Republic of Iraq
The Ministry of Higher Education
& Scientific Research
Al-Ma'arif University College
Department of Computer Science

Lecturer Name: Shihab Hamad Khaleefah
Academic Status: BhD. In Computer Science
Qualification: - Lecturer
Course Material: Database Management System
Grade Level: Second Stage, Lec. 8-10

THE CONIDITION OPERATORS & FUNCTION OPERATORS IN STRUCTURED QUERY LANGUAGE (SQL)

THE CONIDITION OPERATORS

A. SQL – Operators

What is an Operator in SQL?

An operator is a reserved word or a character used primarily in an SQL statement's WHERE clause to perform operation(s), such as comparisons and arithmetic operations. These Operators are used to specify conditions in an SQL statement and to serve as conjunctions for multiple conditions in a statement.

- Arithmetic operators
- Comparison operators
- Logical operators
- Operators used to negate conditions

1. SQL Arithmetic Operators

Arithmetic operators can perform arithmetical operations on numeric operands involved. Arithmetic operators are addition (+), subtraction (-), multiplication (*) and division (/). The + and - operators can also be used in date arithmetic.

Operator	Meaning
+ (Add)	Addition
- (Subtract)	Subtraction
* (Multiply)	Multiplication
/ (Divide)	Division
% (Modulo)	Returns the integer remainder of a division. For example, 17 % 5 = 2 because the remainder of 17 divided by 5 is 2.

Syntax:

```
SELECT <Expression>[arithmetic operator]<expression>...
 FROM [table_name]
 WHERE [expression];
```

Parameter	Description
Expression	Expression made up of a single constant, variable, scalar function, or column name and can also be the pieces of a SQL query that compare values against other values or perform arithmetic calculations.
arithmetic operator	Plus(+), minus(-), multiply(*), and divide(/).
table_name	Name of the table.

Assume **'variable a'** holds 10 and **'variable b'** holds 20, then:

Operator Description Example

+ Addition - Adds values on either side of the operator.

a + b will give 30

-Subtraction - Subtracts right hand operand from left hand operand.

a - b will give -10

* Multiplication - Multiplies values on either side of the operator.

a * b will give 200

/ Division - Divides left hand operand by right hand operand.

b / a will give 2

% Modulus - Divides left hand operand by right hand operand and returns remainder.

b % a will give 0

Arithmetic Operators – Examples

Here are a few simple examples showing the usage of SQL Arithmetic Operators:

Example 1:

```
select 10+ 20;
```

Output:

```
+--------+
| 10+ 20 |
+--------+
| 30 |
+--------+
1 row in set (0.00 sec)
```

Example 2:

```
select 10 * 20;
```

Output:

```
+---------+
| 10 * 20 |
+---------+
| 200 |
+---------+
1 row in set (0.00 sec)
```

Example 3:

```
select 10 / 5;
```

Output:

```
+--------+
| 10 / 5 |
+--------+
| 2.0000 |
+--------+
1 row in set (0.03 sec)
```

Example 4:

SQL

25

```
select 12 % 5;
```

Output:

```
+---------+
| 12 % 5 |
+---------+
| 2 |
+---------+
1 row in set (0.00 sec)
```

Example: SQL Arithmetic Operators

This is a simple example of using SQL arithmetic operators:

SELECT 15+10-5*5/5
FROM dual;

SQL plus (+) operator

The SQL plus **(+)** operator is used to add two or more expressions or numbers.

Example:

Sample table: customer

Cust_name	Opening_amt	Receive_amt
Ali	7000.00	7000.00
Ahmad	9000.00	10000.00
Aziz	5000.00	11000.00

To get data of 'cust_name', 'opening_amt', 'receive_amt', ('opening_amt' + 'receive_amt') from the 'customer' table with following condition :

1. sum of 'opening_amt' and 'receive_amt' is greater than 15000,

the following SQL statement can be used :

```
SELECT cust_name, opening_amt, receive_amt, (opening_amt +
receive_amt)
FROM customer
WHERE (opening_amt + receive_amt)>15000;
```

Cust_name	Opening_amt	Receive_amt	(Opening_amt + Receive_amt)
Ahmad	9000.00	10000.00	19000.00
Aziz	5000.00	11000.00	16000.00

SQL minus (-) operator

The SQL minus (-) operator is used to subtract one expression or number from another expression or number.

Example:

To get data of 'cust_name', 'opening_amount', 'payment_amount' and 'oustanding_amount' from the 'customer' table with following condition -

1. 'outstanding_amt' - 'payment_amt' is equal to the 'receive_amt',

the following SQL statement can be used:

```
SELECT cust_name,opening_amt, payment_amt, outstanding_amt
FROM customer
WHERE(outstanding_amt-payment_amt)=receive_amt;
```

Cust_name	Opening_amt	Receive_amt	Payment_amt	Outstanding_amt
Ali	7000.00	10000.00	8000.00	10000.00
Ahmad	9000.00	14000.00	10000.00	25000.00
Aziz	5000.00	12000.00	11000.00	12000.00

Output:

Cust_name	Opening_amt	Receive_amt	Payment_amt	Outstanding_amt
Ali	7000.00	10000.00	8000.00	10000.00
Aziz	5000.00	12000.00	11000.00	12000.00

SQL multiply (*) operator

The SQL multiply (*) operator is used to multiply two or more expressions or numbers.

Cust_name	Opening_amt	Receive_amt	Payment_amt	Outstanding_amt
Ali	7000.00	10000.00	8000.00	10000.00
Ahmad	9000.00	14000.00	10000.00	25000.00
Aziz	5000.00	12000.00	11000.00	12000.00

To get data of ' CUST_NAME ', and (' payment_amt'*2) from the 'agents' table with following condition -

1. two times of the default ' payment_amt' is greater than 144000.00,

the following SQL statement can be used :

```
SELECT CUST_NAME , (payment_amt *2)
FROM agents
WHERE (payment_amt *2)> 144000.00;
```

Cust_name	Opening_amt	Receive_amt	Payment_amt	Outstanding_amt
Ahmad	9000.00	14000.00	10000.00	25000.00
Aziz	5000.00	12000.00	11000.00	12000.00

SQL divide (/) operator

The SQL divide (/) operator is used to divide one expressions or numbers by another.

