**SQL – ORDER BY**

The ORDER BY statement in SQL is used to sort the fetched data in either ascending or descending according to one or more columns.

* By default ORDER BY sorts the data in ascending order.
* We can use the keyword DESC to sort the data in descending order and the keyword ASC to sort in ascending order.

## ntroduction to the SQL Server ORDER BY clause

When you use the [SELECT](https://www.sqlservertutorial.net/sql-server-basics/sql-server-select/) statement to query data from a table, the order of rows in the result set is not guaranteed. It means that SQL Server can return a result set with an unspecified order of rows.

The only way for you to guarantee that the rows in the result set are sorted is to use the ORDER BY clause. The following illustrates the ORDER BY clause syntax:

SELECT

select\_list

FROM

table\_name

ORDER BY

Code language: SQL (Structured Query Language) (sql)

[ASC | DESC ]

In this syntax:

### column\_name | expression

First, you specify a column name or an expression on which to sort the result set of the query. If you specify multiple columns, **the result set is sorted by the first column and** **then that sorted result set is sorted by the second column**, and so on.

The columns that appear in the ORDER BY clause must correspond to either column in the select list or to columns defined in the table specified in the FROM clause.

### ASC | DESC

Second, use ASC or DESC to specify whether the values in the specified column should be sorted in ascending or descending order.

The ASC sorts the result from the lowest value to the highest value while the DESC sorts the result set from the highest value to the lowest one.

If you don’t explicitly specify ASC or DESC, SQL Server uses ASC as the default sort order. Also, SQL Server treats [NULL](https://www.sqlservertutorial.net/sql-server-basics/sql-server-null/) as the lowest value.

When processing the SELECT statement that has an ORDER BY clause, the ORDER BY clause is the very last clause to be processed.

### What are DESC and ASC Keywords?

|  |  |
| --- | --- |
| **ORDER BY in MySQL: DESC & ASCASC is the short form for ascending** | **ORDER BY in MySQL: DESC & ASCSQL DESC is the short form for descending** |
| It is used to sort the query results in a top to bottom style. | It is used to sort the query results in a bottom to top style |
| When working on date data types, the **earliest date is** shown on top of the list. | . When working on date types, the latest date is shown on top of the list. |
| When working with numeric data types, the lowest values are shown on top of the list. | When working with numeric data types, the highest values are shown at top of the query result set. |
| When working with **string data types,** the query result set is sorted from those starting with the letter A going up to the letter Z. | When working with string data types, the query result set is sorted from those starting with the letter Z going down to the letter A. |

**Sort according to one column:**

 To sort in ascending or descending order we can use the keywords ASC or DESC respectively.

**Syntax:**

**SELECT \* FROM table\_name ORDER BY column\_name ASC|DESC**

**//Where**

**table\_name**: name of the table.

**column\_name**: name of the column according to which the data is needed to be arranged.

**ASC**: to sort the data in ascending order.

**DESC**: to sort the data in descending order.

**|** : use either ASC or DESC to sort in ascending or descending order//

**Sort according to multiple columns:**

To sort in ascending or descending order we can use the keywords ASC or DESC respectively. To sort according to multiple columns, separate the names of columns by the (,) operator.

**Syntax:**

**SELECT \* FROM table\_name ORDER BY column1 ASC|DESC , column2 ASC|DESC**



Now consider the above database table and find the results of different queries.

**Sort according to a single column:**

In this example, we will fetch all data from the table Student and sort the result in descending order according to the column ROLL\_NO.

**Query:**

SELECT \* FROM Student ORDER BY ROLL\_NO DESC;

**Output:**

| **ROLL\_NO** | **NAME** | **ADDRESS** | **PHONE** | **Age** |
| --- | --- | --- | --- | --- |
| 8 | NIRAJ | ALIPUR | XXXXXXXXXX | 19 |
| 7 | ROHIT | BALURGHAT | XXXXXXXXXX | 18 |
| 6 | DHANRAJ | BARABAJAR | XXXXXXXXXX | 20 |
| 5 | SAPTARHI | KOLKATA | XXXXXXXXXX | 19 |
| 4 | DEEP | RAMNAGAR | XXXXXXXXXX | 18 |
| 3 | RIYANKA | SILIGURI | XXXXXXXXXX | 20 |
| 2 | PRATIK | BIHAR | XXXXXXXXXX | 19 |
| 1 | HARSH | DELHI | XXXXXXXXXX | 18 |

In the above example, if we want to sort in ascending order we have to use ASC in place of DESC.

