# Lab 4
# Advanced Data Manipulation

CSE 4308

DATABASE MANAGEMENT SYSTEMS LAB

SEPTEMBER 1, 2023

# Contents

# 1 Distinct

The DISTINCT statement is used to return only distinct (different) values. Inside a table, a column often contains many duplicate values; and sometimes you only want to list the different (distinct) values. The syntax is following:

```
SELECT DISTINCT attributename
FROM tablename;
```

For example,

```
SELECT DISTINCT DEPT_NAME FROM EMPLOYEE;
```

# 2 Range Operator

## 2.1 BETWEEN Operator

The BETWEEN operator selects values within a given range. The values can be numbers, strings or dates. The BETWEEN operator is inclusive: begin and end values will be included. However, you have to provide the lower value first then the upper value.

```
SELECT attribute_1, .....
FROM tablename
WHERE attribute_1 BETWEEN l_value AND u_value;
```

For example,

```
SELECT NAME, DEPT_NAME, SALARY
FROM EMPLOYEE
WHERE SALARY BETWEEN 35000 AND 80000;
```

For excluding a range we can use NOT operator before BETWEEN. For example,

```
SELECT NAME, DEPT_NAME, SALARY
FROM EMPLOYEE
WHERE SALARY NOT BETWEEN 35000 AND 80000;
```

## 2.2 IN Operator

The IN operator allows one to specify multiple values in WHERE clause. It is a shorthand for multiple OR conditions.

```
SELECT attribute_1, .....
FROM tablename
WHERE attribute_1 IN (value1, value2, ...);
```

For example,

```
SELECT NAME, SALARY
FROM EMPLOYEE
WHERE DEPT_NAME in ('DEV', 'TESTING');
```

Similar to between, we can use NOT to exclude some values.

# 3 Rename

SQL aliases are used to give a table, or a column in a table, a temporary name. We can also use it for renaming an expression or an entire query. The primary purpose of using aliases is to make a column or table more readable. For example:
For example,

```
SELECT ID, NAME, (SALARY*12) AS Y_SALARY
FROM EMPLOYEE:
```

```
SELECT E.NAME, P,NAME
FROM EMPLOYEE E, (SELECT * FROM PROJECT) P
WHERE E.P_ID=P.P_ID;
```

Here using AS is optional for the expression, table and any query.

# 4 String Operator

## 4.1 LIKE Operator

In simple terms, the LIKE operator is used to match substrings in a query. It is used in a WHERE clause to search for a specified pattern in a column. There are two wildcards often used in conjunction with the LIKE operator:

- % to represent any substring.
- _ to represent any single character.

```
SELECT attribute_1, attribute_2 .....
FROM tablename
WHERE attribute_1 LIKE pattern;
```

Here, Patterns are case-sensitive. For example,

```
SELECT NAME
FROM EMPLOYEE
WHERE NAME LIKE 'A%';
```

So, It will show you all the names that start with capital A. For restricting the name size of the resultant name additionally, let's say 4 characters. We can use the following statement:

```
SELECT NAME
FROM EMPLOYEE
WHERE NAME LIKE 'A___';
```

## 4.2 Concatination

To concatenate multiple columns or any additional string with any column at the time of retrieving data, SQL supports string operator(||). For instance,

```
SELECT 'Employee NAME: '|| NAME
FROM EMPLOYEE;
```

It will show the term 'Employee NAME: ' as prefix of any name.

# 5 Null Value Handling

Null Values present special problems in relational operations, including arithmetic operations, comparison operations, and joining. Consider the following:

```
SELECT NAME, (SALARY+BONUS) AS TOTAL
FROM EMPLOYEE;
```

Employees who have null values for their bonus will return Null as total salary!! Solution: Use nvl() built-in. Example:

```
SELECT NAME, (SALARY+nvl(BONUS,0)) AS TOTAL
FROM EMPLOYEE;
```

Now the null values found will be treated as 0, so total salary is now meaningful. We can use nvl() also for string or date data type.

# 6 Set Operator

In practice, set operations are used to merge data from different sources where conditions vary in each case. In many cases, these work as an alternative method of using AND, OR and NOT operators.