Example:

To get data of 'cust_name', 'opening_amt', 'receive_amt', 'outstanding_amt' and ('receive_amt'*5/ 100) as a column heading 'commission' from the customer table with following condition :

1. 'outstanding_amt' is less than or equal to 4000,

The following SQL statement can be used :

```
SELECT cust_name, opening_amt, receive_amt,
outstanding_amt, (receive_amt*5/ 100) commission
FROM customer
WHERE outstanding_amt <=4000;
```

Cust_name	Opening_amt	Receive_amt	Payment_amt	Outstanding_amt
Ali	7000.00	10000.00	8000.00	10000.00
Ahmad	9000.00	14000.00	10000.00	25000.00
Aziz	5000.00	12000.00	11000.00	12000.00

Output:

Cust_name	Opening_amt	Receive_amt	Payment_amt	Outstanding_amt	Commission
Ali	7000.00	10000.00	8000.00	10000.00	500

SQL modulo (%) operator

The SQL MODULO operator returns the remainder (an integer) of the division.

Example:

To get the modulus of a division of 150 by 7 from the customer table, the following SQL statement can be used :

SELECT 10000.00 %7;

Output:

4.00

2. SQL Comparison Operators

Assume **'variable a'** holds 10 and **'variable b'** holds 20, then:

Operator Description Example

= Checks if the values of two operands are equal or not, if yes then condition becomes true.
(a = b) is not true.

!= Checks if the values of two operands are equal or not, if values are not equal then condition becomes true.
(a != b) is true.

<> Checks if the values of two operands are equal or not, if values are not equal then condition becomes true.
(a <> b) is true.

> Checks if the value of left operand is greater than the value of right operand, if yes then condition becomes true.
(a > b) is not true.

< Checks if the value of left operand is less than the value of right operand, if yes then condition becomes true.
(a < b) is true.

>= Checks if the value of left operand is greater than or equal to the value of right operand, if yes then condition becomes true.
(a >= b) is not true.

<= Checks if the value of left operand is less than or equal to the value of right operand, if yes then condition becomes true.
(a <= b) is true.

!< Checks if the value of left operand is not less than the value of right operand, if yes then condition becomes true.
(a !< b) is false.
!> Checks if the value of left operand is not greater than the value of right operand, if yes then condition becomes true.
(a !> b) is true.

Comparison Operators – Examples

Consider the CUSTOMERS table having the following records:

SELECT * FROM CUSTOMERS;

```
+----+----------+-----+-----------+----------+
| ID | NAME     | AGE | ADDRESS   | SALARY   |
+----+----------+-----+-----------+----------+
| 1  | Ramesh   | 32  | Ahmedabad | 2000.00  |
| 2  | Khilan   | 25  | Delhi     | 1500.00  |
| 3  | Kaushik  | 23  | Kota      | 2000.00  |
| 4  | Chaitali | 25  | Mumbai    | 6500.00  |
| 5  | Hardik   | 27  | Bhopal    | 8500.00  |
| 6  | Komal    | 22  | MP        | 4500.00  |
| 7  | Muffy    | 24  | Indore    | 10000.00 |
+----+----------+-----+-----------+----------+
```

7 rows in set (0.00 sec)

Here are some simple examples showing the usage of SQL Comparison Operators:

Example 1:

SELECT * FROM CUSTOMERS WHERE SALARY > 5000;

Output:

```
+----+----------+-----+---------+----------+
| ID | NAME     | AGE | ADDRESS |   SALARY |
+----+----------+-----+---------+----------+
| 4  | Chaitali | 25  | Mumbai  |  6500.00 |
| 5  | Hardik   | 27  | Bhopal  |  8500.00 |
| 7  | Muffy    | 24  | Indore  | 10000.00 |
+----+----------+-----+---------+----------+
```

3 rows in set (0.00 sec)

Example 2:

SELECT * FROM CUSTOMERS WHERE SALARY = 2000;

Output:

```
+----+---------+-----+-----------+---------+
| ID | NAME    | AGE | ADDRESS   | SALARY  |
+----+---------+-----+-----------+---------+
| 1  | Ramesh  | 32  | Ahmedabad | 2000.00 |
| 3  | kaushik | 23  | Kota      | 2000.00 |
+----+---------+-----+-----------+---------+
```

2 rows in set (0.00 sec)

Example 3:

SELECT * FROM CUSTOMERS WHERE SALARY != 2000;

Output:

```
+----+----------+-----+---------+----------+
| ID | NAME     | AGE | ADDRESS | SALARY   |
+----+----------+-----+---------+----------+
| 2  | Khilan   | 25  | Delhi   | 1500.00  |
| 4  | Chaitali | 25  | Mumbai  | 6500.00  |
| 5  | Hardik   | 27  | Bhopal  | 8500.00  |
| 6  | Komal    | 22  | MP      | 4500.00  |
| 7  | Muffy    | 24  | Indore  | 10000.00 |
+----+----------+-----+---------+----------+
```

5 rows in set (0.00 sec)

Example 4:

```
SELECT * FROM CUSTOMERS WHERE SALARY <> 2000;
```

Output:

```
+----+----------+-----+---------+----------+
| ID | NAME     | AGE | ADDRESS | SALARY   |
+----+----------+-----+---------+----------+
| 2  | Khilan   | 25  | Delhi   | 1500.00  |
| 4  | Chaitali | 25  | Mumbai  | 6500.00  |
| 5  | Hardik   | 27  | Bhopal  | 8500.00  |
| 6  | Komal    | 22  | MP      | 4500.00  |
| 7  | Muffy    | 24  | Indore  | 10000.00 |
+----+----------+-----+---------+----------+
```

5 rows in set (0.00 sec)

Example 5:

```
SELECT * FROM CUSTOMERS WHERE SALARY >= 6500;
```

Output:

```
+----+----------+-----+---------+----------+
| ID | NAME     | AGE | ADDRESS |   SALARY |
+----+----------+-----+---------+----------+
| 4  | Chaitali | 25  | Mumbai  |  6500.00 |
| 5  | Hardik   | 27  | Bhopal  |  8500.00 |
| 7  | Muffy    | 24  | Indore  | 10000.00 |
+----+----------+-----+---------+----------+
```

3 rows in set (0.00 sec)

3. SQL Logical Operators

Here is a list of all the logical operators available in SQL.

