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

SQL Arithmetic Operators

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

Show Examples

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

SQL Comparison Operators

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

Show Examples

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	$a! = b$ is true.

	are not equal then condition becomes 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.

SQL Logical Operators

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

Show Examples

Sr.No.	Operator & Description
1	**ALL** The ALL operator is used to compare a value to all values in another value set.
2	**AND** The AND operator allows the existence of multiple conditions in an SQL statement's WHERE clause.
3	**ANY** The ANY operator is used to compare a value to any applicable value in the list as per the condition.

4	**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.
5	**EXISTS** The EXISTS operator is used to search for the presence of a row in a specified table that meets a certain criterion.
6	**IN** The IN operator is used to compare a value to a list of literal values that have been specified.
7	**LIKE** The LIKE operator is used to compare a value to similar values using wildcard operators.
8	**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.**
9	**OR** The OR operator is used to combine multiple conditions in an SQL statement's WHERE clause.
10	**IS NULL** The NULL operator is used to compare a value with a NULL value.
11	**UNIQUE** The UNIQUE operator searches every row of a specified table for uniqueness $noduplicates$.

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

Let us now discuss each of these in detail.

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 –

```
SQL> 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 –

```
SQL> SELECT * FROM CUSTOMERS WHERE SALARY = 10000;
+----+-------+-----+---------+----------+
| ID | NAME  | AGE | ADDRESS | SALARY   |
+----+-------+-----+---------+----------+
|  7 | Muffy |  24 | Indore  | 10000.00 |
+----+-------+-----+---------+----------+
1 row in set (0.00 sec)
```

Numeric Expression

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 –

```
SQL> 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.

```
SQL> SELECT COUNT(*) AS "RECORDS" FROM CUSTOMERS;
+---------+
| RECORDS |
+---------+
|       7 |
+---------+
1 row in set (0.00 sec)
```

Date Expressions

Date Expressions return current system date and time values –

```
SQL>  SELECT CURRENT_TIMESTAMP;
+---------------------+
| Current_Timestamp   |
+---------------------+
| 2009-11-12 06:40:23 |
+---------------------+
1 row in set (0.00 sec)
```

Another date expression is as shown below –

```
SQL>  SELECT  GETDATE();;
+-------------------------+
| GETDATE                 |
+-------------------------+
| 2009-10-22 12:07:18.140 |
+-------------------------+
1 row in set (0.00 sec)
```

SQL - WHERE CLAUSE

The SQL **WHERE** clause is used to specify a condition while fetching the data from a single table or by joining with multiple tables. If the given condition is satisfied, then only it returns a specific value from the table. You should use the WHERE clause to filter the records and fetching only the necessary records.

The WHERE clause is not only used in the SELECT statement, but it is also used in the UPDATE, DELETE statement, etc., which we would examine in the subsequent chapters.

Syntax

The basic syntax of the SELECT statement with the WHERE clause is as shown below.

```
SELECT column1, column2, columnN
FROM table_name
WHERE [condition]
```

You can specify a condition using the comparison or logical operators like >, <, =, **LIKE, NOT**, etc. The following examples would make this concept clear.

Example

Consider the CUSTOMERS table having the following records –

```
+----+----------+-----+-----------+----------+
| 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 |
+----+----------+-----+-----------+----------+
```

The following code is an example which would fetch the ID, Name and Salary fields from the CUSTOMERS table, where the salary is greater than 2000 –

```
SQL> SELECT ID, NAME, SALARY
FROM CUSTOMERS
WHERE SALARY > 2000;
```

This would produce the following result –

```
+----+----------+----------+
| ID | NAME     | SALARY   |
+----+----------+----------+
|  4 | Chaitali |  6500.00 |
|  5 | Hardik   |  8500.00 |
|  6 | Komal    |  4500.00 |
|  7 | Muffy    | 10000.00 |
+----+----------+----------+
```

The following query is an example, which would fetch the ID, Name and Salary fields from the CUSTOMERS table for a customer with the name **Hardik**.