**Sort according to multiple columns:**

In this example we will fetch all data from the table Student and then sort the result in ascending order first according to the column Age. and then in descending order according to the column ROLL\_NO.   
**Query:**

SELECT \* FROM Student ORDER BY Age ASC , ROLL\_NO DESC;

**Output:**

| **ROLL\_NO** | **NAME** | **ADDRESS** | **PHONE** | **Age** |
| --- | --- | --- | --- | --- |
| 7 | ROHIT | BALURGHAT | XXXXXXXXXX | 18 |
| 4 | DEEP | RAMNAGAR | XXXXXXXXXX | 18 |
| 1 | HARSH | DELHI | XXXXXXXXXX | 18 |
| 8 | NIRAJ | ALIPUR | XXXXXXXXXX | 19 |
| 5 | SAPTARHI | KOLKATA | XXXXXXXXXX | 19 |
| 2 | PRATIK | BIHAR | XXXXXXXXXX | 19 |
| 6 | DHANRAJ | BARABAJAR | XXXXXXXXXX | 20 |
| 3 | RIYANKA | SILIGURI | XXXXXXXXXX | 20 |

In the above output, we can see that first the result is sorted in ascending order according to Age. There are multiple rows of having the same Age. Now, sorting further this result-set according to ROLL\_NO will sort the rows with the same Age according to ROLL\_NO in descending order.

**Note:**

ASC is the default value for the ORDER BY clause. So, if we don’t specify anything after the column name in the ORDER BY clause, the output will be sorted in ascending order by default.

Take another example of the following query will give similar output as the above:   
**Query:**

SELECT \* FROM Student ORDER BY Age , ROLL\_NO DESC;

**Output:**

| **ROLL\_NO** | **NAME** | **ADDRESS** | **PHONE** | **Age** |
| --- | --- | --- | --- | --- |
| 7 | ROHIT | BALURGHAT | XXXXXXXXXX | 18 |
| 4 | DEEP | RAMNAGAR | XXXXXXXXXX | 18 |
| 1 | HARSH | DELHI | XXXXXXXXXX | 18 |
| 8 | NIRAJ | ALIPUR | XXXXXXXXXX | 19 |
| 5 | SAPTARHI | KOLKATA | XXXXXXXXXX | 19 |
| 2 | PRATIK | BIHAR | XXXXXXXXXX | 19 |
| 6 | DHANRAJ | BARABAJAR | XXXXXXXXXX | 20 |
| 3 | RIYANKA | SILIGURI | XXXXXXXXXX | 20 |

**Sorting by column number (instead of name):**

An integer that identifies the number of the column in the SelectItems in the underlying query of the [SELECT statement](https://www.geeksforgeeks.org/sql-order-by/#rrefsqlj41360). Column number must be greater than 0 and not greater than the number of columns in the result table. In other words, if we want to order by a column, that column must be specified in the SELECT list.

The rule checks for ORDER BY clauses that reference select list columns using the column number instead of the column name. The column numbers in the ORDER BY clause impairs the readability of the SQL statement. Further, changing the order of columns in the SELECT list has no impact on the ORDER BY when the columns are referred by names instead of numbers.

**Syntax:**

Order by Column\_Number asc/desc

Here we take an example to sort a database table according to column 1 i.e Roll\_Number. For this a query will be:

**Query:**

CREATE TABLE studentinfo

( Roll\_no INT,

NAME VARCHAR(25),

Address VARCHAR(20),

CONTACTNO BIGINT NOT NULL,

Age INT );

INSERT INTO studentinfo

VALUES (7,'ROHIT','GAZIABAD',9193458625,18),

(4,'DEEP','RAMNAGAR',9193458546,18),

(1,'HARSH','DELHI',9193342625,18),

(8,'NIRAJ','ALIPUR',9193678625,19),

(5,'SAPTARHI','KOLKATA',9193789625,19),

(2,'PRATIK','BIHAR',9193457825,19),

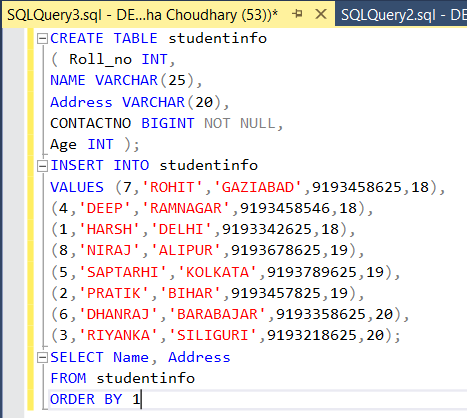
(6,'DHANRAJ','BARABAJAR',9193358625,20),

(3,'RIYANKA','SILIGURI',9193218625,20);

SELECT Name, Address

FROM studentinfo

ORDER BY 1

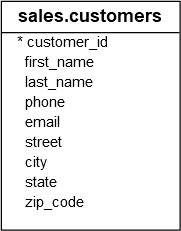


**Output:**



## QL Server ORDER BY clause example

We will use the customers table in the [sample database](https://www.sqlservertutorial.net/sql-server-sample-database/) from the demonstration.



