Database Lab

CSE 3104

Session 03

13 Operators

With the WHERE clause, the following operators can be used:

Operator	Description
=	Equal
<>	Not equal
>	Greater than
<	Less than
>=	Greater than or equal
<=	Less than or equal
BETWEEN	Between an inclusive range
LIKE	Search for a pattern
IN	If you know the exact value you want to return for at least one of the columns

Examples:

```
select * from CUSTOMER where AreaCode>30
```

13.1 LIKE Operator

The LIKE operator is used to search for a specified pattern in a column.

Syntax:

```
SELECT column_name(s)
FROM table_name
WHERE column_name LIKE pattern
```

The **SQL LIKE** clause is used to compare a value to similar values using **wildcard operators**. There are two wildcards used in conjunction with the LIKE operator:

→ The percent sign (%)

→ The underscore (_)

The percent sign represents zero, one, or multiple characters. The underscore represents a single number or character. The symbols can be used in combinations.

Syntax:

The basic syntax of % and _ is as follows:

```
SELECT FROM table_name
WHERE column LIKE 'XXXX%'

or

SELECT FROM table_name
WHERE column LIKE '%XXXX%'

or

SELECT FROM table_name
WHERE column LIKE 'XXXX_'

or

SELECT FROM table_name
WHERE column LIKE '_XXXX'

or

SELECT FROM table_name
WHERE column LIKE '_XXXX_'
```

You can combine N number of conditions using AND or OR operators. Here, XXXX could be any numeric or string value.

Example:

Here are number of examples showing WHERE part having different LIKE clause with '%' and '_' operators:

Statement	Description
WHERE SALARY LIKE '200%'	Finds any values that start with 200
WHERE SALARY LIKE '%200%'	Finds any values that have 200 in any position
WHERE SALARY LIKE '_00%'	Finds any values that have 00 in the second and third positions
WHERE SALARY LIKE '2_%_%'	Finds any values that start with 2 and are at least 3 characters in length
WHERE SALARY LIKE '%2'	Finds any values that end with 2
WHERE SALARY LIKE '_2%3'	Finds any values that have a 2 in the second position and end with a 3
WHERE SALARY LIKE '2___3'	Finds any values in a five-digit number that start with 2 and end with 3

Following is an example, which would display all the records from CUSTOMER table where SALARY starts with 200.

```
SELECT  *  FROM CUSTOMER WHERE SALARY LIKE '200%'
```

You may also combine with the **NOT** keyword (Logical Negation). The NOT operator reverses the meaning of the logical operator with which it is used. Eg: NOT EXISTS, NOT BETWEEN, NOT IN, etc. This is a negate operator.

```
SELECT  *  FROM CUSTOMER WHERE SALARY NOT LIKE '200%'
```

13.2 IN Operator

The IN operator allows you to specify multiple values in a WHERE clause.

Syntax:

```
SELECT column_name(s)
FROM table_name
WHERE column_name IN (value1,value2,...)
```

```
SELECT * FROM CUSTOMER WHERE AGE IN ( 25, 27 )
```

13.3 BETWEEN Operator

The BETWEEN operator selects a range of data between two values. The values can be numbers, text, or dates.

Syntax:

```
SELECT column_name(s)
FROM table_name
WHERE column_name
BETWEEN value1 AND value2
```

```
SELECT * FROM CUSTOMER WHERE AGE BETWEEN 23 AND 27
```

13.4 AND & OR Operators

The AND operator displays a record if both the first condition and the second condition is true.

The OR operator displays a record if either the first condition or the second condition is true.

Examples:

```
SELECT * FROM CUSTOMER WHERE AGE >= 25 AND SALARY >= 6500
```

```
SELECT * FROM CUSTOMER WHERE AGE >= 25 OR SALARY >= 6500
```

You can also combine AND and OR (use parenthesis to form complex expressions).

```
SELECT * FROM CUSTOMER WHERE NAME LIKE 'Ka%' AND (AGE >= 25 OR SALARY >= 6500)
```

14 SELECT TOP Clause

The TOP clause is used to specify the number of records to return.

The TOP clause can be very useful on large tables with thousands of records. Returning a large number of records can impact on performance.

Syntax:

```
SELECT TOP number|percent column_name(s)
FROM table_name
```

```
SELECT TOP 3 * from CUSTOMER
```

```
SELECT TOP 60 percent * from CUSTOMER
```

15 Built-in Functions

SQL has many built-in functions for performing calculations on data.

We have 2 categories of functions, namely **aggregate** functions and **scalar** functions. Aggregate functions return a single value, calculated from values in a column, while scalar functions return a single value, based on the input value.

Aggregate functions - examples:

- **AVG()** - Returns the average value
- **STDEV()** - Returns the standard deviation value
- **COUNT()** - Returns the number of rows
- **MAX()** - Returns the largest value
- **MIN()** - Returns the smallest value
- **SUM()** - Returns the sum
- etc.

Scalar functions - examples:

- **UPPER()** - Converts a field to upper case
- **LOWER()** - Converts a field to lower case
- **LEN()** - Returns the length of a text field
- **ROUND()** - Rounds a numeric field to the number of decimals specified
- **GETDATE()** - Returns the current system date and time
- etc.

16 The Group By Statement

Aggregate functions often need an added GROUP BY statement.

The GROUP BY statement is used in conjunction with the aggregate functions to group the result-set by one or more columns.

Syntax

```
SELECT column_name, aggregate_function(column_name)
FROM table_name
WHERE column_name operator value
GROUP BY column_name
```

GROUP BY clause is used in collaboration with the SELECT statement to arrange identical data into groups.

If we try the following:

```
SELECT Age, MAX(Salary) FROM CUSTOMER
```

We will get the following error message

```
Msg 8120, Level 16, State 1, Line 1
Column 'CUSTOMER.Age' is invalid in the select list because it is not
contained in either an aggregate function or the GROUP BY clause.
```

The solution is to use GROUP BY:

```
SELECT Age, MAX(Salary) FROM CUSTOMER GROUP BY Age
```

17 The Having Clause

The full syntax of the SELECT statement is complex, but the main clauses can be summarized as:

```
SELECT
[ ALL | DISTINCT ]
   [TOP ( expression ) [PERCENT] [ WITH TIES ] ]
select_list [ INTO new_table ]
[ FROM table_source ] [ WHERE search_condition ]
[ GROUP BY group_by_expression ]
[ HAVING search_condition ]
[ ORDER BY order_expression [ ASC | DESC ] ]
```

If we try the following:

```
SELECT Age, MAX(Salary) FROM CUSTOMER GROUP BY Age HAVING NAME LIKE 'Ka%'
```

We will get the following error message

```
Msg 8121, Level 16, State 1, Line 1
Column 'CUSTOMER.Name' is invalid in the HAVING clause because it is not
contained in either an aggregate function or the GROUP BY clause.
```

```
SELECT Age, MAX(Salary) FROM CUSTOMER GROUP BY Age HAVING Age >=25
```