Here, it is important to note that all the strings should be given inside single quotes ''. Whereas, numeric values should be given without any quote as in the above example.

```
SQL> SELECT ID, NAME, SALARY
FROM CUSTOMERS
WHERE NAME = 'Hardik';
```

This would produce the following result –

```
+----+----------+----------+
| ID | NAME     | SALARY   |
+----+----------+----------+
|  5 | Hardik   |  8500.00 |
+----+----------+----------+
```

SQL - AND AND OR CONJUNCTIVE OPERATORS

The SQL **AND** & **OR** operators are used to combine multiple conditions to narrow data in an SQL statement. These two operators are called as the conjunctive operators.

These operators provide a means to make multiple comparisons with different operators in the same SQL statement.

The AND Operator

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

Syntax

The basic syntax of the AND operator with a WHERE clause is as follows –

```
SELECT column1, column2, columnN
FROM table_name
WHERE [condition1] AND [condition2]...AND [conditionN];
```

You can combine N number of conditions using the AND operator. For an action to be taken by the SQL statement, whether it be a transaction or a query, all conditions separated by the AND must be TRUE.

Example

Consider the CUSTOMERS table having the following records –

```
+----+----------+-----+-----------+----------+
| 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 |
+----+----------+-----+-----------+----------+
```

Following is an example, which would fetch the ID, Name and Salary fields from the CUSTOMERS table, where the salary is greater than 2000 and the age is less than 25 years –

```
SQL> SELECT ID, NAME, SALARY
FROM CUSTOMERS
WHERE SALARY > 2000 AND age < 25;
```

This would produce the following result –

```
+----+-------+----------+
| ID | NAME  | SALARY   |
+----+-------+----------+
|  6 | Komal |  4500.00 |
|  7 | Muffy | 10000.00 |
+----+-------+----------+
```

The OR Operator

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

Syntax

The basic syntax of the OR operator with a WHERE clause is as follows –

```
SELECT column1, column2, columnN
FROM table_name
WHERE [condition1] OR [condition2]...OR [conditionN]
```

You can combine N number of conditions using the OR operator. For an action to be taken by the SQL statement, whether it be a transaction or query, the only any ONE of the conditions separated by the OR must be TRUE.

Example

Consider the CUSTOMERS table having the following records –

```
+----+----------+-----+-----------+----------+
| 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 |
+----+----------+-----+-----------+----------+
```

The following code block hasa query, which would fetch the ID, Name and Salary fields from the CUSTOMERS table, where the salary is greater than 2000 OR the age is less than 25 years.

```
SQL> SELECT ID, NAME, SALARY
FROM CUSTOMERS
WHERE SALARY > 2000 OR age < 25;
```

This would produce the following result –

```
+----+----------+----------+
| ID | NAME     | SALARY   |
+----+----------+----------+
|  3 | kaushik  |  2000.00 |
|  4 | Chaitali |  6500.00 |
|  5 | Hardik   |  8500.00 |
|  6 | Komal    |  4500.00 |
|  7 | Muffy    | 10000.00 |
+----+----------+----------+
```

SQL - UPDATE QUERY

The SQL **UPDATE** Query is used to modify the existing records in a table. You can use the WHERE clause with the UPDATE query to update the selected rows, otherwise all the rows would be affected.

Syntax

The basic syntax of the UPDATE query with a WHERE clause is as follows –

```
UPDATE table_name
SET column1 = value1, column2 = value2...., columnN = valueN
WHERE [condition];
```

You can combine N number of conditions using the AND or the OR operators.

