SQLite Group By

Summary: in this tutorial, you will learn how to use SQLite `GROUP BY` clause to make a set of summary rows from a set of rows.

Introduction to SQLite GROUP BY clause

The `GROUP BY` clause is an optional clause of the SELECT (https://www.sqlitetutorial.net/sqlite-select/) statement. The `GROUP BY` clause a selected group of rows into summary rows by values of one or more columns.

The `GROUP BY` clause returns one row for each group. For each group, you can apply an aggregate function such as MIN (https://www.sqlitetutorial.net/sqlite-min/) , MAX (https://www.sqlitetutorial.net/sqlite-max/) , SUM (https://www.sqlitetutorial.net/sqlite-sum/) , COUNT (https://www.sqlitetutorial.net/sqlite-count-function/) , or AVG (https://www.sqlitetutorial.net/sqlite-avg/) to provide more information about each group.

The following statement illustrates the syntax of the SQLite `GROUP BY` clause.

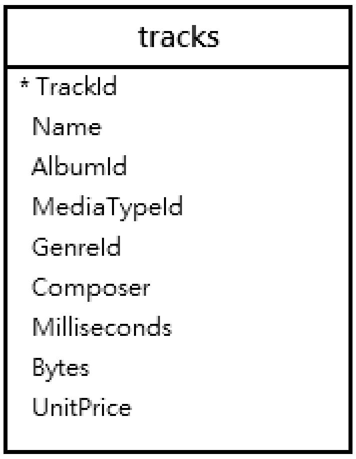
```
SELECT
    column_1,
    aggregate_function(column_2)
FROM
    table
GROUP BY
    column_1,
    column_2;
```

The `GROUP BY` clause comes after the `FROM` clause of the `SELECT` statement. In case a statement contains a WHERE (https://www.sqlitetutorial.net/sqlite-where/) clause, the `GROUP BY` clause must come after the `WHERE` clause.

Following the `GROUP BY` clause is a column or a list of comma-separated columns used to specify the group.

SQLite GROUP BY examples

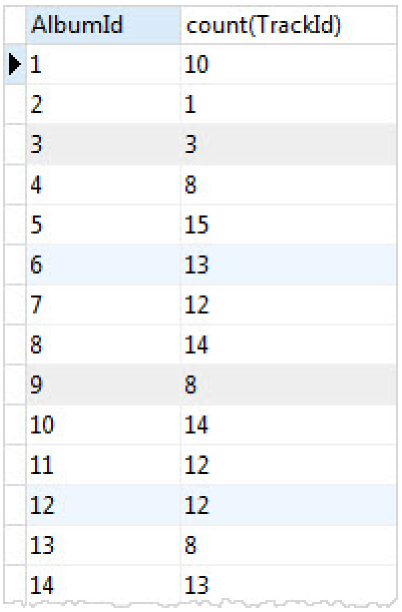
We use the `tracks` table from the [sample database (https://www.sqlitetutorial.net/sqlite-sample-database/)](https://www.sqlitetutorial.net/sqlite-sample-database/) for the demonstration.



SQLite `GROUP BY` clause with `COUNT` function

The following statement returns the album id and the number of tracks per album. It uses the `GROUP BY` clause to groups tracks by album and applies the [`COUNT()` (https://www.sqlitetutorial.net/sqlite-count-function/)](https://www.sqlitetutorial.net/sqlite-count-function/) function to each group.

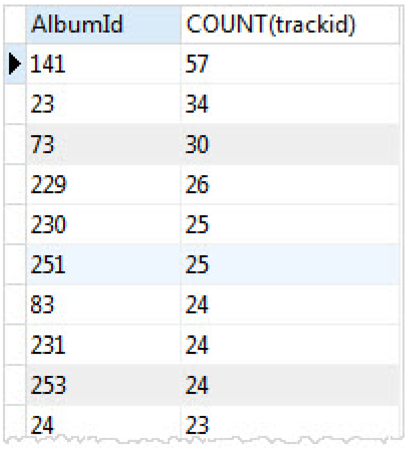
```
SELECT
	albumid,
	COUNT(trackid)
FROM
	tracks
GROUP BY
	albumid;
```



AlbumId	count(TrackId)
1	10
2	1
3	3
4	8
5	15
6	13
7	12
8	14
9	8
10	14
11	12
12	12
13	8
14	13

