('fractional number.');
    end if;
end;
/

-----1(f)-----
-----procedure-----
create or replace
procedure check_prime(num in number, msg out varchar2)
as
begin
    msg := ' is a prime number.';

    for i in 2..round(sqrt(num)) loop
        if mod(num, i)=0
            then msg := ' is a composite number.';
            exit;
        end if;
    end loop;
end;
/
```

```
----checking procedure-----
declare
    num number;
    msg varchar2(20);
begin
    num := 23;
    check_prime(num, msg);
    dbms_output.put_line(num || msg);
end;
/
```

1.2 Analysis and Explanation

——Task 1(a)——

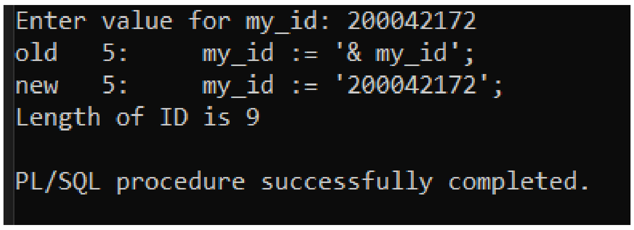
This task was straightforward and easy to solve. I declared a variable `my_name` where I stored the user input and then printed the result using `dbms_output.put_line()`. At the end, I put an exception block for catching any results which are not of type `varchar2` or exceeds the `varchar2` character limit.



```
Enter value for my_name: Shanta
old   5:      my_name := '& my_name';
new   5:      my_name := 'Shanta';
My name is Shanta

PL/SQL procedure successfully completed.
```

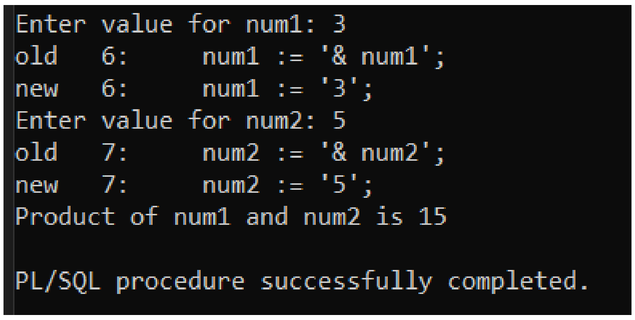
——Task 1(b)——

This task was also easy to solve. I declared a variable to store the user input ID and then printed the length of the ID using the `length()` function. An exception block was also put at the end to show any errors.


```
Enter value for my_id: 200042172
old   5:     my_id := '& my_id';
new   5:     my_id := '200042172';
Length of ID is 9

PL/SQL procedure successfully completed.
```

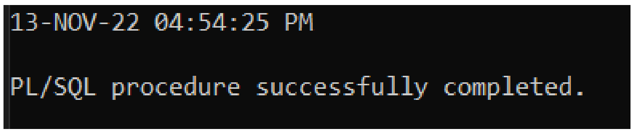
——Task 1(c)——

For this task, I declared two variables to store the two user input numbers. Then I used `'*'` operator to calculate the product of the numbers and printed the result. An exception block was also put here at the end to catch any errors.


```
Enter value for num1: 3
old   6:     num1 := '& num1';
new   6:     num1 := '3';
Enter value for num2: 5
old   7:     num2 := '& num2';
new   7:     num2 := '5';
Product of num1 and num2 is 15

PL/SQL procedure successfully completed.
```

——Task 1(d)——

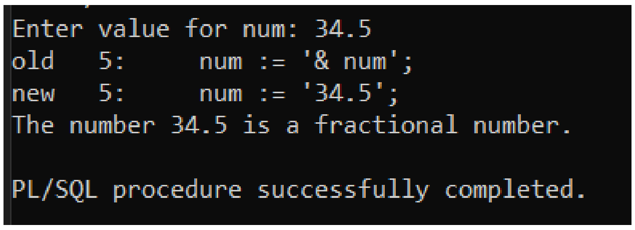
At first I declared a variable of type `date` then initialized it with the system date using `sysdate` statement. Then I printed the variable by specifying the date format to `'DD-MON-YY HH12:MI:SS PM'` where `HH12` and `PM` indicates the 12-hour format. An exception block is written at the end to catch any errors.