## 6.1 UNION

Just like the role of the union operator in set theory, in SQL UNION also retrieves all the data vertically and keeps the common data only for once. For example, to find the employee name of 'DEV' or 'TESTING' department, we can write:

```
SELECT NAME FROM EMPLOYEE WHERE DEPT_NAME='DEV'
UNION
SELECT NAME FROM EMPLOYEE WHERE DEPT_NAME='TESTING';
```

## 6.2 INTERSECTION

For retrieving the common data from both tables one can use the INTERSECTION operator. Let's say, we want to retrieve those employee who works both for 'DEV' and 'TESTING'. Then,

```
SELECT NAME FROM EMPLOYEE WHERE DEPT_NAME='DEV'
INTERSECT
SELECT NAME FROM EMPLOYEE WHERE DEPT_NAME='TESTING';
```

# 7 Data Sorting

The ORDER BY keyword is used to sort the result-set in ascending or descending order. If one does not specify the sorting order, the ORDER BY keyword sorts the records in ascending order by default. Otherwise, to specify ascending or descending order of sorting, we use the keywords 'ASC' and 'DESC' respectively.

```
SELECT attribute_1, attribute_2 .....
FROM tablename
ORDER BY attribute_1 [ASC/DESC], .....;
```

For example,

```
SELECT NAME, SALARY
FROM EMPLOYEE
ORDER BY DEPT, SALARY DESC;
```

# 8 Aggregation Function

Aggregate functions are functions that take a collection (a set or multiset) of values as input and return a single value. SQL offers five built-in aggregate functions:

- Average: avg
- Minimum: min
- Maximum: max
- Total: sum
- Count: count

The input to sum and avg must be a collection of numbers, but the other operators can operate on collections of non-numeric data types, such as strings, as well. The syntax is following:

```
SELECT [avg/min/max/sum/count](attribute_1)
FROM EMPLOYEE;
```

For example,

```
SELECT avg(SALARY)
FROM EMPLOYEE;
```

## 8.1 Group By

There are circumstances where we would like to apply the aggregate function not only to a single set of tuples but also to a group of sets of tuples; we specify this wish in SQL using the GROUP BY clause. The attribute or attributes given in the group by clause are used to form groups. Let's say, now we want to know the average salary of each department. So, the statement will be:

```
SELECT DEPT, avg(SALARY)
FROM EMPLOYEE
GROUP BY DEPT;
```

**All selected attributes must be present either in aggregate functions or in group by clause, otherwise the query will be considered erroneous.** For example, the following query is incorrect as ID appears in the SELECT statement, but not in the GROUP BY clause:

```
SELECT DEPT, ID, avg(SALARY)
FROM EMPLOYEE
GROUP BY DEPT;
```

## 8.2 Having

The HAVING clause is used to specify conditions on groups rather than on tuples. For example, we might be interested in only those departments where the average salary of the employees is more than $5000. We cannot use the WHERE clause in this case because WHERE is used to specify conditions on a single set of tuples, whereas in this case, we are interested in finding a group of departments where the average salary is more than $5000.

```
SELECT DEPT, avg(SALARY)
FROM EMPLOYEE
GROUP BY DEPT
HAVING avg(SALARY)>5000;
```

**As was the case for the SELECT statement, any attribute that is present in the HAVING clause without being aggregated must appear in the GROUP BY clause, otherwise the query is treated as erroneous.**

# 9 Subquery

SQL provides a mechanism for nesting subqueries. A subquery is a select-from-where expression that is nested within another query. It can be placed at two places:

