Islamic University of Technology
Lab Task 04
CSE 4308 - DBMS Lab

Submitted To :

MD Bakhtiar Hasan
Ast. Professor, CSE Department
Islamic University of Technology

Zannatun Naim Sristy
Lecturer, CSE Department
Islamic University of Technology

Submitted By :

Sian Ashsad
ID : 200042151
Prog. : SWE
Dept. : CSE

Task 1 :

Working code :

```
--1--

SELECT DISTINCT ACT_FIRSTNAME, ACT_LASTNAME
FROM ACTOR, DIRECTOR
WHERE ACTOR.ACT_FIRSTNAME = DIRECTOR.DIR_FIRSTNAME AND ACTOR.ACT_LASTNAME =
DIRECTOR.DIR_LASTNAME;

--WITH INTERSECTION--
SELECT ACT_FIRSTNAME, ACT_LASTNAME
FROM ACTOR
INTERSECT
SELECT DIR_FIRSTNAME, DIR_LASTNAME
FROM DIRECTOR;
```

Report :

Analysis : In this task, we need to find out the names of the actors/actresses that are also directors with and without the 'intersect' clause. First we have to take the cross product of both the Actor and Director table. Then apply conditions to check the same first and last names.

Explanation of solution :

Without INTERSECT : The query chooses the first and last name of actors/actresses from the Actor table who have the same first and last name as the Directors in the Director table.

With INTERSECT : Shows the common first and last name of actors/actresses and directors from Actor and Director table respectively.

Findings : Using NATURAL JOIN instead of cross product produces similar results.

Task 2 :

Working code :

```
--2--

SELECT ACT_FIRSTNAME
FROM ACTOR
UNION ALL
SELECT DIR_FIRSTNAME
FROM DIRECTOR;
```

Report :

Analysis : In this task, we need to find the list of all the first names stored in the database.

Explanation of solution : Shows the first names including repetition from both Actor and Director tables.

Findings : using ALL after union allows repetition of the same first name. This allows us to show all first names stored in the database.

Problems : Not using ALL after Union doesn't give us repetition of the same first name. In which case, we do not get every single first name in the database.

Task 3 :

Working code :

```
--3--

SELECT DISTINCT MOV_TITLE
FROM MOVIE
WHERE MOVIE.MOV_ID NOT IN (SELECT MOV_ID FROM RATING);

--WITH MINUS--
SELECT MOV_TITLE
FROM MOVIE
MINUS
SELECT MOV_TITLE
FROM MOVIE, RATING
WHERE MOVIE.MOV_ID = RATING.MOV_ID;
```

Report :

Analysis : In this task, we need to find the movie titles that did not receive any ratings with and without 'minus' clause. We have to show the movies which are not present in the Rating table.

Explanation of solution :

Without MINUS : Selects distinct movie titles from movie table but sets the condition to not show titles which are not present in the Rating table by checking the movie id.

With MINUS : First choose all the movies from the Movie table. Then choose all the movies from the Movie table which are also in the Rating table by checking the movie id. Then minus the first query with the second to show the movies which did not receive any rating.

Task 4 :

Working code :

```
--4--

SELECT AVG(NVL(REV_STARS,0)) AS AVG_RATING
FROM RATING;
```

Report :

Analysis : We need to find the average rating of all movies. We have to iterate through all the ratings and find the average score.

Explanation of solution : Simply uses the AVG aggregate function to find the average from the Rating table. The NVL function registers any null values to 0. (Assuming null entries mean the rating was given 0).

Findings : Without NVL, the average rating is 6.8 whereas with NVL, the rating is ~ 6.56.

Task 5 :

Working code :

```
--5--

SELECT MOVIE.MOV_TITLE, MIN(NVL(REV_STARS,0)) AS MIN_STARS
FROM RATING NATURAL JOIN MOVIE
GROUP BY MOVIE.MOV_TITLE
ORDER BY MIN_STARS DESC;
```

Report :

Analysis : In this task, we need to find the minimum rating for each movie and display them in descending order of rating. We have to use an aggregate function to find the minimum rating for each movie.

Explanation of solution : The query selects title and minimum rating of each movie from Movie and Rating table. If there are any null ratings on the Rating table, that is considered as a zero.