```
13-NOV-22 04:54:25 PM

PL/SQL procedure successfully completed.
```

——Task 1(e)——

——with case——

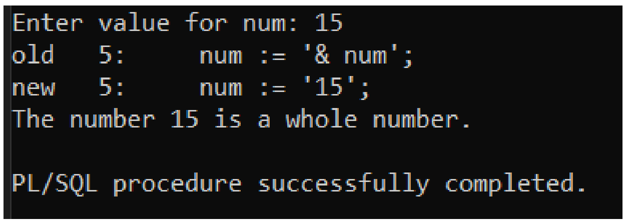
I declared a variable at first which stores the user input number. Then I used `case` statement to check if the number was fractional or whole. The `round()` function was used to check if the number was the same after rounding it or not. If it was the same then it meant that the number is a whole number otherwise it was a fractional number. The output was printed using `dbms_output.put_line()` after checking each condition. At the end of the `case` statements, the `case` is closed using `end case`.



```
Enter value for num: 34.5
old   5:     num := '& num';
new   5:     num := '34.5';
The number 34.5 is a fractional number.

PL/SQL procedure successfully completed.
```

——without case——

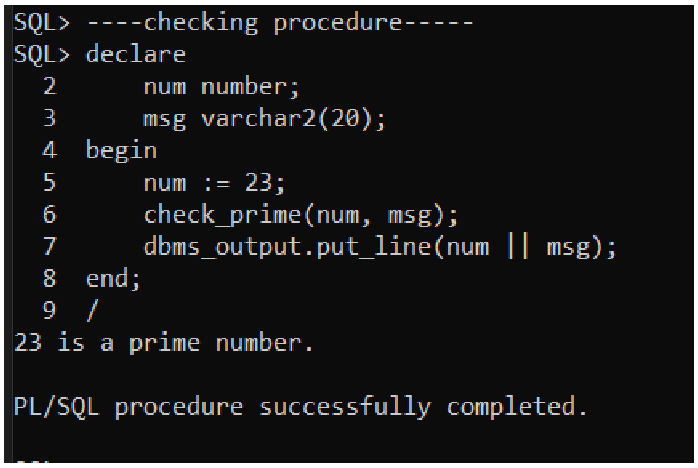
Without the case statement, the code is almost exactly the same. I replaced the condition checking statement using `if {condition}` instead of `case true when {condition}`. At the end of the `if` statements, it is closed using `end if` just like in `case`.



```
Enter value for num: 15
old   5:     num := '& num';
new   5:     num := '15';
The number 15 is a whole number.

PL/SQL procedure successfully completed.
```

——Task 1(f)——

For this task, I created a procedure named `check_prime` which takes a number as input and outputs a string message. At first, I initialized the return variable with `'is a prime number'`. Then a for loop was used from 1 to the rounded square root of the input number. This is to increase efficiency of the loop. For every iteration, it was checked if the input number was divisible by the iteration number of the loop. This was done using `mod()` function which returns 0 if the number is divisible. If the number was divisible then the message variable was changed to `'is a composite number'` and the loop was terminated. The procedure was then checked by calling it using the number 23 and the output was printed.



```
SQL> ----checking procedure-----
SQL> declare
  2       num number;
  3       msg varchar2(20);
  4  begin
  5       num := 23;
  6       check_prime(num, msg);
  7       dbms_output.put_line(num || msg);
  8  end;
  9  /
23 is a prime number.

PL/SQL procedure successfully completed.
```

1.3 Difficulties

This task was easily solved and I did not face any mentionable issues when solving it.

2 Task 2

(a) Write a procedure to find the *N* top-rated movies and their details. The procedure will take *N* as input and print the details upto *N* movies. If *N* is greater then the number of movies, then it will print an error message.

(b) Write a function to find the movie status ("Solo", "Ensemble"). If the total number of actors/actresses in a movie is 1, then the status should be "Solo", else it should be "Ensemble". The function will take the title of the movie as input as input and return the status.

(c) Write a procedure to find the possible nominees for Oscars. A director is eligible for Oscar if at least one of their movies have average rating of at least 7. Also, the movie should be reviewed by more than 10 reviwers.

(d) Write a function that will take the title of the movie as input and find the movie category based on Table 1.

Table 1: Movie Category Table for Question 2(d).

Movie Category	**Release Year**	**Average Rating**
Fantastic Fifties	1950 - 1959	>6.5
Sweet Sixties	1960 - 1969	>6.7
Super Seventies	1970 - 1979	>6.9
Ecstatic Eighties	1980 - 1989	>7.1
Neat Nineties	1990 - 1999	>7.3
Garbage	All other movies	

Write anonymous blocks to illustrate your programs, if needed.

2.1 Solution