### A) Sort a result set by one column in ascending order

The following statement sorts the customer list by the first name in ascending order:

SELECT

first\_name,

last\_name

FROM

sales.customers

ORDER BY

first\_name;

Code language: SQL (Structured Query Language) (sql)



In this example, because we did not specify ASC or DESC, the ORDER BY clause used ASC by default.

### B) Sort a result set by one column in descending order

The following statement sorts the customer list by the first name in descending order.

SELECT

firstname,

lastname

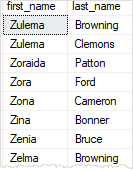
FROM

sales.customers

ORDER BY

first\_name DESC;

Code language: SQL (Structured Query Language) (sql)



In this example, because we specified the DESC explicitly, the ORDER BY clause sorted the result set by values in the first\_name column in descending order.

### C) Sort a result set by multiple columns

The following statement retrieves the first name, last name, and city of the customers. It sorts the customer list by the city first and then by the first name.

SELECT

city,

first\_name,

last\_name

FROM

sales.customers

ORDER BY

city,

first\_name;

Code language: SQL (Structured Query Language) (sql)

### SQL Server ORDER BY - sort by two columns

### D) Sort a result set by multiple columns and different orders

The following statement sorts the customers by the city in descending order and the sort the sorted result set by the first name in ascending order.

SELECT

city,

first\_name,

last\_name

FROM

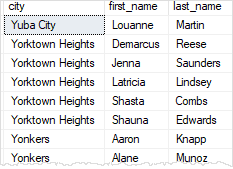
sales.customers

ORDER BY

city DESC,

first\_name ASC;

Code language: SQL (Structured Query Language) (sql)



### E) Sort a result set by a column that is not in the select list

It is possible to sort the result set by a column that does not appear on the select list. For example, the following statement sorts the customer by the state even though the state column does not appear on the select list.

SELECT

city,

first\_name,

last\_name

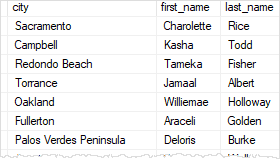
FROM

sales.customers

ORDER BY

state;

Code language: SQL (Structured Query Language) (sql)



Note that the state column is defined in the  customers table. If it was not, then you would have an invalid query.

### F) Sort a result set by an expression

The [LEN()](https://www.sqlservertutorial.net/sql-server-string-functions/sql-server-len-function/) function returns the number of characters of a string. The following statement uses the [LEN()](https://www.sqlservertutorial.net/sql-server-string-functions/sql-server-len-function/) function in the ORDER BY clause to retrieve a customer list sorted by the length of the first name.

SELECT

first\_name,

last\_name

FROM

sales.customers

ORDER BY

LEN(first\_name) DESC;

Code language: SQL (Structured Query Language) (sql)



### G) Sort by ordinal positions of columns

SQL Server allows you to sort the result set based on the ordinal positions of columns that appear in the select list.

The following statement sorts **the customers by first name and last name**. But instead of specifying the column names explicitly, it **uses the ordinal positions of the columns:**

SELECT

first\_name,

last\_name

FROM

sales.customers

ORDER BY

1,

2;

Code language: SQL (Structured Query Language) (sql)

In this example, 1 means the first\_name column and 2 means the last\_name column.

Using the ordinal positions of columns in the ORDER BY clause is considered a bad programming practice for a couple of reasons.

* First, the columns in a table don’t have ordinal positions and need to be referenced by name.
* Second, when you modify the select list, you may forget to make the corresponding changes in the ORDER BY clause.

Therefore, it is a good practice to always specify the column names explicitly in the ORDER BY clause.

In this tutorial, you have learned how to use the SQL Server ORDER BY clause to sort a result set by columns in ascending or descending order.