Findings : Natural Join avoids redundant data upon joining tables. Without it, most of the movies get a minimum rating of zero.

Task 6 :

Working code :

```
--6--

SELECT ACTOR.ACT_LASTNAME, COUNT(REV_STARS) AS CNT_STARS
FROM (CASTS NATURAL JOIN ACTOR ) NATURAL JOIN RATING
GROUP BY ACTOR.ACT_LASTNAME
ORDER BY CNT_STARS;
```

Report :

Analysis : In this task, we need to find the last name of actors/actresses and the number of ratings received by the movies that they played a role in. For this task we need to use the COUNT aggregate function.

Explanation of solution : Here the query selects the last name of actors from the Actor table and counts the number of ratings received in the Rating table through the Casts table. The query forms a Natural Join based on the actor id from the Casts and Actor tables. Then forms another join based on the movie id from the Casts and Rating tables. Finally it groups the joins by the last name of the Actor table and shows it in ascending order through the order by clause.

Task 7 :

Working code :

```
--7--

SELECT ACT_LASTNAME, AVG(MOVIE.MOV_TIME)
FROM ACTOR, CASTS, MOVIE
WHERE ACTOR.ACT_ID = CASTS.ACT_ID AND CASTS.MOV_ID = MOVIE.MOV_ID
GROUP BY ACTOR.ACT_LASTNAME
MINUS
SELECT ACT_LASTNAME, AVG(MOVIE.MOV_TIME)
FROM ACTOR, CASTS, MOVIE, DIRECTION, DIRECTOR
WHERE ACTOR.ACT_ID = CASTS.ACT_ID AND CASTS.MOV_ID =
        MOVIE.MOV_ID AND MOVIE.MOV_ID = DIRECTION.MOV_ID AND
        DIRECTION.DIR_ID = DIRECTOR.DIR_ID AND
DIRECTOR.DIR_FIRSTNAME = 'James' AND DIRECTOR.DIR_LASTNAME = 'Cameron'
GROUP BY ACTOR.ACT_LASTNAME;

--WITH HAVING CLAUSE--
SELECT ACTOR.ACT_LASTNAME, AVG(MOVIE.MOV_TIME) AS AVG_TIME
FROM CASTS JOIN ACTOR ON CASTS.ACT_ID = ACTOR.ACT_ID
     JOIN MOVIE ON CASTS.MOV_ID = MOVIE.MOV_ID
     JOIN DIRECTION ON CASTS.MOV_ID = DIRECTION.MOV_ID
     JOIN DIRECTOR ON DIRECTION.DIR_ID = DIRECTOR.DIR_ID
GROUP BY ACTOR.ACT_LASTNAME,DIRECTOR.DIR_FIRSTNAME,DIRECTOR.DIR_LASTNAME
HAVING DIRECTOR.DIR_FIRSTNAME NOT IN ('James') AND DIRECTOR.DIR_LASTNAME NOT IN
('Cameron');
```

Report :

Analysis : We need to find the last name and average runtime of movies of different actors/actresses excluding any actor/actress who worked with ‘James Cameron’ (with and without ‘having’ clause). Here we can use the Minus clause to figure out the query.

Explanation of solution :

Without HAVING : First creates a query to find the last name of actors/actresses and average runtime of movies they partook. The AVG aggregate function is used to calculate average runtime. The second query does the same operation but for only movies directed by ‘James Cameron’. Using Minus between the first and second query gives us our solution.

With HAVING : Does the same operation as the second query mentioned earlier. The difference is that the HAVING clause excludes all records with first name ‘James’ and last name ‘Cameron’. The group by function is required to have two more attributes : DIR_FIRSTNAME and DIR_LASTNAME from the director table so that HAVING clause can perform the filter.

Findings : The same solution can be done using Natural Join.

Problems : Finding out the answer is difficult because there is too much data in the table provided.