```
 -----Task 2-----

-----2(a)-----

-----procedure-----

create or replace

procedure top_rated(n in number)

is

    row_num int;

    mov_id movie.mov_id%type;

    mov_title movie.mov_title%type;

    mov_year movie.mov_year%type;

    mov_language movie.mov_language%type;

    mov_releasedate movie.mov_releasedate%type;

    mov_country movie.mov_country%type;

    rev_stars rating.rev_stars%type;


    cursor find_movies(n number)

    is

        select *

        from

        (

            select mov_id, mov_title, mov_year, mov_language,

            mov_releasedate, mov_country, rev_stars

            from movie natural join rating

            where rev_stars>0

            order by rev_stars desc

        )

        where rownum<=n;
```

```
begin
    for row in (select count(*) as c from movie) loop
        if n>row.c then
            raise_application_error(-2000,
            'Number exceeds total number of movies!');
        else
            open find_movies(n);
            loop
                fetch find_movies into mov_id, mov_title, mov_year,
                mov_language, mov_releasedate, mov_country, rev_stars;
                exit when find_movies%notfound;
                dbms_output.put_line(row_num);
                dbms_output.put_line('Movie ID: ' || mov_id);
                dbms_output.put_line('Movie Title: ' || mov_title);
                dbms_output.put_line('Movie Year: ' || mov_year);
                dbms_output.put_line('Movie Language: ' || mov_language);
                dbms_output.put_line('Movie Release Date: ' || mov_releasedate);
                dbms_output.put_line('Movie Country: ' || mov_country);
                dbms_output.put_line('Rev Stars: ' || rev_stars);
                dbms_output.put_line('-----');
            end loop;
            close find_movies;
        end if;
    end loop;
end;
/

-----call-----
declare
```

```
    num int;
begin
    num := '& num';
    top_rated(num);
end;
/

-----2(b)-----
-----function-----
create or replace
function movie_status(mov_title in movie.mov_title%type)
return varchar2
is
    mov_status varchar2(10);
    cast_num int;

    cursor find_status(m_title in movie.mov_title%type)
    is
        select count(act_id) as c
        from movie natural join casts
        where mov_title=m_title
        group by mov_id;

begin
    open find_status(mov_title);
    loop
        fetch find_status into cast_num;
        exit when find_status%notfound;
        if cast_num=1 then
```

```
            mov_status :='Solo';
        else
            mov_status :='Ensemble';
        end if;
    end loop;
return mov_status;
end;
/

-----call-----
declare
    movie varchar2(25);
begin
    movie := '& movie';
    dbms_output.put_line(movie_status(movie));
end;
/

-----2(c)-----
-----procedure-----
create or replace
procedure oscar_nominees
is
    dir_id director.dir_id%type;
    dir_firstname director.dir_firstname%type;
    dir_lastname director.dir_lastname%type;

    cursor find_nominees
    is
```

```
        select dir_id, dir_firstname, dir_lastname
        from director natural join direction natural join movie
        -----oscar movies-----
        where mov_id in (select mov_id
                            -----number of reviewers-----
                            from (select mov_id, count(rev_id) as rev_c
                                    from movie natural join rating
                                    group by mov_id
                                    order by count(rev_id) desc)q1 natural join
                            -----avg rating-----
                                (select mov_id, avg(rev_stars) as rat_c
                                    from movie natural join rating
                                    group by mov_id
                                    order by count(rev_stars) desc)q2
                            where rev_c>10 and rat_c>=7);

begin
    open find_nominees;
    loop
        fetch find_nominees into dir_id, dir_firstname, dir_lastname;
        exit when find_nominees%notfound;
        dbms_output.put_line('Director ID: ' || dir_id);
        dbms_output.put_line('Director First Name: ' || dir_firstname);
        dbms_output.put_line('Director Last Name: ' || dir_lastname);
        dbms_output.put_line('-----');
    end loop;
    close find_nominees;
end;
/
```

