CMP020L012A	Data Analytics	2022/23

Lab Manual
Managing Data in RDBMS using SQL-2
Week-3

The contents will be focusing on various operators that are used to access data effectively by querying from any database table. The topic will also include various Clauses(s) and Function(s) we use for retrieving data from RDBMS.

There are Comparison Operators which can be directly used while performing any query on an existing database or, database table as below:

Operators	Meaning
=	Equal to
>	Greater than
>=	Greater than or Equal to
<	Less than
<=	Less than or Equal to
<>	Not Equal to

For example, now if it is asked to find the employee's name and job from the "Employee" table where the salary of the employee is less than 1000 then,

```
SQL> SELECT ename, job
     FROM Employee
     WHERE salary<1000;
```

The employee names and jobs will be displayed but only those of whose salaries are less than 1000.

***Use the comparison operators on the example tables that are provided.

Other comparison operators used while accessing data from the database table(s) are:

Operators	Meaning
BETWEEN…AND…	Between two values (Inclusive)
IN	Match any of the list values
LIKE	Match a character Pattern
IS NULL	Is a NULL value

Let's consider that, we have to find the employee's name and job and salary who has salaries between 1000 and 2000.

```
SQL> SELECT ename, job, salary
     FROM Employee
     WHERE salary BETWEEN 1000 AND 2000;
```

To find the department name and location for the sales and accounting departments.

```
SQL> SELECT name, location
     FROM department
     WHERE name IN ('SALES', 'ACCOUNTING');
```

The comparison operator "LIKE" is mainly used for character match operation, for example, if we want to see who are the employees, whose job title starts with letter "M", or, for the jobs which contain letter "A" in theirs title, or even the job titles which end with let's say letter "T". Try the examples below:

```
SQL> SELECT name, job
     FROM employee
     WHERE job LIKE "M%";
```

```
SQL> SELECT name, job
     FROM employee
     WHERE job LIKE “%A%”;
```

```
SQL> SELECT name, job
     FROM employee
     WHERE job LIKE “%T”;
```

More About “LIKE” operators:

LIKE Operator	Description
WHERE … LIKE 'a%'	Finds any values that start with "a"
WHERE … LIKE '%a'	Finds any values that end with "a"
WHERE … LIKE '%or%'	Finds any values that have "or" in any position
WHERE … LIKE '_r%'	Finds any values that have "r" in the second position
WHERE … LIKE 'a_%'	Finds any values that start with "a" and are at least 2 characters in length
WHERE … LIKE 'a__%'	Finds any values that start with "a" and are at least 3 characters in length
WHERE … LIKE 'a%o'	Finds any values that start with "a" and ends with "o"

Using “IS NULL” condition tests for the NULL values in the database table(s), for example, the employees whose “Commission” is NULL can be checked as,

```
SQL> SELECT name, salary, comm
     FROM employee
     WHERE comm IS NULL;
```

*****Remember that the Comparison Operators are contained in the “WHERE” block as condition(s).**

There are usage of Logical Operators as well while effective querying from any database table(s). Some of them basically works like “connectives” where there are more than one condition to satisfy or compare:

Operators	Meaning
AND	Returns TRUE if *both* component conditions are TRUE
OR	Returns TRUE if *either* component condition is TRUE
NOT	Returns TRUE if the following condition is FALSE

Now, if we want to see the information about the employees who have salaries more than 1000 and they also work as “clerk” then this is obvious that both the given conditions have to be satisfied together.

```
SQL> SELECT empno, name, salary, job
     FROM employee
     WHERE salary>1000
     AND job= ‘CLERK’;
```

Try to find the details of the employees whose job is either manager or clerk.

Let’s try another example where we will find the employees details whose commission is not NULL.

```
SQL> SELECT *
     FROM employee
     WHERE comm IS NOT NULL;
```

The Rules of precedence of using different operators are given here, though the rules can be overridden by using parenthesis:

Operators	Orders
All Comparison Operators	1
NOT	2
AND	3
OR	4

```
SQL> SELECT   name, job, salary
       FROM    employee
       WHERE (job='SALESMAN'
       OR      job='PRESIDENT')
       AND     salary>1500;
```

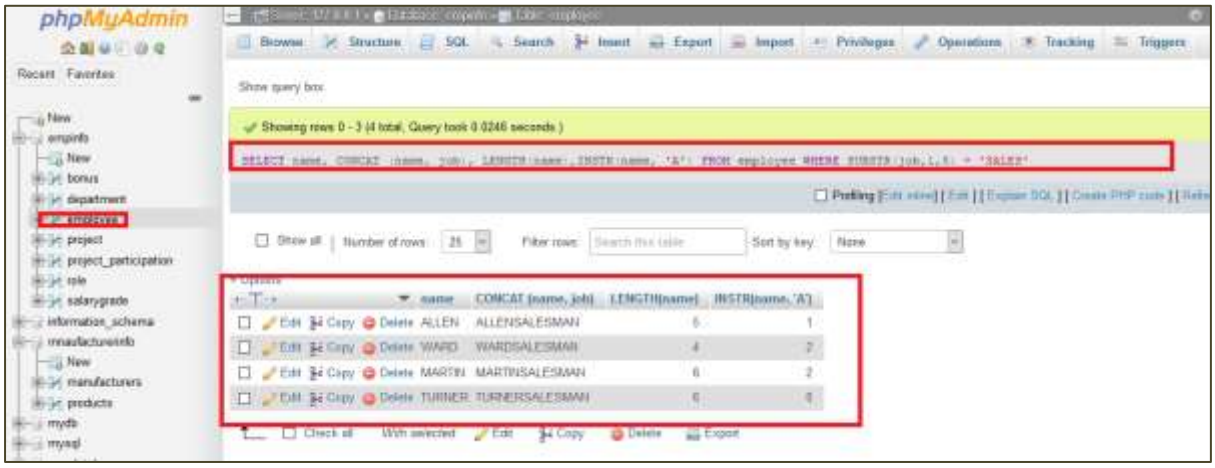
Here, the parenthesis enforced the “OR” condition to be executed first.