You can use the ORDER BY (https://www.sqlitetutorial.net/sqlite-order-by/) clause to sort the groups as follows:

```
SELECT
	albumid,
	COUNT(trackid)
FROM
	tracks
GROUP BY
	albumid
ORDER BY COUNT(trackid) DESC;
```

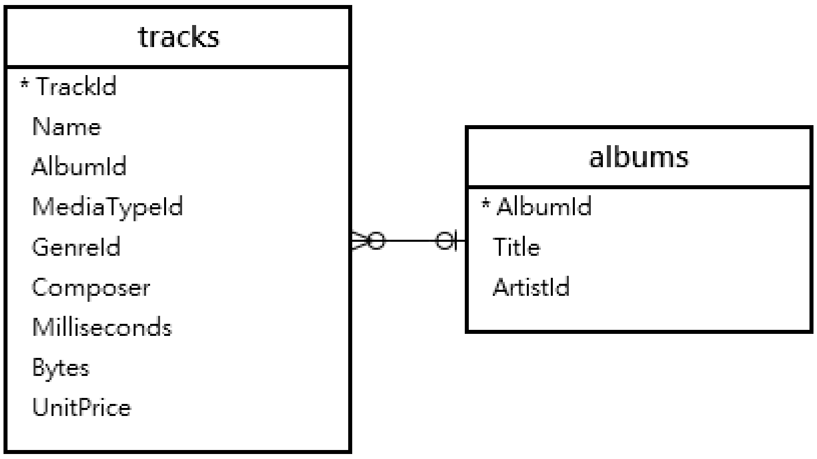


AlbumId	COUNT(trackid)
141	57
23	34
73	30
229	26
230	25
251	25
83	24
231	24
253	24
24	23

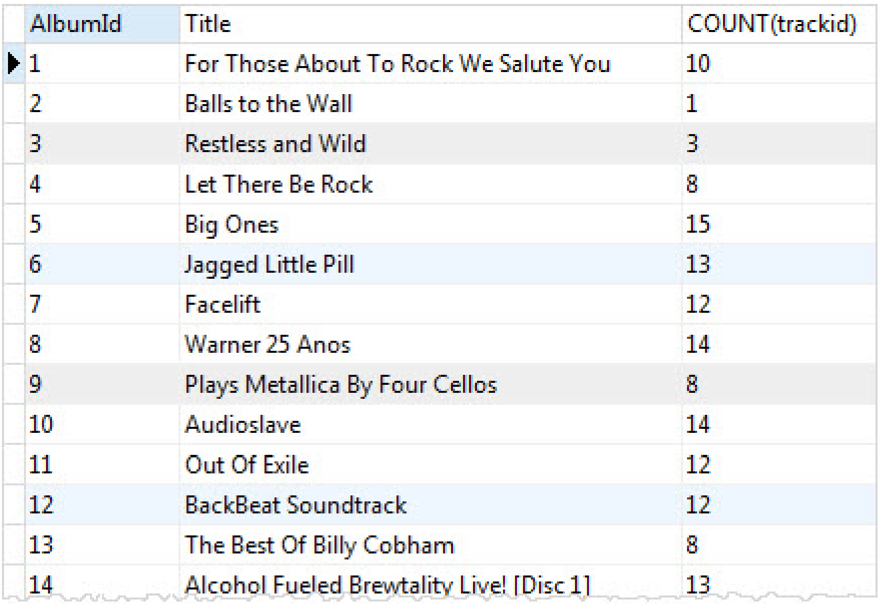
SQLite `GROUP BY` and `INNER JOIN` clause

You can query data from multiple tables using the INNER JOIN (https://www.sqlitetutorial.net/sqlite-inner-join/) clause, then use the `GROUP BY` clause to group rows into a set of summary rows.