Operator Description

ALL The ALL operator is used to compare a value to all values in another value set.

AND The AND operator allows the existence of multiple conditions in an SQL statement's WHERE clause.

ANY The ANY operator is used to compare a value to any applicable value in the list as per the condition.

BETWEEN The BETWEEN operator is used to search for values that are within a set of values, given the minimum value and the maximum value.

EXISTS The EXISTS operator is used to search for the presence of a row in a specified table that meets a certain criterion.

IN The IN operator is used to compare a value to a list of literal values that have been specified.

LIKE The LIKE operator is used to compare a value to similar values using wildcard operators.

NOT 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.**

OR The OR operator is used to combine multiple conditions in an SQL statement's WHERE clause.

IS NULL The NULL operator is used to compare a value with a NULL value.

UNIQUE The UNIQUE operator searches every row of a specified table for uniqueness (no duplicates).

Logical Operators – Examples

Consider the CUSTOMERS table having the following records:

SELECT * FROM CUSTOMERS;

```
+----+----------+-----+-----------+----------+
| ID | NAME     | AGE | ADDRESS   | SALARY   |
+----+----------+-----+-----------+----------+
| 1  | Ramesh   | 32  | Ahmedabad | 2000.00  |
| 2  | Khilan   | 25  | Delhi     | 1500.00  |
| 3  | Kaushik  | 23  | Kota      | 2000.00  |
| 4  | Chaitali | 25  | Mumbai    | 6500.00  |
| 5  | Hardik   | 27  | Bhopal    | 8500.00  |
| 6  | Komal    | 22  | MP        | 4500.00  |
| 7  | Muffy    | 24  | Indore    | 10000.00 |
+----+----------+-----+-----------+----------+
```

7 rows in set (0.00 sec)

Here are some simple examples showing usage of SQL Comparison Operators:

Example 1:

SELECT * FROM CUSTOMERS WHERE AGE >= 25 AND SALARY >= 6500;

Output:

```
+----+----------+-----+---------+---------+
| ID | NAME     | AGE | ADDRESS | SALARY  |
+----+----------+-----+---------+---------+
| 4  | Chaitali | 25  | Mumbai  | 6500.00 |
| 5  | Hardik   | 27  | Bhopal  | 8500.00 |
+----+----------+-----+---------+---------+
```

2 rows in set (0.00 sec)

Example 2:

SELECT * FROM CUSTOMERS WHERE AGE >= 25 OR SALARY >= 6500;

Output:

```
+----+----------+-----+-----------+----------+
| ID | NAME     | AGE | ADDRESS   | SALARY   |
+----+----------+-----+-----------+----------+
| 1  | Ramesh   | 32  | Ahmedabad | 2000.00  |
| 2  | Khilan   | 25  | Delhi     | 1500.00  |
| 4  | Chaitali | 25  | Mumbai    | 6500.00  |
| 5  | Hardik   | 27  | Bhopal    | 8500.00  |
| 7  | Muffy    | 24  | Indore    | 10000.00 |
+----+----------+-----+-----------+----------+
```

5 rows in set (0.00 sec)

Example 3:

SELECT * FROM CUSTOMERS WHERE AGE IS NOT NULL;

Output:

```
+----+----------+-----+-----------+----------+
| ID | NAME     | AGE | ADDRESS   | SALARY   |
+----+----------+-----+-----------+----------+
| 1  | Ramesh   | 32  | Ahmedabad | 2000.00  |
| 2  | Khilan   | 25  | Delhi     | 1500.00  |
| 3  | kaushik  | 23  | Kota      | 2000.00  |
| 4  | Chaitali | 25  | Mumbai    | 6500.00  |
| 5  | Hardik   | 27  | Bhopal    | 8500.00  |
| 6  | Komal    | 22  | MP        | 4500.00  |
| 7  | Muffy    | 24  | Indore    | 10000.00 |
+----+----------+-----+-----------+----------+
7 rows in set (0.00 sec)
```

Example 4:

```
SELECT * FROM CUSTOMERS WHERE NAME LIKE 'Ko%';
```

Output:

```
+----+-------+-----+---------+---------+
| ID | NAME  | AGE | ADDRESS | SALARY  |
+----+-------+-----+---------+---------+
| 6  | Komal | 22  | MP      | 4500.00 |
+----+-------+-----+---------+---------+
```

1 row in set (0.00 sec)

Example 5:

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

Output:

```
+----+----------+-----+---------+---------+
| ID | NAME     | AGE | ADDRESS | SALARY  |
+----+----------+-----+---------+---------+
| 2  | Khilan   | 25  | Delhi   | 1500.00 |
| 4  | Chaitali | 25  | Mumbai  | 6500.00 |
| 5  | Hardik   | 27  | Bhopal  | 8500.00 |
+----+----------+-----+---------+---------+
```

3 rows in set (0.00 sec)

Example 6:

```
SELECT * FROM CUSTOMERS WHERE AGE BETWEEN 25 AND 27;
```

Output:

```
+----+----------+-----+---------+---------+
| ID | NAME     | AGE | ADDRESS | SALARY  |
+----+----------+-----+---------+---------+
| 2  | Khilan   | 25  | Delhi   | 1500.00 |
| 4  | Chaitali | 25  | Mumbai  | 6500.00 |
| 5  | Hardik   | 27  | Bhopal  | 8500.00 |
+----+----------+-----+---------+---------+
```

3 rows in set (0.00 sec)

Example 7:

```
SQL> SELECT AGE FROM CUSTOMERS
WHERE EXISTS (SELECT AGE FROM CUSTOMERS WHERE SALARY > 6500);
```

Output:

```
+-----+
| AGE |
+-----+
| 32  |
| 25  |
| 23  |
| 25  |
| 27  |
| 22  |
| 24  |
+-----+
7 rows in set (0.02 sec)
```

Example 8:

```
SELECT * FROM CUSTOMERS
WHERE AGE > ALL (SELECT AGE FROM CUSTOMERS WHERE SALARY > 6500);
```

Output:

```
+----+--------+-----+-----------+---------+
| ID | NAME   | AGE | ADDRESS   |  SALARY |
+----+--------+-----+-----------+---------+
| 1  | Ramesh | 32  | Ahmedabad | 2000.00 |
+----+--------+-----+-----------+---------+
1 row in set (0.02 sec)
```

