SQLite Order By

Summary: in this tutorial, you will learn how to sort a result set of a query using SQLite `ORDER BY` clause.

Introduction to SQLite ORDER BY clause

SQLite stores data in the tables in an unspecified order. It means that the rows in the table may or may not be in the order that they were inserted.

If you use the `SELECT` (https://www.sqlitetutorial.net/sqlite-select/) statement to query data from a table, the order of rows in the result set is unspecified.

To sort the result set, you add the `ORDER BY` clause to the `SELECT` statement as follows:

```
SELECT
   select_list
FROM
   table
ORDER BY
    column_1 ASC,
    column_2 DESC;
```

The `ORDER BY` clause comes after the `FROM` clause. It allows you to sort the result set based on one or more columns in ascending or descending order.

In this syntax, you place the column name by which you want to sort after the `ORDER BY` clause followed by the `ASC` or `DESC` keyword.

- The `ASC` keyword means ascending.
- And the `DESC` keyword means descending.

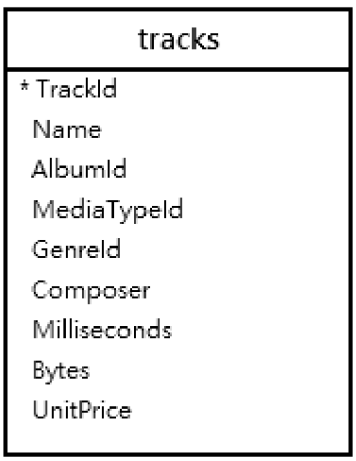
If you don't specify the `ASC` or `DESC` keyword, SQLite sorts the result set using the `ASC` option. In other words, it sorts the result set in the ascending order by default.

In case you want to sort the result set by multiple columns, you use a comma (,) to separate two columns. The `ORDER BY` clause sorts rows using columns or expressions from left to right. In other words, the `ORDER BY` clause sorts the rows using the first column in the list. Then, it sorts the sorted rows using the second column, and so on.

You can sort the result set using a column that does not appear in the select list of the `SELECT` clause.

SQLite ORDER BY clause example

Let's take the `tracks` table in the sample database (https://www.sqlitetutorial.net/sqlite-sample-database/) for the demonstration.



Suppose, you want to get data from name, milliseconds, and album id columns, you use the following statement:

```
SELECT
        name,
        milliseconds,
        albumid
FROM
        tracks;
```

Try It ›

Name	Milliseconds	AlbumId
For Those About To Rock (We Salute You)	343719	1
Balls to the Wall	342562	2
Fast As a Shark	230619	3
Restless and Wild	252051	3
Princess of the Dawn	375418	3
Put The Finger On You	205662	1
Let's Get It Up	233926	1

(https://www.sqlitetutorial.net/wp-content/uploads/2015/11/tracks-table-data-without-sorting.jpg)

The `SELECT` statement that does not use `ORDER BY` clause returns a result set that is not in any order.

Suppose you want to sort the result set based on `AlbumId` column in ascending order, you use the following statement:

```
SELECT
        name,
        milliseconds,
        albumid
FROM
        tracks
ORDER BY
        albumid ASC;
```

Try It ›

Name	Milliseconds	AlbumId
For Those About To Rock (We Salute You)	343719	1
Put The Finger On You	205662	1
Let's Get It Up	233926	1
Inject The Venom	210834	1
Snowballed	203102	1
Evil Walks	263497	1
C.O.D.	199836	1
Breaking The Rules	263288	1
Night Of The Long Knives	205688	1
Spellbound	270863	1
Balls to the Wall	342562	2
Fast As a Shark	230619	3
Restless and Wild	252051	3
Princess of the Dawn	375418	3
Go Down	331180	4
Dog Eat Dog	215196	4

(https://www.sqlitetutorial.net/wp-content/uploads/2015/11/SQLite-ORDER-BY-example.jpg)

The result set now is sorted by the `AlbumId` column in ascending order as shown in the screenshot.