Example

Consider the CUSTOMERS table having the following records –

```
+----+----------+-----+-----------+----------+
| 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 |
+----+----------+-----+-----------+----------+
```

The following query will update the ADDRESS for a customer whose ID number is 6 in the table.

```
SQL> UPDATE CUSTOMERS
SET ADDRESS = 'Pune'
WHERE ID = 6;
```

Now, the CUSTOMERS table would have the following records –

```
+----+----------+-----+-----------+----------+
| 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 | Pune      |  4500.00 |
|  7 | Muffy    |  24 | Indore    | 10000.00 |
+----+----------+-----+-----------+----------+
```

If you want to modify all the ADDRESS and the SALARY column values in the CUSTOMERS table, you do not need to use the WHERE clause as the UPDATE query would be enough as shown in the following code block.

```
SQL> UPDATE CUSTOMERS
SET ADDRESS = 'Pune', SALARY = 1000.00;
```

Now, CUSTOMERS table would have the following records –

```
+----+----------+-----+---------+---------+
| ID | NAME     | AGE | ADDRESS | SALARY  |
+----+----------+-----+---------+---------+
|  1 | Ramesh   |  32 | Pune    | 1000.00 |
|  2 | Khilan   |  25 | Pune    | 1000.00 |
|  3 | kaushik  |  23 | Pune    | 1000.00 |
|  4 | Chaitali |  25 | Pune    | 1000.00 |
|  5 | Hardik   |  27 | Pune    | 1000.00 |
|  6 | Komal    |  22 | Pune    | 1000.00 |
|  7 | Muffy    |  24 | Pune    | 1000.00 |
+----+----------+-----+---------+---------+
```

SQL - ORDER BY CLAUSE

The SQL **ORDER BY** clause is used to sort the data in ascending or descending order, based on one or more columns. Some databases sort the query results in an ascending order by default.

Syntax

The basic syntax of the ORDER BY clause is as follows –

```
SELECT column-list
FROM table_name
[WHERE condition]
[ORDER BY column1, column2, .. columnN] [ASC | DESC];
```

You can use more than one column in the ORDER BY clause. Make sure whatever column you are using to sort that column should be in the column-list.

Example

Consider the CUSTOMERS table having the following records –

```
+----+----------+-----+-----------+----------+
| 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 |
+----+----------+-----+-----------+----------+
```

The following code block has an example, which would sort the result in an ascending order by the NAME and the SALARY –

```
SQL> SELECT * FROM CUSTOMERS
   ORDER BY NAME, SALARY;
```

This would produce the following result –

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

The following code block has an example, which would sort the result in the descending order by NAME.

```
SQL> SELECT * FROM CUSTOMERS
   ORDER BY NAME DESC;
```

This would produce the following result –

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

SQL - GROUP BY

The SQL **GROUP BY** clause is used in collaboration with the SELECT statement to arrange identical data into groups. This GROUP BY clause follows the WHERE clause in a SELECT statement and precedes the ORDER BY clause.

Syntax

The basic syntax of a GROUP BY clause is shown in the following code block. The GROUP BY clause must follow the conditions in the WHERE clause and must precede the ORDER BY clause if one is used.

```
SELECT column1, column2
FROM table_name
WHERE [ conditions ]
GROUP BY column1, column2
ORDER BY column1, column2
```

Example

Consider the CUSTOMERS table is having the following records –

```
+----+----------+-----+-----------+----------+
| 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 |
+----+----------+-----+-----------+----------+
```

If you want to know the total amount of the salary on each customer, then the GROUP BY query would be as follows.

```
SQL> SELECT NAME, SUM(SALARY) FROM CUSTOMERS
   GROUP BY NAME;
```

This would produce the following result –

```
+----------+-------------+
| NAME     | SUM(SALARY) |
+----------+-------------+
| Chaitali |     6500.00 |
| Hardik   |     8500.00 |
| kaushik  |     2000.00 |
| Khilan   |     1500.00 |
| Komal    |     4500.00 |
| Muffy    |    10000.00 |
| Ramesh   |     2000.00 |
+----------+-------------+
```

Now, let us look at a table where the CUSTOMERS table has the following records with duplicate names –