Example 9:

```
SELECT * FROM CUSTOMERS
WHERE AGE > ANY (SELECT AGE FROM CUSTOMERS WHERE SALARY > 6500);
```

Output:

```
+----+----------+-----+-----------+---------+
| ID | NAME     | AGE | ADDRESS   | SALARY  |
+----+----------+-----+-----------+---------+
| 1  | Ramesh   | 32  | Ahmedabad | 2000.00 |
| 2  | Khilan   | 25  | Delhi     | 1500.00 |
| 4  | Chaitali | 25  | Mumbai    | 6500.00 |
| 5  | Hardik   | 27  | Bhopal    | 8500.00 |
+----+----------+-----+-----------+---------+
4 rows in set (0.00 sec)
```

B. SQL – Expressions

An expression is a combination of one or more values, operators and SQL functions that evaluate to a value. These SQL EXPRESSIONs are like formulae and they are written in query language. You can also use them to query the database for a specific set of data.

Syntax

Consider the basic syntax of the SELECT statement as follows:

```
SELECT column1, column2, columnN
FROM table_name
WHERE [CONDITION|EXPRESSION];
```

There are different types of SQL expressions, which are mentioned below:

- **Boolean**
- **Numeric**
- **Date**

1. Boolean Expressions

SQL Boolean Expressions fetch the data based on matching a single value. Following is the

syntax:

```
SELECT column1, column2, columnN
FROM table_name
WHERE SINGLE VALUE MATCHING EXPRESSION;
```

Consider the CUSTOMERS table having the following records:

SELECT * FROM CUSTOMERS;

```
+----+----------+-----+-----------+----------+
| ID | NAME     | AGE | ADDRESS   | SALARY   |
+----+----------+-----+-----------+----------+
| 1  | Ramesh   | 32  | Ahmedabad | 2000.00  |
| 2  | Khilan   | 25  | Delhi     | 1500.00  |
| 3  | kaushik  | 23  | Kota      | 2000.00  |
| 4  | Chaitali | 25  | Mumbai    | 6500.00  |
| 5  | Hardik   | 27  | Bhopal    | 8500.00  |
| 6  | Komal    | 22  | MP        | 4500.00  |
| 7  | Muffy    | 24  | Indore    | 10000.00 |
+----+----------+-----+-----------+----------+
```

7 rows in set (0.00 sec)

The following table is a simple example showing the usage of various SQL Boolean Expressions:

SELECT * FROM CUSTOMERS WHERE SALARY = 10000;

```
+----+-------+-----+---------+----------+
| ID | NAME  | AGE | ADDRESS | SALARY   |
+----+-------+-----+---------+----------+
| 7  | Muffy | 24  | Indore  | 10000.00 |
+----+-------+-----+---------+----------+
```

1 row in set (0.00 sec)

2. Numeric Expressions

These expressions are used to perform any mathematical operation in any query. Following is the **syntax**:

SELECT numerical_expression as OPERATION_NAME
[FROM table_name WHERE CONDITION] ;

Here, the numerical_expression is used for a mathematical expression or any formula.

Following is a simple example showing the usage of SQL Numeric Expressions:

SELECT (15 + 6) AS ADDITION

```
+----------+
| ADDITION |
+----------+
| 21       |
+----------+
```

1 row in set (0.00 sec)

There are several built-in functions like avg (), sum (), count (), etc., to perform what is known as the aggregate data calculations against a table or a specific table column.

SELECT COUNT(*) AS "RECORDS" FROM CUSTOMERS;

```
+---------+
| RECORDS |
+---------+
| 7       |
+---------+
```

1 row in set (0.00 sec)

3. Date Expressions

Date Expressions return the current system date and time values:

SELECT CURRENT_TIMESTAMP;

```
+---------------------+
| Current_Timestamp   |
+---------------------+
| 2022-04-12 11:11:22 |
+---------------------+
```

1 row in set (0.00 sec)

Another date expression is as shown below:

SELECT GETDATE();

```
+-------------------------+
| GETDATE                 |
+-------------------------+
| 2022-04-12 11:11:22.140 |
+-------------------------+
```

1 row in set (0.00 sec)

C. IN operator

The IN operator checks a value within a set of values separated by commas and retrieve the rows from the table which are matching. The IN returns 1 when the search value present within the range otherwise returns 0.

Syntax:

SELECT [column_name... | expression]

FROM [table_name]

{WHERE | {AND | OR}} value [NOT] IN ({comp_value1, comp_value2[, ...] | sub

Parameters:

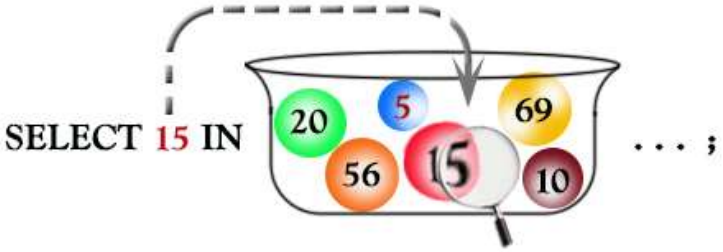
Name	Description
column_name	Name of the column of the table.
expression	Expression made up of a single constant, variable, scalar function, or column name and can also be the pieces of a SQL query that compare values against other values or perform arithmetic calculations.
table_name	Name of the table
{WHERE | HAVING | {AND | OR}} value	IN works with either the WHERE or the HAVING clause. You can also use AND or OR clause for multi-condition WHERE or the HAVING clause.
NOT	Used to exclude the defined multiple values in a WHERE clause condition.
comp_value1, comp_value2...| subquery	List of comparative values within the parentheses or a subquery that returns one or more values of a compatible datatype of the main query.

Example: SQL IN Operator

To know whether the search value 15 is present within the specified range from the can table, the following SQL statement can be used :

SQL Code:

```
SELECT 15 IN (5,10,15,20,56,69)
FROM can;
```



SQL IN operator with text value

The checking value of IN operator can be a string or word or sentence. These values can also be checked within a set of values separated by commas and retrieve the rows containing these values.