**Lab-2**

Execute the following DDL statements in your database:

CREATE TABLE categories

( category\_id int NOT NULL,

category\_name char(50) NOT NULL,

CONSTRAINT categories\_pk PRIMARY KEY (category\_id)

);

CREATE TABLE customers

( customer\_id int NOT NULL,

last\_name char(50) NOT NULL,

first\_name char(50) NOT NULL,

favorite\_website char(50),

CONSTRAINT customers\_pk PRIMARY KEY (customer\_id)

);

CREATE TABLE departments

( dept\_id int NOT NULL,

dept\_name char(50) NOT NULL,

CONSTRAINT departments\_pk PRIMARY KEY (dept\_id)

);

CREATE TABLE employees

( employee\_number int NOT NULL,

last\_name char(50) NOT NULL,

first\_name char(50) NOT NULL,

salary int,

dept\_id int,

CONSTRAINT employees\_pk PRIMARY KEY (employee\_number)

);

CREATE TABLE orders

( order\_id int NOT NULL,

customer\_id int,

order\_date date,

CONSTRAINT orders\_pk PRIMARY KEY (order\_id)

);

CREATE TABLE products

( product\_id int NOT NULL,

product\_name char(50) NOT NULL,

category\_id int,

CONSTRAINT products\_pk PRIMARY KEY (product\_id)

);

CREATE TABLE suppliers

( supplier\_id int NOT NULL,

supplier\_name char(50) NOT NULL,

city char(50),

state char(50),

CONSTRAINT suppliers\_pk PRIMARY KEY (supplier\_id)

);

## DML for Tutorial Examples

DML stands for Data Manipulation Language. These are the [INSERT statements](https://www.techonthenet.com/sql/insert.php) that you will need to run in your database to populate the data:

Execute the following DML statements in your database:

INSERT INTO categories

(category\_id, category\_name)

VALUES

(25, 'Deli');

INSERT INTO categories

(category\_id, category\_name)

VALUES

(50, 'Produce');

INSERT INTO categories

(category\_id, category\_name)

VALUES

(75, 'Bakery');

INSERT INTO categories

(category\_id, category\_name)

VALUES

(100, 'General Merchandise');

INSERT INTO categories

(category\_id, category\_name)

VALUES

(125, 'Technology');

INSERT INTO customers

(customer\_id, last\_name, first\_name, favorite\_website)

VALUES

(4000, 'Jackson', 'Joe', 'techonthenet.com');

INSERT INTO customers

(customer\_id, last\_name, first\_name, favorite\_website)

VALUES

(5000, 'Smith', 'Jane', 'digminecraft.com');

INSERT INTO customers

(customer\_id, last\_name, first\_name, favorite\_website)

VALUES

(6000, 'Ferguson', 'Samantha', 'bigactivities.com');

INSERT INTO customers

(customer\_id, last\_name, first\_name, favorite\_website)

VALUES

(7000, 'Reynolds', 'Allen', 'checkyourmath.com');

INSERT INTO customers

(customer\_id, last\_name, first\_name, favorite\_website)

VALUES

(8000, 'Anderson', 'Paige', NULL);

INSERT INTO customers

(customer\_id, last\_name, first\_name, favorite\_website)

VALUES

(9000, 'Johnson', 'Derek', 'techonthenet.com');

INSERT INTO departments

(dept\_id, dept\_name)

VALUES

(500, 'Accounting');

INSERT INTO departments

(dept\_id, dept\_name)

VALUES

(501, 'Sales');

INSERT INTO employees

(employee\_number, last\_name, first\_name, salary, dept\_id)

VALUES

(1001, 'Smith', 'John', 62000, 500);

INSERT INTO employees

(employee\_number, last\_name, first\_name, salary, dept\_id)

VALUES

(1002, 'Anderson', 'Jane', 57500, 500);

INSERT INTO employees

(employee\_number, last\_name, first\_name, salary, dept\_id)

VALUES

(1003, 'Everest', 'Brad', 71000, 501);

INSERT INTO employees

(employee\_number, last\_name, first\_name, salary, dept\_id)

VALUES

(1004, 'Horvath', 'Jack', 42000, 501);

INSERT INTO orders

(order\_id, customer\_id, order\_date)

VALUES

(1,7000,'2016/04/18');

INSERT INTO orders

(order\_id, customer\_id, order\_date)

VALUES

(2,5000,'2016/04/18');

INSERT INTO orders

(order\_id, customer\_id, order\_date)

VALUES

(3,8000,'2016/04/19');

INSERT INTO orders

(order\_id, customer\_id, order\_date)

VALUES

(4,4000,'2016/04/20');

INSERT INTO orders

(order\_id, customer\_id, order\_date)

VALUES

(5,null,'2016/05/01');