While querying some functions can be used to manipulate the character strings on the data stored in the database.

Functions	Description	Results
CONCAT('Good', 'String')	**Joins values together**	Good String
SUBSTR('String',1,3)	Extracts a string of determined length	Str
LENGTH('String')	Shows the length of a string as a numeric value	6
INSTR('String', 'r')	Finds numeric position of a named character	3
Trim(‘S’ from ‘SSMITH’)	Trims the exact character	MITH
Replace(‘toy’,’y’,’let’)	Does an replacement of character or a part of the string	TOLET

Let’s execute the following query to see the result:

```
SQL>SELECT name, CONCAT (name, job), LENGTH(name),INSTR(name, 'A')
FROM employee
WHERE SUBSTR(job,1,5) = 'SALES'
```



However, there is very useful feature of SQL, called “SQL Aliases”. Aliases improve the readability of the query report.

- Renames a column heading
- Is useful with calculations
- Immediately follows column name; optional AS keyword between column name and alias
- Requires double quotation marks if it contains spaces or special characters or is case sensitive

For example,

```
SQL>  SELECT name AS “Employee_name”, salary AS “Payroll”
          FROM  employee;
```

Another example,

```
SQL>  SELECT name AS Employee_name, (salary*12) AS “Annual Salary”
          FROM  employee;
```

Also, multiple columns/ attributes can be concatenated if necessary.

- Concatenates columns or character strings to other columns
- Is represented by two vertical bars (||)
- Creates a resultant column that is a character expression

```
SQL> SELECT name||job AS "Employees"
        FROM Employee;
```

Another Example,

```
SQL> SELECT name||' '||'is a'||' '||job AS "Employee Details"
        FROM   Employee;
```

Types of Aggregation Functions

AVG
COUNT
MAX
MIN
SUM

Each function accepts an argument and results accordingly on sets of rows, where appropriate.

AVG, SUM, MIN, and MAX functions are used against columns that can store numeric data. The example the average, highest, lowest, and sum of monthly salaries for the employees.

*** Remember the group functions are contained in the “select” block of SQL.

```
SQL> SELECT MAX(salary)
     FROM employee;
```

```
SQL> SELECT MIN(salary)
     FROM employee;
```

```
SQL> SELECT AVG(salary)
     FROM employee;
```

```
SQL> SELECT SUM(salary)
     FROM employee;
```

COUNT function returns the number of rows in a table, for instance, the number of departments present in the department table can be found using this group function.

```
SQL> SELECT COUNT(deptno)
     FROM department;
```

However, the number of departments can also be found from the employee table, but as multiple employees work in the same department, the number of departments can be repetitive. In that case, the query can be like:

```
SQL> SELECT COUNT( DISTINCT deptno)
        FROM employee;
```

*** ”DISTINCT” is a keyword, used within the select block to ensure that while accessing any data from a table, it avoids redundancy of a single value until the execution sustains.

Another example of “COUNT” function:

```
SQL> SELECT COUNT( empno)
        FROM employee
        WHERE deptno=10
```

Creating group data:

Until now, all group functions have treated the table as one large group of information. At times, you need to divide the table of information into smaller groups. This can be done by using the GROUP BY clause.

DEPTNO	SAL
---	---
10	2450
10	5000
10	1300
20	800
20	1100
20	3000
20	3000
20	2975
30	1600
30	2850
30	1250
30	950
30	1500
30	1250

In the example above, the employees salary data is provided as per the they are working in (each department is a group).

GROUP BY clause can be used to divide the rows in a table into groups. You can then use the group functions to return summary information for each group. Some important notes about “GROUP BY” clause:

- If a group function is included in a SELECT clause, selection of individual result can not be produced unless the individual column appears in the GROUP BY clause.
- Using a WHERE clause, the pre-exclution of rows before dividing them into groups can be done.
- The columns must be included in the GROUP BY clause.
- Column alias cannot be used in the GROUP BY clause.
- By default, rows are sorted by ascending order of the columns included in the GROUP BY list.

```
SQL> SELECT AVG (salary)
     FROM employee
     GROUP BY deptno;
```

The GROUP BY result can be excluded using “HAVING” clause:

- Rows are grouped.
- The group function is applied.
- Groups matching the HAVING clause are displayed.

```
SQL> SELECT deptno
     FROM employee
     GROUP BY deptno
     HAVING salary>1000;
```

It's a MUST to remember the order of "GROUP BY Function:

- WHERE clause
- GROUP BY clause
- HAVING clause

ORDER BY Function:

ORDER BY clause sorts row values in two different ways:

- ASC: ascending order, default
- DESC: descending order

The order of rows returned in a query result is undefined. The ORDER BY clause can be used to sort the rows. If the ORDER BY clause is used , it must be placed in the last line. ***Aliases can be used if needed.

```
SQL> SELECT hiredate, ename
        FROM employee
        ORDER BY hiredate;
```

To see the same result in Descending Order, the query can be written as,

```
SQL> SELECT hiredate, ename
        FROM employee
        ORDER BY hiredate DESC;
```