For example, the following statement joins the `tracks` table with the `albums` table to get the album's titles and uses the `GROUP BY` clause with the `COUNT` function to get the number of tracks per album.



```
SELECT
        tracks.albumid,
        title,
        COUNT(trackid)
FROM
        tracks
INNER JOIN albums ON albums.albumid = tracks.albumid
GROUP BY
        tracks.albumid;
```

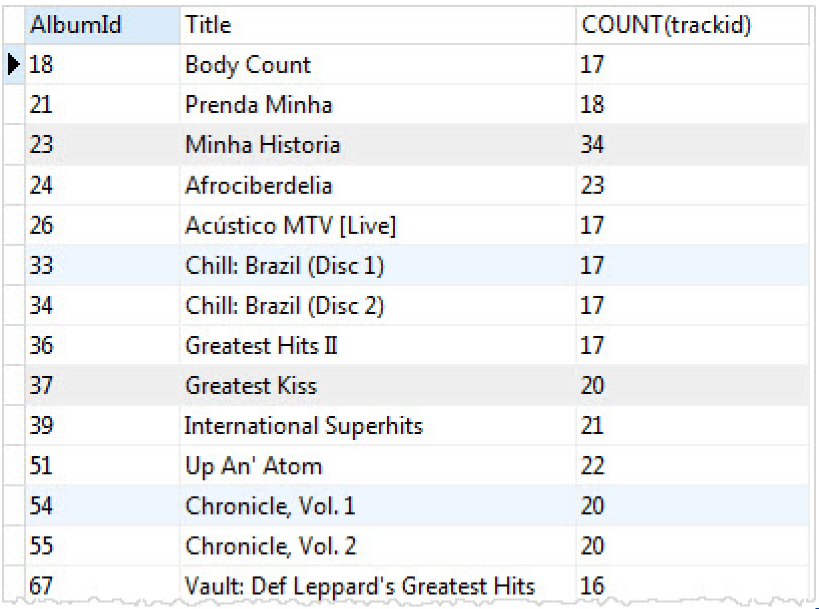


AlbumId	Title	COUNT(trackid)
1	For Those About To Rock We Salute You	10
2	Balls to the Wall	1
3	Restless and Wild	3
4	Let There Be Rock	8
5	Big Ones	15
6	Jagged Little Pill	13
7	Facelift	12
8	Warner 25 Anos	14
9	Plays Metallica By Four Cellos	8
10	Audioslave	14
11	Out Of Exile	12
12	BackBeat Soundtrack	12
13	The Best Of Billy Cobham	8
14	Alcohol Fueled Brewtality Live! [Disc 1]	13

SQLite GROUP BY with HAVING clause

To filter groups, you use the `GROUP BY` with HAVING (https://www.sqlitetutorial.net/sqlite-having/) clause. For example, to get the albums that have more than 15 tracks, you use the following statement:

```
SELECT
        tracks.albumid,
        title,
        COUNT(trackid)
FROM
        tracks
INNER JOIN albums ON albums.albumid = tracks.albumid
GROUP BY
        tracks.albumid
HAVING COUNT(trackid) > 15;
```



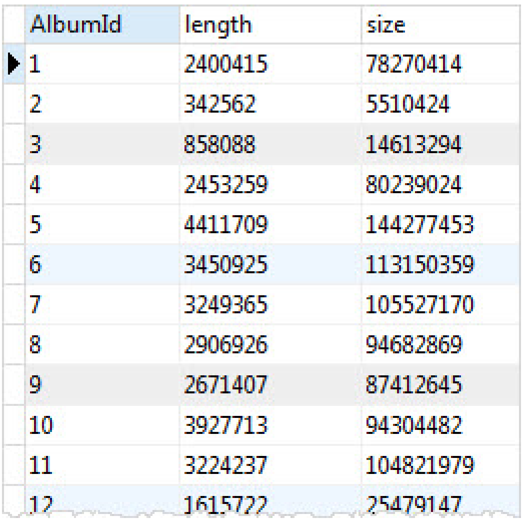
AlbumId	Title	COUNT(trackid)
18	Body Count	17
21	Prenda Minha	18
23	Minha Historia	34
24	Afrociberdelia	23
26	Acústico MTV [Live]	17
33	Chill: Brazil (Disc 1)	17
34	Chill: Brazil (Disc 2)	17
36	Greatest Hits II	17
37	Greatest Kiss	20
39	International Superhits	21
51	Up An' Atom	22
54	Chronicle, Vol. 1	20
55	Chronicle, Vol. 2	20
67	Vault: Def Leppard's Greatest Hits	16

(https://cdn.sqlitetutorial.net/wp-content/uploads/2015/11/SQLite-GROUP-BY-with-HAVING-clause.jpg)