INSERT INTO products

(product\_id, product\_name, category\_id)

VALUES

(1,'Pear',50);

INSERT INTO products

(product\_id, product\_name, category\_id)

VALUES

(2,'Banana',50);

INSERT INTO products

(product\_id, product\_name, category\_id)

VALUES

(3,'Orange',50);

INSERT INTO products

(product\_id, product\_name, category\_id)

VALUES

(4,'Apple',50);

INSERT INTO products

(product\_id, product\_name, category\_id)

VALUES

(5,'Bread',75);

INSERT INTO products

(product\_id, product\_name, category\_id)

VALUES

(6,'Sliced Ham',25);

INSERT INTO products

(product\_id, product\_name, category\_id)

VALUES

(7,'Kleenex',null);

INSERT INTO suppliers

(supplier\_id, supplier\_name, city, state)

VALUES

(100, 'Microsoft', 'Redmond', 'Washington');

INSERT INTO suppliers

(supplier\_id, supplier\_name, city, state)

VALUES

(200, 'Google', 'Mountain View', 'California');

INSERT INTO suppliers

(supplier\_id, supplier\_name, city, state)

VALUES

(300, 'Oracle', 'Redwood City', 'California');

INSERT INTO suppliers

(supplier\_id, supplier\_name, city, state)

VALUES

(400, 'Kimberly-Clark', 'Irving', 'Texas');

INSERT INTO suppliers

(supplier\_id, supplier\_name, city, state)

VALUES

(500, 'Tyson Foods', 'Springdale', 'Arkansas');

INSERT INTO suppliers

(supplier\_id, supplier\_name, city, state)

VALUES

(600, 'SC Johnson', 'Racine', 'Wisconsin');

INSERT INTO suppliers

(supplier\_id, supplier\_name, city, state)

VALUES

(700, 'Dole Food Company', 'Westlake Village', 'California');

INSERT INTO suppliers

(supplier\_id, supplier\_name, city, state)

VALUES

(800, 'Flowers Foods', 'Thomasville', 'Georgia');

INSERT INTO suppliers

(supplier\_id, supplier\_name, city, state)

VALUES

(900, 'Electronic Arts', 'Redwood City', 'California');

**Write the queris for the following**

## Example - Sorting Results by relative position

You can also use the SQL ORDER BY clause to sort by relative position in the result set, where the first field in the result set is 1, the second field is 2, the third field is 3, and so on.

In this example, we have a table called products with the following data:

| product\_id | product\_name | category\_id |
| --- | --- | --- |
| 1 | Pear | 50 |
| 2 | Banana | 50 |
| 3 | Orange | 50 |
| 4 | Apple | 50 |
| 5 | Bread | 75 |
| 6 | Sliced Ham | 25 |
| 7 | Kleenex | NULL |

Now enter the following SQL statement:

SELECT product\_id, product\_name

FROM products

WHERE product\_name <> 'Bread'

ORDER BY 1 DESC;

There will be 6 records selected. These are the results that you should see:

| product\_id | product\_name |
| --- | --- |
| 7 | Kleenex |
| 6 | Sliced Ham |
| 4 | Apple |
| 3 | Orange |
| 2 | Banana |
| 1 | Pear |

This example would sort the results by the product\_id field in descending order, since the product\_id field is in position #1 in the result set and would be equivalent to the following SQL ORDER BY clause:

SELECT product\_id, product\_name

FROM products

WHERE product\_name <> 'Bread'

ORDER BY product\_id DESC;

## Example - Using both ASC and DESC attributes

When sorting your result set using the SQL ORDER BY clause, you can use the ASC and DESC attributes in a single [SELECT statement](https://www.techonthenet.com/sql/select.php).

In this example, let's use the same products table as the previous example:

| product\_id | product\_name | category\_id |
| --- | --- | --- |
| 1 | Pear | 50 |
| 2 | Banana | 50 |
| 3 | Orange | 50 |
| 4 | Apple | 50 |
| 5 | Bread | 75 |
| 6 | Sliced Ham | 25 |
| 7 | Kleenex | NULL |

Now enter the following SQL statement:

SELECT \*

FROM products

WHERE product\_id <> 7

ORDER BY category\_id DESC, product\_name ASC;

There will be 6 records selected. These are the results that you should see:

| product\_id | product\_name | category\_id |
| --- | --- | --- |
| 5 | Bread | 75 |
| 4 | Apple | 50 |
| 2 | Banana | 50 |
| 3 | Orange | 50 |
| 1 | Pear | 50 |
| 6 | Sliced Ham | 25 |