Student Name: **A Z Hasnain Kabir**
Student ID: **200042102**

**CSE 4410**
**Database Management Systems II**
**Lab 3**

---

## Introduction

In this lab, we explored various SQL queries on a database containing tables for movie, reviewer, rating, mtype, direction, etc. The following are the tasks and queries that were completed

## Task 1

Write a procedure that will take a mov_title and show the required time (–hour –minute) to play that movie in a cinema hall. Let's say, there will be an intermission of 15 minutes after every 70 minutes only if the remaining movie time is greater than 30 minutes.

### Code

```
CREATE OR REPLACE PROCEDURE FIND_TIME(TITLE IN VARCHAR2)
IS
    time MOVIE.MOV_TIME%TYPE;
    total_time MOVIE.MOV_TIME%TYPE;
BEGIN
    SELECT MOV_TIME INTO TIME FROM MOVIE WHERE MOV_TITLE = TITLE;
    total_time := TIME + trunc(TIME/70) * 15;

    IF (MOD(TIME, 70) < 30) THEN

        total_time := total_time - 15;

    END IF;

    DBMS_OUTPUT.PUT_LINE(TRUNC(total_time/60)|| ' hrs and '|| TRUNC(MOD(total_time, 60))
        || ' minutes ');
END;
/

set serveroutput on;
begin
    FIND_TIME('Blade Runner');
end;
/
```

### Difficulties

Not using truncate can return values that we cannot use for hours and minutes.

## Task 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 up to N movies. If N is greater then the number of movies, then it will print an error message.

### Code

```
CREATE OR REPLACE PROCEDURE FIND_TOP_RATED(N IN NUMBER)
IS
begin
    FOR I IN
    (SELECT * FROM (SELECT MOV_TITLE, AVG(REV_STARS) AS REV
    FROM MOVIE NATURAL JOIN RATING
    GROUP BY MOV_TITLE ORDER BY REV DESC) WHERE ROWNUM<=N) LOOP
        DBMS_OUTPUT.PUT_LINE(I.MOV_TITLE || ' ' || I.REV);
    END loop;
end;
/

set serveroutput on;

begin
    FIND_TOP_RATED(10);
end;
/
```

### Difficulties

In order to use rownum after results are ordered a nested query is needed as rownum cannot be used using having clause after group by clause.

## Task 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 earnings (total earnings/year between the current date and release date) of a movie that is obtained from user reviews.

### Code

```
    CREATE OR REPLACE FUNCTION YEARLY_EARNING(TITLE VARCHAR2)
RETURN NUMBER
IS
    EARNING NUMBER;
    YEARS NUMBER(8,2);
    RELEASE_DATE MOVIE.MOV_RELEASEDATE%TYPE;
```

```
    YEARLY_EARNING NUMBER(8,2);
BEGIN
    FOR I IN
    (SELECT RATING.REV_STARS, MOVIE.MOV_RELEASEDATE
    FROM MOVIE NATURAL JOIN RATING WHERE MOV_TITLE = TITLE)
    LOOP
        IF(I.REV_STARS >= 6) THEN
            EARNING:= EARNING+10;
        END IF;
    RELEASE_DATE:= I.MOV_RELEASEDATE;
    END LOOP;
    SELECT MONTHS_BETWEEN(SYSDATE, RELEASE_DATE) / 12 INTO YEARS FROM DUAL;
    YEARLY_EARNING := EARNING / YEARS;
RETURN YEARLY_EARNING;
END;
/

set serveroutput on;

begin
    DBMS_OUTPUT.PUT_LINE(YEARLY_EARNING('Blade Runner'));
end;
/
```

### Difficulties

Table name has to be clearly mentioned in case of calling table attributes in query.

## Task 4

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

### Code

```
CREATE OR REPLACE FUNCTION GENRE_STATUS(GENID GENRES.GEN_ID%TYPE)
RETURN VARCHAR2 AS
    T_AVG_RATING FLOAT;
    T_REV_COUNT NUMBER;
    REVIEW_COUNT NUMBER;
    AVG_RATING FLOAT;

BEGIN
    SELECT COUNT(*), AVG(RATING.REV_STARS) INTO T_REV_COUNT, T_AVG_RATING
    FROM GENRES, MTYPE, RATING
    WHERE GENRES.GEN_ID = MTYPE.GEN_ID AND MTYPE.MOV_ID = RATING.MOV_ID;
    SELECT COUNT(*), AVG(RATING.REV_STARS) INTO REVIEW_COUNT, AVG_RATING
    FROM GENRES, MTYPE, RATING
    WHERE GENRES.GEN_ID = MTYPE.GEN_ID AND MTYPE.MOV_ID = RATING.MOV_ID AND GENRES.GEN_ID
        = GENID;

```

```
    IF (REVIEW_COUNT > T_REV_COUNT AND AVG_RATING < T_AVG_RATING) THEN
        RETURN 'WIDELY WATCHED';
    ELSIF(REVIEW_COUNT < T_REV_COUNT AND AVG_RATING > T_AVG_RATING) THEN
        RETURN 'HIGHLY RATED';
    ELSIF(REVIEW_COUNT > T_REV_COUNT AND AVG_RATING > T_AVG_RATING) THEN
        RETURN 'PEOPLES FAVOURITE';
    ELSE
        RETURN 'SO SO';
    END IF;
END;
/
begin
  DBMS_OUTPUT.PUT_LINE(GENRE_STATUS(903))
end;
/
```

### Difficulties

Difficult to understand the question.

## Task 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 that had been released in the given time range.

### Code

```
CREATE OR REPLACE TYPE MOVIE_TYPE AS OBJECT(
    GEN_TITLE VARCHAR2(100),
    NUM NUMBER
);
/

CREATE OR REPLACE TYPE MOVIE_TABLE AS TABLE OF MOVIE_TYPE;
/

CREATE OR REPLACE FUNCTION FREQUENT_GENRE(DATE1 DATE, DATE2 DATE)
RETURN MOVIE_TABLE
IS
    M MOVIE_TABLE;
BEGIN
    SELECT MOVIE_TYPE(GEN_TITLE, NUM) BULK COLLECT INTO M FROM(
        SELECT GEN_TITLE, count(MOV_ID) AS NUM FROM
        GENRES NATURAL JOIN MTYPE WHERE MOV_ID IN(
            SELECT MOV_ID FROM MOVIE WHERE MOVIE.MOV_RELEASEDATE BETWEEN DATE1 AND DATE2
        )GROUP BY GEN_TITLE ORDER BY NUM DESC
    ) WHERE ROWNUM<=1;

RETURN M;
END;
```

```
24 /
```

### Difficulties

Function cannot return more than one values so we had to create a multivalued attribute in order to complete this query.