- In the WHERE clause of a SELECT statement. Example,

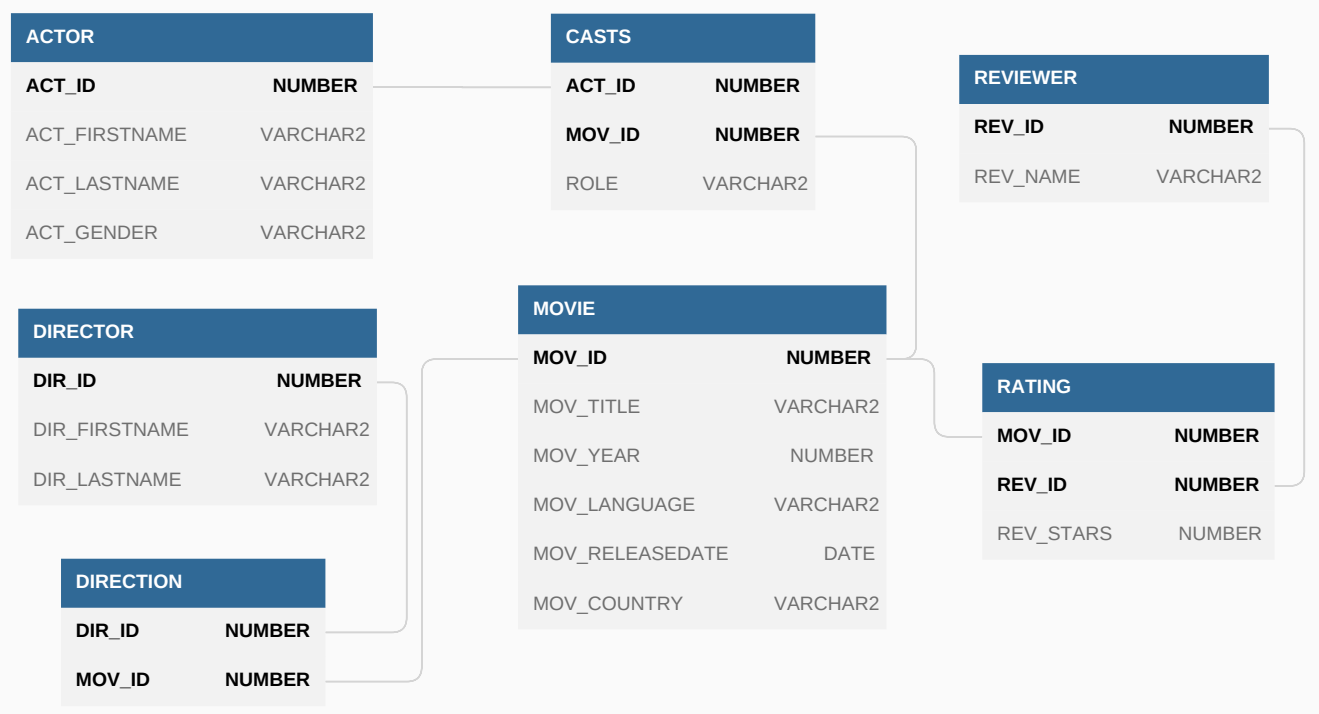
```
SELECT DEPT
FROM EMPLOYEE
WHERE SALARY>
    (SELECT avg(SALARY)
     FROM EMPLOYEE);
```

- In the FROM clause of a SELECT statement. The same result can be produced by using subquery in FROM clause as following:

```
SELECT DEPT
FROM EMPLOYEE E, (SELECT avg(SALARY) as A_SALARY
     FROM EMPLOYEE) A
WHERE E.SALARY>A.A_SALARY;
```

# 10 Task

Execute the `movie.sql` script using  command. It creates a set of tables along with values that maintain the following schema:




Here, the boldfaces denote the primary keys and the arcs denote the foreign key relationships.
In this lab, you have to write all SQL statements in an editor first and save them with .sql extension. Then execute the SQL script.
Write SQL statements for the following queries:

1. Find the name of the actors/actresses that are also directors (with and without set operator).
2. Find the actresses with the same first name.
3. Find the list of all the full names stored in the database.
4. Find the movie titles that did not receive any ratings.
5. Find the average rating of all movies.
6. Find the minimum rating for each movie and display them in descending order of rating.
7. Find the title of the movie having an average rev_star higher than the average rev_star of all the movies.
8. Find the name of actors/actresses and the number of ratings received by the movies in which they played a role.
9. Find the name of the director of the movie having the highest average rev_star.
10. Find all the movie-related information of movies acted and directed by the same person.
11. Find the title and average rating of the movies that have an average rev_star of more than 7.
12. Find the reviewer who gives the highest number of lowest rev_star.
13. Find the name and average runtime of movies of different actors/actresses. Do not include any actor/actress who worked with 'James Cameron'.