Task 8 :

Working code :

```
--8--

SELECT A.FIRSTNAME, A.LASTNAME, A.AVG_STARS
FROM    (SELECT *
         FROM(  SELECT DIRECTOR.DIR_FIRSTNAME FIRSTNAME, DIRECTOR.DIR_LASTNAME LASTNAME,
MOVIE.MOV_TITLE, AVG(NVL(REV_STARS,0)) AS AVG_STARS
                FROM  (((DIRECTOR NATURAL JOIN DIRECTION) NATURAL JOIN MOVIE) NATURAL JOIN RATING)
                GROUP BY (DIRECTOR.DIR_FIRSTNAME, DIRECTOR.DIR_LASTNAME, MOVIE.MOV_TITLE)
                ORDER BY AVG_STARS DESC)) A
WHERE ROWNUM <= 1;




--WITH ALL--
SELECT ALL A.FIRSTNAME, A.LASTNAME, A.AVG_STARS
FROM    (SELECT *
         FROM(  SELECT ALL DIRECTOR.DIR_FIRSTNAME FIRSTNAME, DIRECTOR.DIR_LASTNAME LASTNAME,
MOVIE.MOV_TITLE, AVG(NVL(REV_STARS,0)) AS AVG_STARS
                FROM  (((DIRECTOR NATURAL JOIN DIRECTION) NATURAL JOIN MOVIE) NATURAL JOIN RATING)
                GROUP BY (DIRECTOR.DIR_FIRSTNAME, DIRECTOR.DIR_LASTNAME, MOVIE.MOV_TITLE)
                ORDER BY AVG_STARS DESC)) A
WHERE ROWNUM <= 1;
```

Report :

Analysis : In this task, we need to find the first name and last name of the director of the movie having the highest average rating (with and without ‘all’ clause). Nested queries are required for this task.

Explanation of solution : The query chooses first name, last name and average review form the A table created through nested query. The A table consists of the first name, last name, movie title and average rating of the movies. This table was created through the natural join of Director, Direction, Movie and Rating tables. The order was set to descending order to have the highest average rating at the top of the table. Lastly the rownum was restricted to 1. So the query shows only the first name, last name and average rating of the movie with the highest average rating.
The same operation was performed using the ALL clause.

Task 9 :

Working code :

```
--9--
SELECT MOVIE.MOV_ID, MOVIE.MOV_TITLE, MOVIE.MOV_YEAR,
MOVIE.MOV_LANGUAGE,MOVIE.MOV_RELEASEDATE,MOVIE.MOV_COUNTRY,MOV_TIME
FROM ACTOR, DIRECTOR, DIRECTION, MOVIE, CASTS
WHERE  (ACTOR.ACT_FIRSTNAME = DIRECTOR.DIR_FIRSTNAME AND ACTOR.ACT_LASTNAME =
DIRECTOR.DIR_LASTNAME) AND
        (DIRECTOR.DIR_ID = DIRECTION.DIR_ID AND
        DIRECTION.MOV_ID = MOVIE.MOV_ID) AND
        (ACTOR.ACT_ID = CASTS.ACT_ID AND
        CASTS.MOV_ID = MOVIE.MOV_ID);
```

Report :

Analysis : In this task, we need to find all the movie related information of movies acted and directed by the same person.

Explanation of solution : Checks if the first and last name of the actor with the director is the same. Then uses the Direction and Casts table to find the movies that person directed and acted.

Task 10 :

Working code :

```
--10--

SELECT MOVIE.MOV_TITLE, AVG(NVL(REV_STARS,0)) AS AVG_STARS
FROM RATING JOIN MOVIE ON RATING.MOV_ID = MOVIE.MOV_ID
GROUP BY MOVIE.MOV_TITLE
HAVING AVG(NVL(REV_STARS,0)) > 7;

--WITHOUT HAVING CLAUSE--

SELECT MOV_TITLE, AVG_RATING
FROM    (SELECT MOV_TITLE, AVG(NVL(REV_STARS,0)) AS AVG_RATING
        FROM MOVIE, RATING
        WHERE MOVIE.MOV_ID = RATING.MOV_ID
        GROUP BY MOVIE.MOV_TITLE)
WHERE AVG_RATING > 7;
```

Report :

Analysis : In this task, we need to find the title and average rating of the movies that have an average rating more than 7 (with and without using the 'having' clause).