SQLite uses `ASC` by default so you can omit it in the above statement as follows:

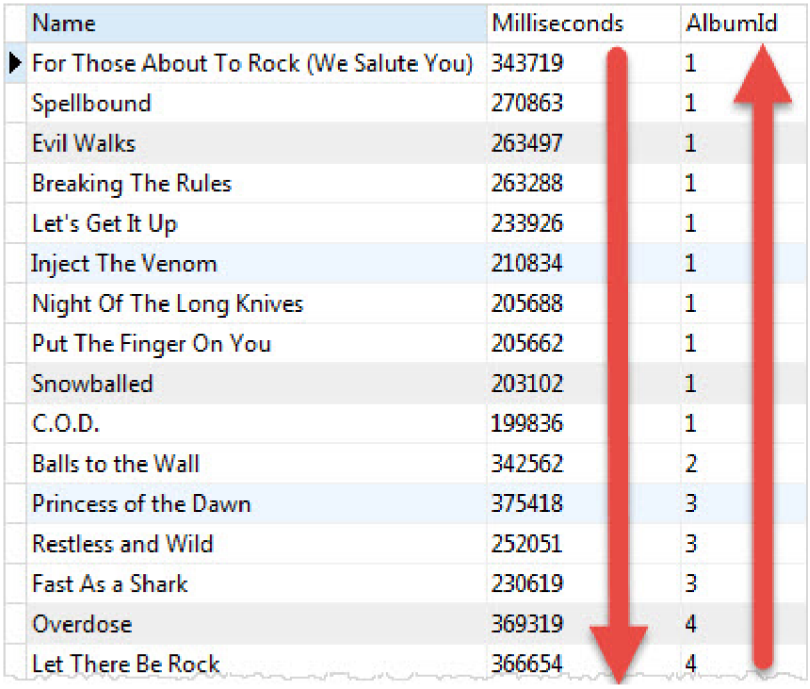
```
SELECT
        name,
        milliseconds,
        albumid
FROM
        tracks
ORDER BY
        albumid;
```

Try It ›

Suppose you want to sort the sorted result (by `AlbumId`) above by the `Milliseconds` column in descending order. In this case, you need to add the `Milliseconds` column to the `ORDER BY` clause as follows:

```
SELECT
        name,
        milliseconds,
        albumid
FROM
        tracks
ORDER BY
        albumid ASC,
        milliseconds DESC;
```

Try It ›



Name	Milliseconds	AlbumId
For Those About To Rock (We Salute You)	343719	1
Spellbound	270863	1
Evil Walks	263497	1
Breaking The Rules	263288	1
Let's Get It Up	233926	1
Inject The Venom	210834	1
Night Of The Long Knives	205688	1
Put The Finger On You	205662	1
Snowballed	203102	1
C.O.D.	199836	1
Balls to the Wall	342562	2
Princess of the Dawn	375418	3
Restless and Wild	252051	3
Fast As a Shark	230619	3
Overdose	369319	4
Let There Be Rock	366654	4

(https://www.sqlitetutorial.net/wp-content/uploads/2015/11/SQLite-ORDER-BY-multiple-columns-example.jpg)

SQLite sorts rows by `AlbumId` column in ascending order first. Then, it sorts the sorted result set by the `Milliseconds` column in descending order.

If you look at the tracks of the album with `AlbumId` 1, you find that the order of tracks changes between the two statements.