Example:

Sample table: agents

AGENT_CODE	AGENT_NAME	WORKING_AREA	COMMISSION	PHONE_NO
A007	Ramasundar	Bangalore	0.15	077-25814763
A003	Alex	London	0.13	075-12458969
A008	Alford	New York	0.12	044-25874365
A011	Ravi Kumar	Bangalore	0.15	077-45625874
A010	Santakumar	Chennai	0.14	007-22388644
A012	Lucida	San Jose	0.12	044-52981425
A005	Anderson	Brisban	0.13	045-21447739
A001	Subbarao	Bangalore	0.14	077-12346674
A002	Mukesh	Mumbai	0.11	029-12358964
A006	McDen	London	0.15	078-22255588
A004	Ivan	Torento	0.15	008-22544166
A009	Benjamin	Hampshair	0.11	008-22536178

Here we look for all agents in the agents table of inventory database who have a working area of the state of 'London', 'Mumbai' or 'Chennai'.

Here is the SQL statement:

SQL Code:

```
SELECT *
FROM agents
WHERE working_area  IN('London','Mumbai','Chennai');
```

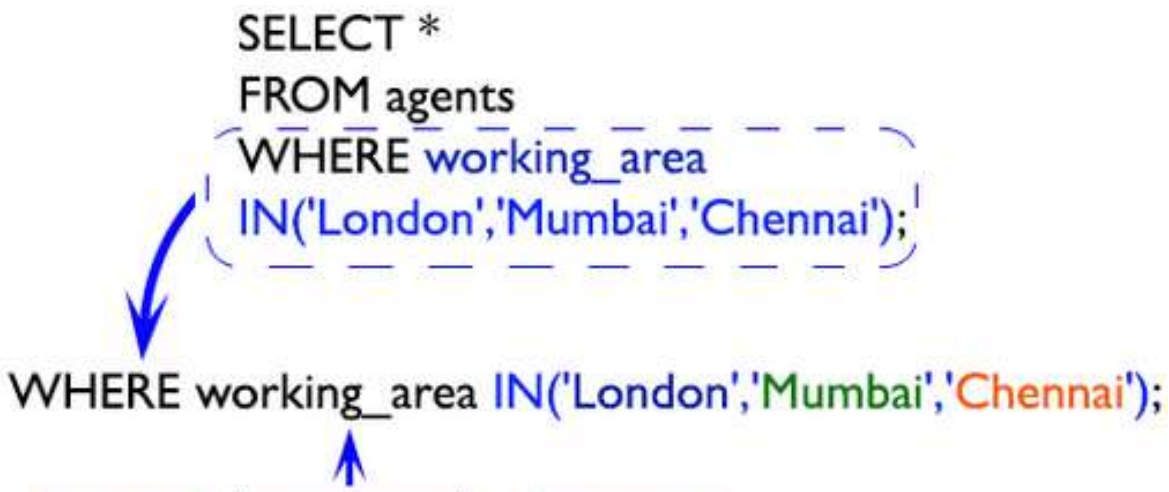
This statement can also be used like bellow:

SQL Code:

```
SELECT *
FROM agents
WHERE working_area='London'  OR working_area='Mumbai'  OR
working_area='Chennai';
```

Output:

```
AGENT_CODE AGENT_NAME     WORKING_AREA    COMMISSION PHONE_NO
---------- ------------------------------ ---------- --------
A003       Alex           London           .13 075-12458969
A010       Santakumar     Chennai          .14 007-22388644
A002       Mukesh         Mumbai           .11 029-12358964
A006       McDen          London           .15 078-22255588
```



agent_code	working_area	commission
A007	Bangalore	0.15
A005	Brisban	0.14
A001	Bangalore	0.14
A003	London	0.12
A008	New York	0.12
A002	Mumbai	0.11
A006	London	0.15
A004	Torento	0.15
A011	Bangalore	0.15
A010	Chennai	0.14
A009	Hampshair	0.11
A012	San Jose	0.12

SQL IN operator with boolean NOT

In the following example, we have discussed the usage of IN operator with the boolean operator NOT in a select statement.

Example:
Sample table: agents

AGENT_CODE	AGENT_NAME	WORKING_AREA	COMMISSION	PHONE_NO
A007	Ramasundar	Bangalore	0.15	077-25814763
A003	Alex	London	0.13	075-12458969
A008	Alford	New York	0.12	044-25874365
A011	Ravi Kumar	Bangalore	0.15	077-45625874
A010	Santakumar	Chennai	0.14	007-22388644
A012	Lucida	San Jose	0.12	044-52981425
A005	Anderson	Brisban	0.13	045-21447739
A001	Subbarao	Bangalore	0.14	077-12346674
A002	Mukesh	Mumbai	0.11	029-12358964
A006	McDen	London	0.15	078-22255588
A004	Ivan	Torento	0.15	008-22544166
A009	Benjamin	Hampshair	0.11	008-22536178

To get data of all columns from the 'agents' table with the following condition :

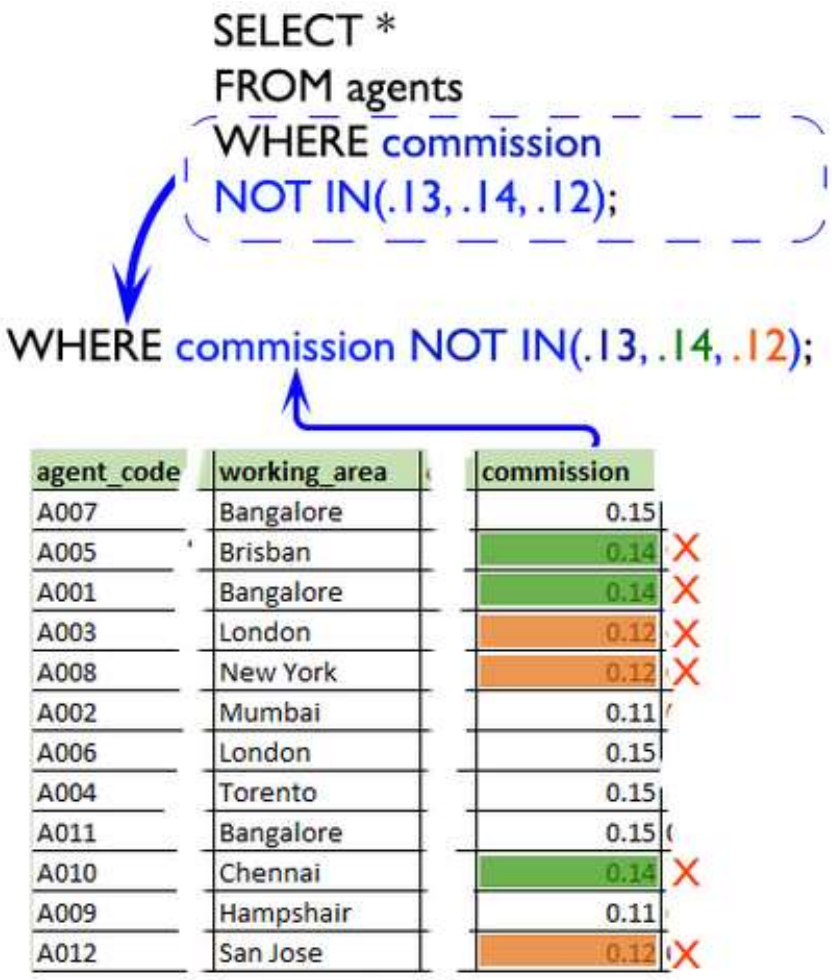
1. 'commission' for the agents will be none of .13, .14, .12,