Explanation of solution :

Using Having : First the query shows the title and average rating of movies based on the natural join of the Rating and Movie table. Afterward, uses the Having clause to only show the movies with average rating higher than 7.

Without Having : Uses nested query to avoid using Having clause. The above mentioned solution is implemented in the nested query and then on the outer query the condition is given in the where clause.

Task 11 :

Working code :

```
--11--

SELECT DISTINCT MOV_TITLE, AVG_RATING
FROM    (SELECT MOV_TITLE, AVG(NVL(REV_STARS,0)) AS AVG_RATING
        FROM MOVIE, RATING
        WHERE MOVIE.MOV_ID = RATING.MOV_ID
        GROUP BY MOVIE.MOV_TITLE)
WHERE AVG_RATING > (
        SELECT AVG(NVL(REV_STARS,0)) AS AVG_RATING
        FROM RATING);
```

Report :

Analysis : In this task, we need to find the title of the movies having average rating higher than the average rating of all the movies.

Explanation of solution : Same method as task 10. Just the condition in where clause has been changed to find the total average rating of all the movies.

Task 12 :

Working code :

```
--12--

SELECT DISTINCT MOV_TITLE,(SELECT AVG(NVL(REV_STARS,0))
                           FROM RATING
                           WHERE MOVIE.MOV_ID = RATING.MOV_ID)
FROM MOVIE, RATING
WHERE MOVIE.MOV_ID = RATING.MOV_ID;
```

Report :

Analysis : In this task, we need to find the title and average rating of the movies without using the group by statement.

Explanation of solution : Uses nested query to avoid using group by. The nested query is performed in the Select clause of the outer query to select average ratings of each movie. Then the outer query shows the selected attributes from Movie and Rating tables.

Findings : Natural join can not be used here as the nested query has conditions to check from the Movie table which is not in the nested query.

Task 13 :

Working code :

```
--13--

SELECT ACT_FIRSTNAME ,COUNT(*) AS Occurrence
FROM ACTOR
WHERE ACT_GENDER = 'F'
GROUP BY ACT_FIRSTNAME
HAVING COUNT(*)>1;
```

Report :

Analysis : In this task, we need to find the actresses with the same first name. For this task we have to specify the gender from the Actor table and find who has the same first name.

Explanation of solution : COUNT aggregate function is used to keep track of the occurrence of same first names. Where clause restricts the search amongst actresses. The group by clause allows us to find and count occurrences. The Having clause allows only multiple occurrences to show.

Task 14 :

Working code :

```
--14--

SELECT MOV_ID,MOV_TITLE,MAX(REV_STARS) AS MAX_RATING
FROM (
        ((SELECT * FROM ACTOR WHERE ACT_GENDER='F') NATURAL JOIN CASTS)
        NATURAL JOIN
        (
                (SELECT MOV_TITLE,MOV_ID,COUNT(*) as OCCURRENCES
                 FROM MOVIE NATURAL JOIN RATING
                 GROUP BY MOV_ID,MOV_TITLE
                 HAVING COUNT(*)>=10)
                NATURAL JOIN
                (
                        DIRECTION
                        NATURAL JOIN
                                (SELECT DIR_ID
                                 FROM DIRECTOR NATURAL JOIN DIRECTION
                                 GROUP BY DIR_ID
                                 HAVING COUNT(DIR_ID)>=1)
                )
        )
        NATURAL JOIN
        (
                (SELECT *
                FROM RATING NATURAL JOIN REVIEWER
                WHERE REV_NAME='Neal Wruck' AND REV_STARS>4)
        )
    )
GROUP BY MOV_ID,MOV_TITLE
ORDER BY MAX_RATING DESC;
```

Report :

Analysis : In this task, we need to find the title and maximum rating of the movies that have at least 10 reviews and have a female actress. One of the reviewers of the movie should be ‘Neal Wruck’. We can not include any movie that received less than 4 stars rating or any movies from directors that have not directed more than one movie. A series of nested queries are required to complete this task.

Explanation of solution : The solution creates natural joins of queries relevant to the given conditions. Basically the whole query is an aggregation of sub queries that satisfies individual conditions.

Problems : This query took a long time to solve. The natural joins of different sub queries were tedious to make and often led to wrong answers.