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

Now again, if you want to know the total amount of salary on each customer, then the GROUP BY query would be as follows –

```
SQL> SELECT NAME, SUM(SALARY) FROM CUSTOMERS
   GROUP BY NAME;
```

This would produce the following result –

```
+---------+-------------+
| NAME    | SUM(SALARY) |
+---------+-------------+
| Hardik  |     8500.00 |
| kaushik |     8500.00 |
| Komal   |     4500.00 |
| Muffy   |    10000.00 |
| Ramesh  |     3500.00 |
+---------+-------------+
```

SQL - DISTINCT KEYWORD

The SQL **DISTINCT** keyword is used in conjunction with the SELECT statement to eliminate all the duplicate records and fetching only unique records.

There may be a situation when you have multiple duplicate records in a table. While fetching such records, it makes more sense to fetch only those unique records instead of fetching duplicate records.

Syntax

The basic syntax of DISTINCT keyword to eliminate the duplicate records is as follows –

```
SELECT DISTINCT column1, column2,.....columnN
FROM table_name
WHERE [condition]
```

Example

Consider the CUSTOMERS table having the following records –

```
+----+----------+-----+-----------+----------+
| 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 |
+----+----------+-----+-----------+----------+
```

First, let us see how the following SELECT query returns the duplicate salary records.

```
SQL> SELECT SALARY FROM CUSTOMERS
   ORDER BY SALARY;
```

This would produce the following result, where the salary 2000 is coming twice which is a duplicate record from the original table.

```
+----------+
| SALARY   |
+----------+
|  1500.00 |
|  2000.00 |
|  2000.00 |
|  4500.00 |
|  6500.00 |
|  8500.00 |
| 10000.00 |
+----------+
```

Now, let us use the DISTINCT keyword with the above SELECT query and then see the result.

```
SQL> SELECT DISTINCT SALARY FROM CUSTOMERS
   ORDER BY SALARY;
```

This would produce the following result where we do not have any duplicate entry.

```
+----------+
| SALARY   |
+----------+
|  1500.00 |
|  2000.00 |
|  4500.00 |
|  6500.00 |
|  8500.00 |
| 10000.00 |
+----------+
```

SQL - SORTING RESULTS

The SQL **ORDER BY** clause is used to sort the data in ascending or descending order, based on one or more columns. Some databases sort the query results in an ascending order by default.

Syntax

The basic syntax of the ORDER BY clause which would be used to sort the result in an ascending or descending order is as follows –

```
SELECT column-list
FROM table_name
[WHERE condition]
[ORDER BY column1, column2, .. columnN] [ASC | DESC];
```

You can use more than one column in the ORDER BY clause. Make sure that whatever column you are using to sort, that column should be in the column-list.

Example

Consider the CUSTOMERS table having the following records –

```
+----+----------+-----+-----------+----------+
| 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 |
+----+----------+-----+-----------+----------+
```

Following is an example, which would sort the result in an ascending order by NAME and SALARY.

```
SQL> SELECT * FROM CUSTOMERS
   ORDER BY NAME, SALARY;
```

This would produce the following result –

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

The following code block has an example, which would sort the result in a descending order by NAME.

```
SQL> SELECT * FROM CUSTOMERS
   ORDER BY NAME DESC;
```

This would produce the following result –

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

To fetch the rows with their own preferred order, the SELECT query used would be as follows –

```
SQL> SELECT * FROM CUSTOMERS
   ORDER BY (CASE ADDRESS
   WHEN 'DELHI'          THEN 1
   WHEN 'BHOPAL'         THEN 2
   WHEN 'KOTA'    THEN 3
   WHEN 'AHMADABAD' THEN 4
   WHEN 'MP'     THEN 5
   ELSE 100 END) ASC, ADDRESS DESC;
```

This would produce the following result –

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

This will sort the customers by ADDRESS in your **ownoOrder** of preference first and in a natural order for the remaining addresses. Also, the remaining Addresses will be sorted in the reverse alphabetical order.