the following SQL statement can be used:

SQL Code:

```
SELECT *
FROM agents
WHERE commission NOT IN (.13,.14,.12);
```

AGENT_CODE	AGENT_NAME	WORKING_AREA	COMMISSION	PHONE_NO
A009	Benjamin	Hampshair	.11	008-22536178
A007	Ramasundar	Bangalore	.15	077-25814763
A011	Ravi Kumar	Bangalore	.15	077-45625874
A002	Mukesh	Mumbai	.11	029-12358964
A006	McDen	London	.15	078-22255588
A004	Ivan	Torento	.15	008-22544166



D. BETWEEN Operator

The SQL BETWEEN operator tests an expression against a range. The range consists of a beginning, followed by an AND keyword and an end expression. The operator returns TRUE when the search value present within the range otherwise returns FALSE. The results are NULL if any of the range values are NULL.

Syntax:

SELECT [column_name... | expression1]
FROM [table_name]
WHERE expression2 [NOT] BETWEEN value_from AND value_to;

Parameters:

Name	Description
column_name	Name of the column of the table.
expression1	Expression made up of a single constant, variable, scalar function, or column name and can also be the pieces of a SQL query that compare values against other values or perform arithmetic calculations.
table_name	Name of the table.
WHERE expression2	Compares a scalar expression, such as a column, to the range of values bounded by value_from and value_to
value_from, value_to	Starting value and ending value.

Example: SQL BETWEEN Operator

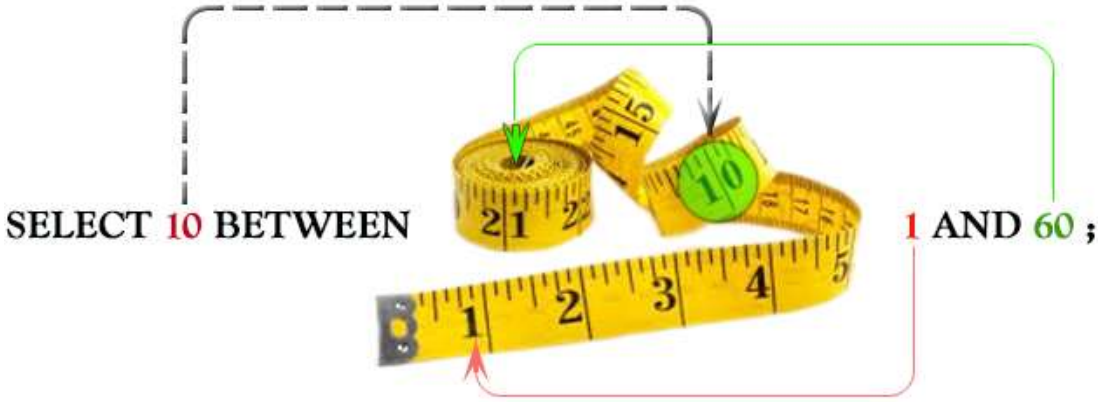
To know whether the search value 15 is present within the specified range from the can table, the following SQL statement can be used:

SELECT 'found'
FROM can
WHERE 10 BETWEEN 5 AND 20;

Output:

```
'FOUND'
--------
found
```

Pictorial Presentation: SQL BETWEEN operator



Here we look for all agents in the agents table of inventory database whose commission should be within .12 to .14.