```
-----call-----
begin
    oscar_nominees;
end;
/

-----2(d)-----
-----function-----
create or replace
function movie_category(mv_title in movie.mov_title%type)
return varchar2
is
    mov_category varchar2(50);
    mov_year number;
    mov_avg_rating number;
    m_title movie.mov_title%type;

    cursor find_category(mv_title in movie.mov_title%type)
    is
        select m, y, rat_c
        from (-----returns the years for the movies-----
                select mov_id, max(mov_title) as m,
                max(extract(year from mov_releasedate))as y
                from movie
                group by mov_id
                order by y)q1 natural join
            (-----returns the avergae rating for each movie-----
                select mov_id, max(mov_title) as m, avg(rev_stars) as rat_c
                from movie natural join rating
                group by mov_id
                order by count(rev_stars) desc)q2
        where m=mv_title;

begin
    open find_category(mv_title);
    loop
        fetch find_category into m_title, mov_year, mov_avg_rating;
        exit when find_category%notfound;

        if mov_year>=1990 and mov_year<=1999 and mov_avg_rating>7.3 then
            mov_category :='Neat Nineties';
        elsif mov_year>=1980 and mov_year<=1989 and mov_avg_rating>7.1 then
            mov_category :='Ecstatic Eighties';
        elsif mov_year>=1970 and mov_year<=1979 and mov_avg_rating>6.9 then
            mov_category :='Super Seventies';
        elsif mov_year>=1960 and mov_year<=1969 and mov_avg_rating>6.7 then
            mov_category :='Sweet Sixties';
        elsif mov_year>=1950 and mov_year<=1959 and mov_avg_rating>6.5 then
            mov_category :='Fantastic Fifties';
        else
            mov_category := 'Garbage';
        end if;
    end loop;
return mov_category;
end;
/
```

```
-----call------
declare
    movie varchar2(25);
begin
    movie := '& movie';
    dbms_output.put_line(movie_category(movie));
end;
/
```

2.2 Analysis and Explanation

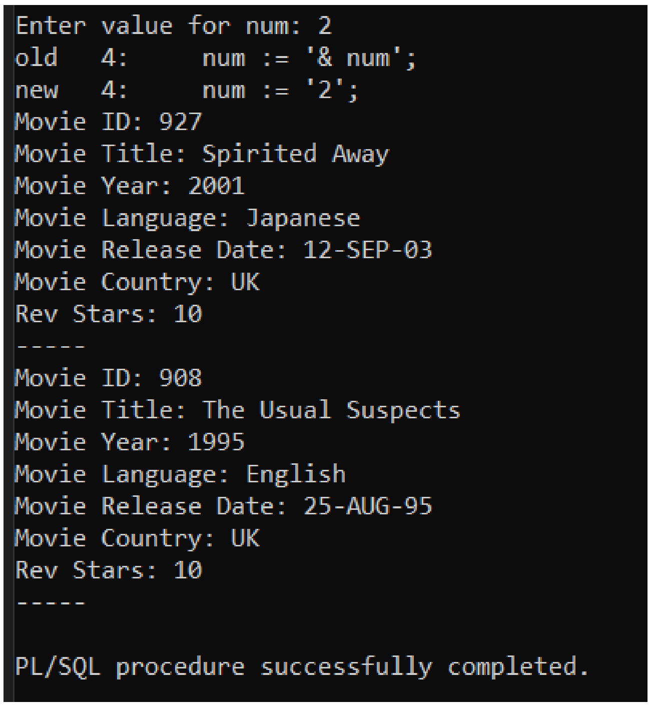
——Task 2(a)——

For this task, I created a procedure names `top_rated` which takes a number `n` as input. At first, I declared all the necessary variables that is each of the movie details which should be printed from the query. All the movie details variables were the same type as their respective columns in the table `movie`. Then I declared a cursor named `find_movies` which took the same input as the procedure.

The `SQL` query in the cursor returns all the necessary details of the top n movies from the movie table. It has a nested query which return all the movies with movie details ordered according to their rating from highest to lowest and disregarding any movies with no rating. Then from that nested query, the top n results are returned using `rownum`.

The exception for when the value of n exceeds the total number of movies is written next which basically checks if the value of n is greater than the number of records in movie table or not. If the value is not greater, then the results from the cursor is fetched into the previously declared variables using a loop and printed until all the results from the cursor are traversed.

To test the procedure, a call was made using a user input number as and then passing it to the procedure. The result for the top 2 rated movies are shown below.