SQLite ORDER BY with the column position

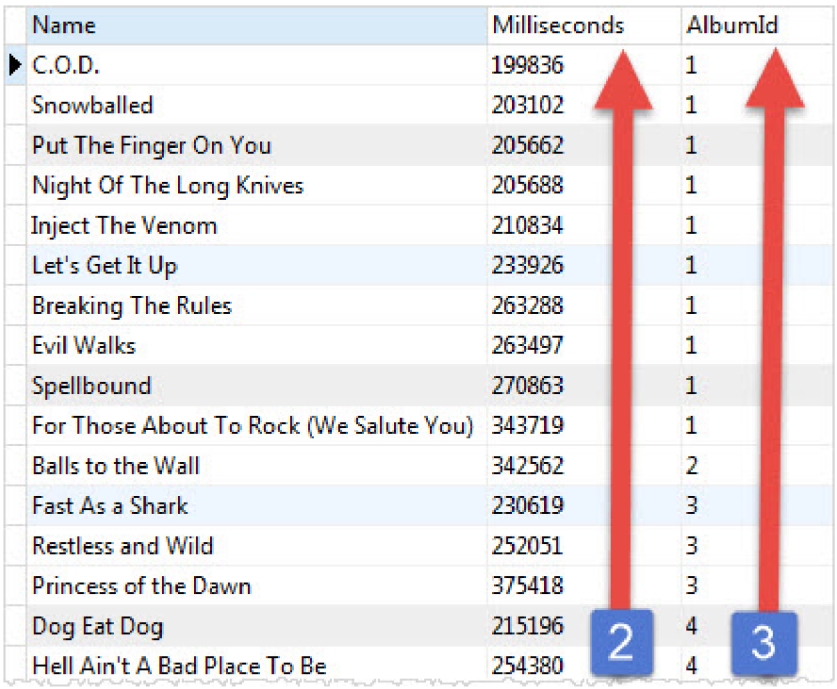
Instead of specifying the names of columns, you can use the column's position in the `ORDER BY` clause.

For example, the following statement sorts the tracks by both `albumid` (3rd column) and `milliseconds` (2nd column) in ascending order.

```
SELECT
        name,
        milliseconds,
        albumid
FROM
        tracks
ORDER BY
         3,2;
```

Try It ›

The number 3 and 2 refers to the `AlbumId` and `Milliseconds` in the column list that appears in the `SELECT` clause.



(https://www.sqlitetutorial.net/wp-content/uploads/2015/11/SQLite-ORDER-BY-multiple-columns-by-positions.jpg)

Sorting NULLs

In the database world, NULL is special. It denotes that the information missing or the data is not applicable.

Suppose you want to store the birthday of an artist in a table. At the time of saving the artist's record, you don't have the birthday information.

To represent the unknown birthday information in the database, you may use a special date like `01.01.1900` or an `''` empty string. However, both of these values do not clearly show that the birthday is unknown.

NULL was invented to resolve this issue. Instead of using a special value to indicate that the information is missing, NULL is used.

NULL is special because you cannot compare it with another value. Simply put, if the two pieces of information are unknown, you cannot compare them.

NULL is even cannot be compared with itself; NULL is not equal to itself so `NULL = NULL` always results in false.

When it comes to sorting, SQLite considers NULL to be smaller than any other value.

It means that NULLs will appear at the beginning of the result set if you use ASC or at the end of the result set when you use DESC.

SQLite 3.30.0 added the `NULLS FIRST` and `NULLS LAST` options to the `ORDER BY` clause. The `NULLS FIRST` option specifies that the NULLs will appear at the beginning of the result set while the `NULLS LAST` option place NULLs at the end of the result set.

The following example uses the `ORDER BY` clause to sort tracks by composers:

```
SELECT
    TrackId,
    Name,
    Composer
FROM
   tracks
ORDER BY
   Composer;
```

First, you see that NULLs appear at the beginning of the result set because SQLite treats them as the lowest values. When you scroll down the result, you will see other values:

The following example uses the `NULLS LAST` option to place NULLs after other values:

```
SELECT
    TrackId,
```

```
    Name,
    Composer
FROM
    tracks
ORDER BY
    Composer NULLS LAST;
```

If you scroll down the output, you will see that NULLs are placed at the end of the result set:

In this tutorial, you have learned how to use the SQLite `ORDER BY` clause to sort the result set using a single column, multiple columns in ascending and descending orders.