Sample table: agents

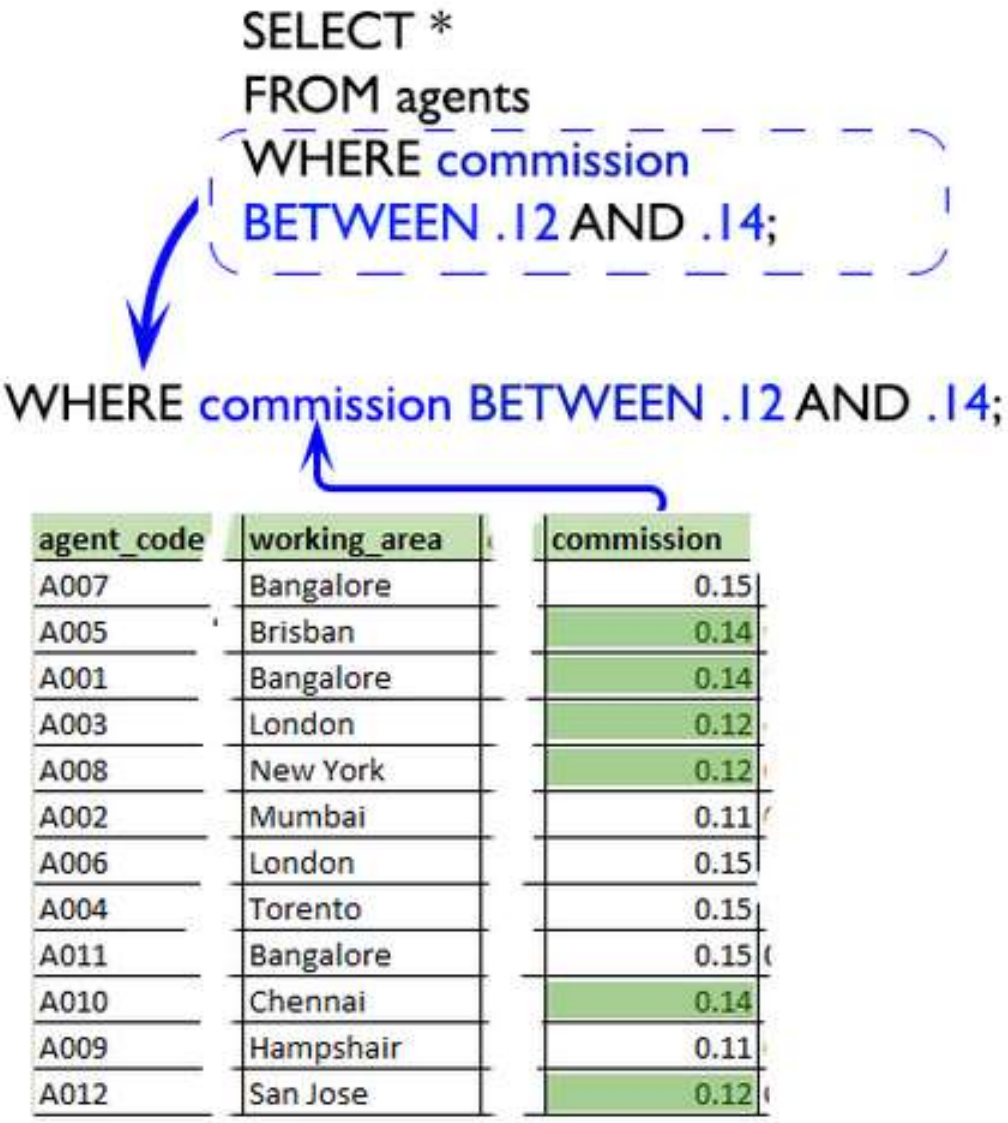
AGENT_CODE	AGENT_NAME	WORKING_AREA	COMMISSION	PHONE_NO
A007	Ramasundar	Bangalore	0.15	077-25814763
A003	Alex	London	0.13	075-12458969
A008	Alford	New York	0.12	044-25874365
A011	Ravi Kumar	Bangalore	0.15	077-45625874
A010	Santakumar	Chennai	0.14	007-22388644
A012	Lucida	San Jose	0.12	044-52981425
A005	Anderson	Brisban	0.13	045-21447739
A001	Subbarao	Bangalore	0.14	077-12346674
A002	Mukesh	Mumbai	0.11	029-12358964
A006	McDen	London	0.15	078-22255588
A004	Ivan	Torento	0.15	008-22544166
A009	Benjamin	Hampshair	0.11	008-22536178

SQL Code:

```
SELECT * FROM agents
WHERE commission BETWEEN .12 AND .14;
```

AGENT_CODE	AGENT_NAME	WORKING_AREA	COMMISSION	PHONE_NO
A003	Alex	London	.13	075-12458969
A001	Subbarao	Bangalore	.14	077-12346674
A008	Alford	New York	.12	044-25874365
A010	Santakumar	Chennai	.14	007-22388644
A012	Lucida	San Jose	.12	044-52981425
A005	Anderson	Brisban	.13	045-21447739

Pictorial Presentation: SQL BETWEEN operator



agent_code	working_area	commission
A007	Bangalore	0.15
A005	Brisban	0.14
A001	Bangalore	0.14
A003	London	0.12
A008	New York	0.12
A002	Mumbai	0.11
A006	London	0.15
A004	Torento	0.15
A011	Bangalore	0.15
A010	Chennai	0.14
A009	Hampshair	0.11
A012	San Jose	0.12

Example: SQL Between operator with IN

Sample table: customer

CUST_CODE	CUST_NAME	CUST_CITY	WORKING_AREA	CUST_COUNTRY	GRADE	OPENING_AMT	RECEIVE_AMT	PAYMENT_AMT	OUTSTANDING_AMT	PHONE_NO	AGENT_CODE
C00013	Holmes	London	London	UK	2	6000.00	5000.00	7000.00	4000.00	BBBBBBB	A003
C00001	Micheal	New York	New York	USA	2	3000.00	5000.00	2000.00	6000.00	CCCCCCC	A008
C00020	Albert	New York	New York	USA	3	5000.00	7000.00	6000.00	6000.00	BBBBSBB	A008
C00025	Ravindran	Bangalore	Bangalore	India	2	5000.00	7000.00	4000.00	8000.00	AVAVAVA	A011
C00024	Cook	London	London	UK	2	4000.00	9000.00	7000.00	6000.00	FSDDSDF	A006
C00015	Stuart	London	London	UK	1	6000.00	8000.00	3000.00	11000.00	GFSGERS	A003
C00002	Bolt	New York	New York	USA	3	5000.00	7000.00	9000.00	3000.00	DDNRDRH	A008
C00018	Fleming	Brisban	Brisban	Australia	2	7000.00	7000.00	9000.00	5000.00	NHBGVFC	A005
C00021	Jacks	Brisban	Brisban	Australia	1	7000.00	7000.00	7000.00	7000.00	WERTGDF	A005
C00019	Yearannaidu	Chennai	Chennai	India	1	8000.00	7000.00	7000.00	8000.00	ZZZZBFV	A010
C00005	Sasikant	Mumbai	Mumbai	India	1	7000.00	11000.00	7000.00	11000.00	147-25896312	A002
C00007	Ramanathan	Chennai	Chennai	India	1	7000.00	11000.00	9000.00	9000.00	GHRDWSD	A010
C00022	Avinash	Mumbai	Mumbai	India	2	7000.00	11000.00	9000.00	9000.00	113-12345678	A002
C00004	Winston	Brisban	Brisban	Australia	1	5000.00	8000.00	7000.00	6000.00	AAAAAAA	A005
C00023	Karl	London	London	UK	0	4000.00	6000.00	7000.00	3000.00	AAAABAA	A006
C00006	Shilton	Torento	Torento	Canada	1	10000.00	7000.00	6000.00	11000.00	DDDDDDD	A004
C00010	Charles	Hampshair	Hampshair	UK	3	6000.00	4000.00	5000.00	5000.00	MMMMMM	A009
C00017	Srinivas	Bangalore	Bangalore	India	2	8000.00	4000.00	3000.00	9000.00	AAAAAAB	A007
C00012	Steven	San Jose	San Jose	USA	1	5000.00	7000.00	9000.00	3000.00	KRFYGJK	A012
C00008	Karolina	Torento	Torento	Canada	1	7000.00	7000.00	9000.00	5000.00	HJKORED	A004
C00003	Martin	Torento	Torento	Canada	2	8000.00	7000.00	7000.00	8000.00	MJYURFD	A004
C00009	Ramesh	Mumbai	Mumbai	India	3	8000.00	7000.00	3000.00	12000.00	Phone No	A002
C00014	Rangarappa	Bangalore	Bangalore	India	2	8000.00	11000.00	7000.00	12000.00	AAAATGF	A001
C00016	Venkatpati	Bangalore	Bangalore	India	2	8000.00	11000.00	7000.00	12000.00	JRTVFDD	A007
C00011	Sundariya	Chennai	Chennai	India	3	7000.00	11000.00	7000.00	11000.00	PPHGRTS	A010