```
Enter value for num: 2
old   4:     num := '& num';
new   4:     num := '2';
Movie ID: 927
Movie Title: Spirited Away
Movie Year: 2001
Movie Language: Japanese
Movie Release Date: 12-SEP-03
Movie Country: UK
Rev Stars: 10
-----
Movie ID: 908
Movie Title: The Usual Suspects
Movie Year: 1995
Movie Language: English
Movie Release Date: 25-AUG-95
Movie Country: UK
Rev Stars: 10
-----

PL/SQL procedure successfully completed.
```

—–Task 2(b)—–

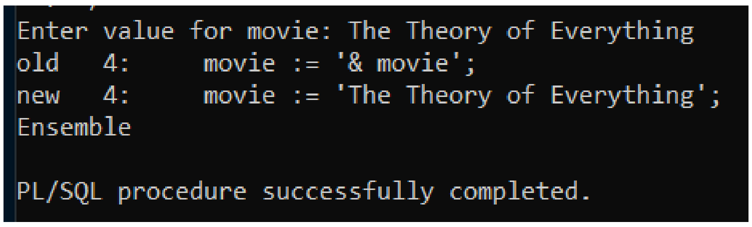
For this task, I created a function called `movie_status` which takes the movie title as input and return a string which would be either `'Solo'` or `'Ensemble'` depending on the number of actors and actresses in the movie from the casts table.

First, I declared two variables. One is to indicate the status of the movie which would be returned at the end of the function and the other variable is used to store the number of cast members for the specified movie from the cursor result.

The cursor created in the function returns the number of cast members for the movie using the `count()` function by matching the movie title with the input title. The returned result is stored in the variable declared beforehand. The `group by` clause ensures that all the `act_id` for that specific movie is counted and not just one record.

A loop is then used to traverse through the cursor result. An if statement is also used to check if the returned result is one or not. If it is one then the status would be set to `'Solo'` otherwise it would be set to `'Ensemble'`. The loop is ended when no more results from the cursor is found. The stored status is then returned using the variable.

The function was tested by calling it using user input as `'The Theory of Everything'` as shown below.



```
Enter value for movie: The Theory of Everything
old   4:     movie := '& movie';
new   4:     movie := 'The Theory of Everything';
Ensemble

PL/SQL procedure successfully completed.
```

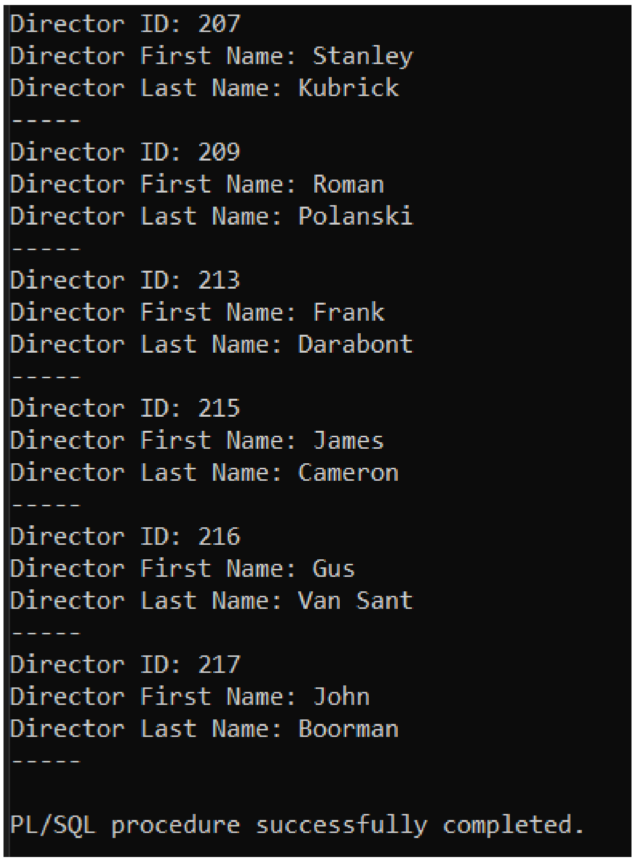
—–Task 2(c)—–

At first, I declared all the necessary variables of their respective data types in the director table. These variables would be printed later.

Then I created a cursor where I used three nested queries to get the desired results. First, I fetched all the information from the direction table which needed to be printed. Then I matched the `mov_id` of those directors with 2 nested queries. One query returns all the movies with their respective number of reviewers in descending order while the other query returns the movies and their average rating using the `avg()` function. The returned results are then checked using the conditions mentioned in the task and the resulting `mov_ids` are returned to be matched with those in the direction table.

A loop is then used to traverse through the cursor results and store the value of each record into the variables declared beforehand. The variable are then printed to show the details of the directors who are nominated for the oscars. The loop is ended when there are no more results returned by the cursor.

The procedure was called later to test its functionality. The results is as show below.