SQLite GROUP BY clause with SUM function example

You can use the SUM (https://www.sqlitetutorial.net/sqlite-sum/) function to calculate total per group. For example, to get total length and bytes for each album, you use the `SUM` function to calculate total milliseconds and bytes.

```
SELECT
	albumid,
	SUM(milliseconds) length,
	SUM(bytes) size
FROM
	tracks
GROUP BY
	albumid;
```

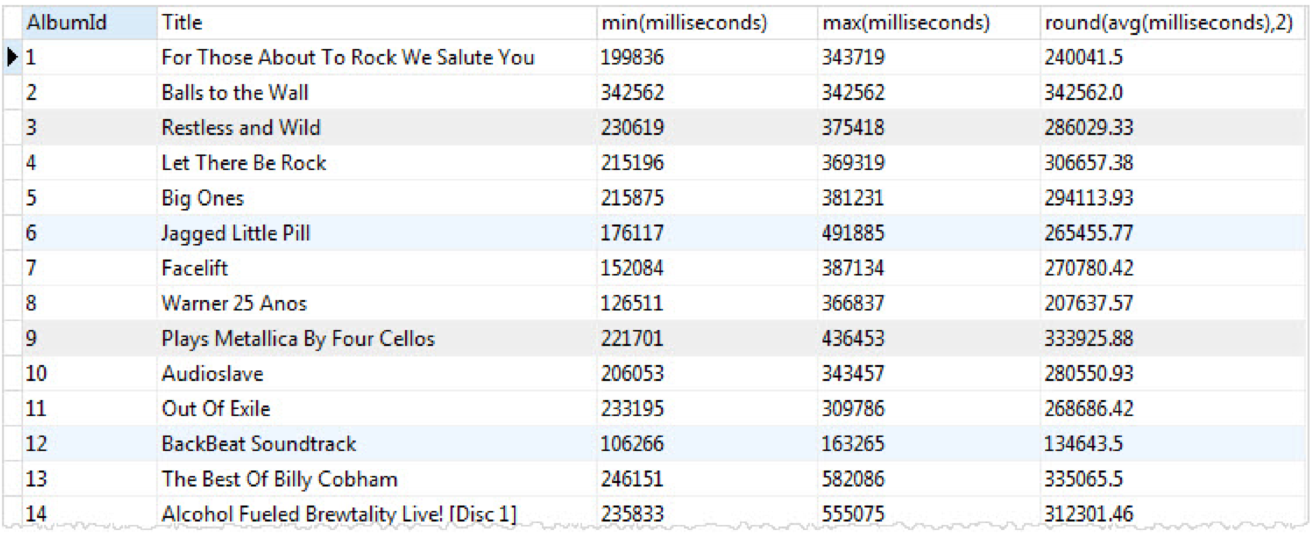


AlbumId	length	size
1	2400415	78270414
2	342562	5510424
3	858088	14613294
4	2453259	80239024
5	4411709	144277453
6	3450925	113150359
7	3249365	105527170
8	2906926	94682869
9	2671407	87412645
10	3927713	94304482
11	3224237	104821979
12	1615722	25479147

SQLite GROUP BY with MAX, MIN, and AVG functions

The following statement returns the album id, album title, maximum length, minimum length, and the average length of tracks in the `tracks` table.

```
SELECT
	tracks.albumid,
	title,
	min(milliseconds),
	max(milliseconds),
	round(avg(milliseconds),2)
FROM
	tracks
INNER JOIN albums ON albums.albumid = tracks.albumid
GROUP BY
	tracks.albumid;
```