To get data of all columns from the 'customer' table with following conditions :

1. 'agent_code' must be within 'A003' and 'A008',

2. but 'agent_code' 'A004', 'A007' and 'A005' should not appear,

The following SQL statement can be used :

```
SELECT agent_code, cust_code, cust_name, cust_city ,
FROM customer,
WHERE (agent_code BETWEEN 'A003' AND 'A008'),
AND NOT agent_code IN('A004','A007','A005');
```

Output:

```
AGENT_CODE CUST_CODE  CUST_NAME                                CUST_CITY
---------- ---------- ---------------------------------------- ------------
A003       C00013     Holmes                                   London
A008       C00001     Micheal                                  New York
A008       C00020     Albert                                   New York
A006       C00024     Cook                                     London
A003       C00015     Stuart                                   London
A008       C00002     Bolt                                     New York
A006       C00023     Karl                                     London
```

THE FUNCTION OPERATORS

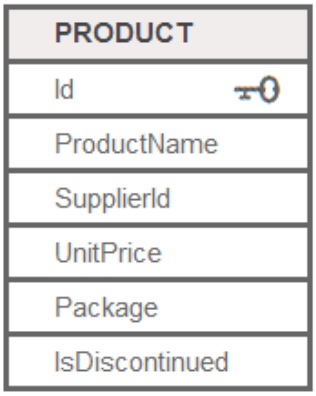
A. SQL MIN() Function

The MIN() function returns the smallest value of the selected column.

SQL MIN() Syntax

```
SELECT MIN(column_name) FROM table_name;
```

Example: Find the cheapest product



SELECT MIN (UnitPrice),

FROM Product;

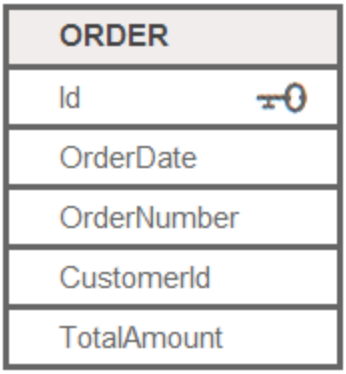
B. The MAX() Function

The MAX() function returns the largest value of the selected column.

SQL MAX() Syntax

SELECT MAX(column_name),

FROM table_name;



Example: Find the largest order placed in 2014

```
SELECT MAX(TotalAmount)

FROM Order

WHERE OrderDate = 2014
```

SQL SELECT COUNT, SUM, AVG

- SELECT COUNT returns a count of the number of data values.
- SELECT SUM returns the sum of the data values.
- SELECT AVG returns the average of the data values.

C. The COUNT () Function

The general COUNT syntax is:

SELECT COUNT(column-name),
FROM table-name;

D. The SUM () Function

The general SUM syntax is:

SELECT SUM(column-name)
FROM table-name

E. The AVG () Function

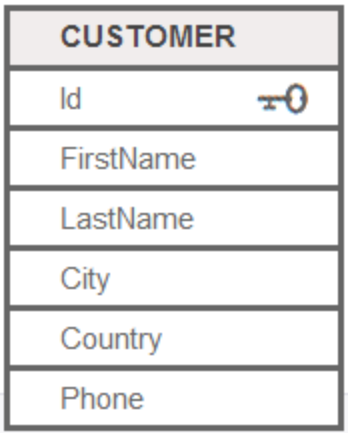
The general AVG syntax is:

SELECT AVG(column-name),
FROM table-name;

Example: Find the number of customers

SELECT COUNT(Id),

FROM Customer;

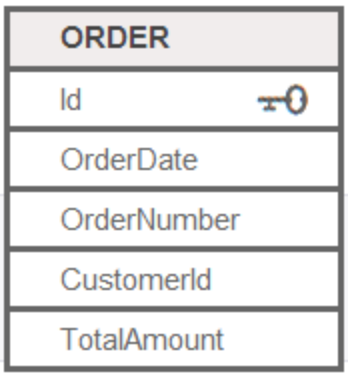


Example: Compute the total amount sold in 2013

SELECT SUM(TotalAmount),

FROM Order,

WHERE OrderDate = 2013;



Example: Compute the average size of all orders

SELECT AVG(TotalAmount),

FROM Order;

Summary

In this lecture,

After the students learned how to deal with the data manipulation language (DML) & data query language (DQL) operators in structured query language (SQL) with entire examples that illustrate the work of its functions. Now, they will learn the most important functions operators in SQLwhich are (**Conidition operators** and **Function operators**).

In **Conidition operators** they will be able to learn how to deal with:

- Sql – Operators (Arithmetic Operators, Comparison Operators and The Logical Operators)
- Sql – Expressions (Boolean Expressions, Numeric Expressions and The Date Expressions)
- In – operators
- BETWEEN – operators

While in the **Function operators** they will be able to learn how to deal with:

- Min () function
- Max () function
- Count () function
- Sum () function
- Avg () function

Finally, from this lecture the students will be able to create a complete database that containing all the important commands for the Conidition operators and Function operators in structured query language (SQL).