```
Director ID: 207
Director First Name: Stanley
Director Last Name: Kubrick
-----
Director ID: 209
Director First Name: Roman
Director Last Name: Polanski
-----
Director ID: 213
Director First Name: Frank
Director Last Name: Darabont
-----
Director ID: 215
Director First Name: James
Director Last Name: Cameron
-----
Director ID: 216
Director First Name: Gus
Director Last Name: Van Sant
-----
Director ID: 217
Director First Name: John
Director Last Name: Boorman
-----

PL/SQL procedure successfully completed.
```

—–Task 2(d)—–

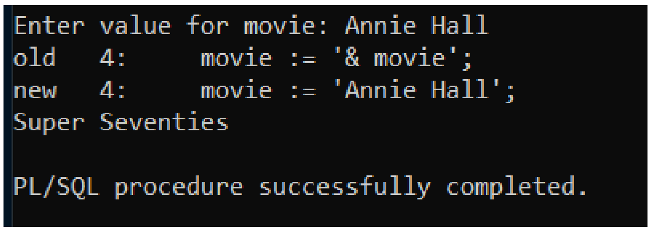
For this task, I created a function called `movie_category` which took the movie title as input and returned a string which would be the category.

First I declared all the necessary variables to store the cursor result and the return value into.

Then I created a cursor which also took the movie title as input. The SQL query in the cursor uses 2 nested queries. The first one returns all the movie years. This was done using the `extract()` function with year specified in the parameter. The other query returns the average rating for each movie which is calculated using the `avg()` function. Here `group by` clause is used to ensure that all the ratings for a single movie is taken into account when calculating the average. From these two nested queries, the movie title, the year and the average rating is selected and the movie title is matched with the input title.

A loop is then used to traverse through the cursor results. The values fetched from the cursor are stored in the variables declared beforehand and the cursor is traversed until no more records are found. While traversing the cursor, an if else conditional block checks the movie year and rating and assigns the movie to the appropriate category. This category is stored in a varchar2 variable which is then returned.

The function is tested by calling it and printing the result with the input as `'Annie Hall'` as show below.



```
Enter value for movie: Annie Hall
old   4:     movie := '& movie';
new   4:     movie := 'Annie Hall';
Super Seventies

PL/SQL procedure successfully completed.
```

2.3 Difficulties

For task `2(a)`, I faced error with the SQL query when using `rownum` because the details were not nested at first. After nesting the query which returned the details and then using `rownum` since the nested query meant that it was calculated first and then temporarily stored. I also faced diffculties in implementing the exception because I found it hard to understand.

For task `2(b)`, I did not face any mentionable problems or difficulties.

For task`2(c)`, I found it difficult to find all the conditional information using nested queries. At first I tried finding all the information using a single query but it showed a lot of errors and some of the information calculated using aggregate functions could not be compared. That is why I used several nested queries to overcome the errors.

For task `2(d)`, I found it very difficult to figure out how to extract the year from the date datatype as it showed a lot of error for different functions. It was also difficult to figure out how to take the nested query results and how to link the nested queries together. The query also showed a lot of error for the variables in the select statement because of the `group by` clause.

Conclusion

As shown in the report, I have solved and tested the solutions for all the tasks given in the lab. All the commands used were written in VS Code which was then saved with .sql extension. The .sql file was then run through the SQL command line to execute all the commands. User input was taken using the `SQL` command line.