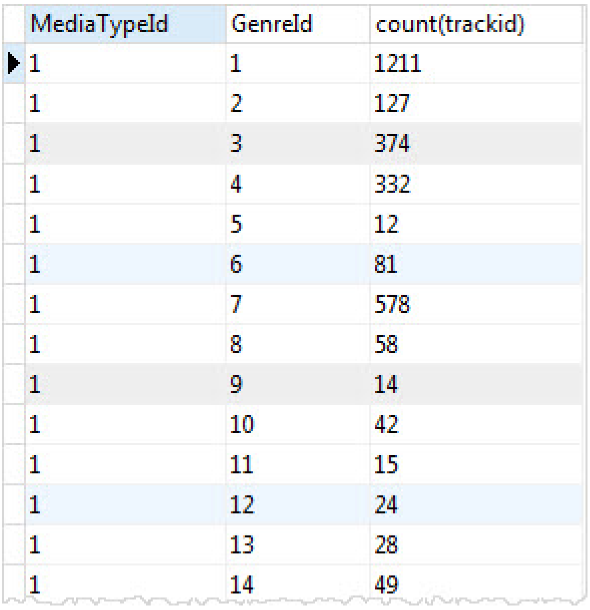
AlbumId	Title	min(milliseconds)	max(milliseconds)	round(avg(milliseconds),2)
1	For Those About To Rock We Salute You	199836	343719	240041.5
2	Balls to the Wall	342562	342562	342562.0
3	Restless and Wild	230619	375418	286029.33
4	Let There Be Rock	215196	369319	306657.38
5	Big Ones	215875	381231	294113.93
6	Jagged Little Pill	176117	491885	265455.77
7	Facelift	152084	387134	270780.42
8	Warner 25 Anos	126511	366837	207637.57
9	Plays Metallica By Four Cellos	221701	436453	333925.88
10	Audioslave	206053	343457	280550.93
11	Out Of Exile	233195	309786	268686.42
12	BackBeat Soundtrack	106266	163265	134643.5
13	The Best Of Billy Cobham	246151	582086	335065.5
14	Alcohol Fueled Brewtality Live! [Disc 1]	235833	555075	312301.46

SQLite GROUP BY multiple columns example

In the previous example, we have used one column in the GROUP BY clause. SQLite allows you to group rows by multiple columns.

For example, to group tracks by media type and genre, you use the following statement:

```
SELECT
   MediaTypeId,
   GenreId,
   COUNT(TrackId)
FROM
   tracks
GROUP BY
   MediaTypeId,
   GenreId;
```

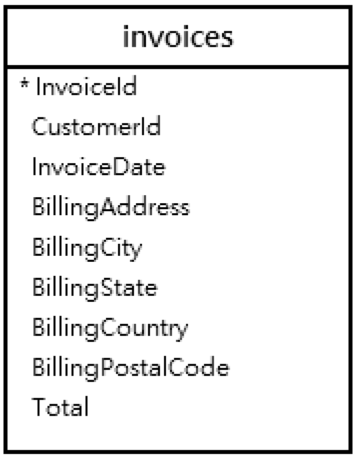


MediaTypeId	GenreId	count(trackid)
1	1	1211
1	2	127
1	3	374
1	4	332
1	5	12
1	6	81
1	7	578
1	8	58
1	9	14
1	10	42
1	11	15
1	12	24
1	13	28
1	14	49

SQLite uses the combination of values of `MediaTypeId` and `GenreId` columns as a group e.g., (1,1) and (1,2). It then applies the COUNT (https://www.sqlitetutorial.net/sqlite-count-function/) function to return the number of tracks in each group.

SQLite GROUP BY date example

See the following invoices table from the sample database:



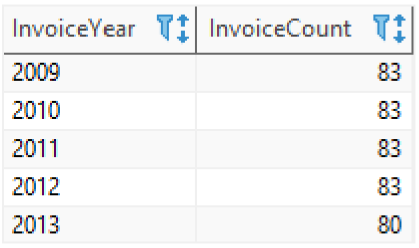
invoices
* InvoiceId
CustomerId
InvoiceDate
BillingAddress
BillingCity
BillingState
BillingCountry
BillingPostalCode
Total

The following statement returns the number of invoice by years.

```
SELECT
    STRFTIME('%Y', InvoiceDate) InvoiceYear,
    COUNT(InvoiceId) InvoiceCount
FROM
    invoices
GROUP BY
```

```
    STRFTIME('%Y', InvoiceDate)
ORDER BY
    InvoiceYear;
```

Here is the output:



InvoiceYear	InvoiceCount
2009	83
2010	83
2011	83
2012	83
2013	80

In this example:

- The function `STRFTIME('%Y', InvoiceDate)` returns a year from a date string.
- The `GROUP BY` clause groups the invoices by years.
- The function `COUNT()` returns the number of invoice in each year (or group).

In this tutorial, you have learned how to use the SQLite `GROUP BY` clause to group rows into a set of summary rows.