SQLITE TUTORIAL

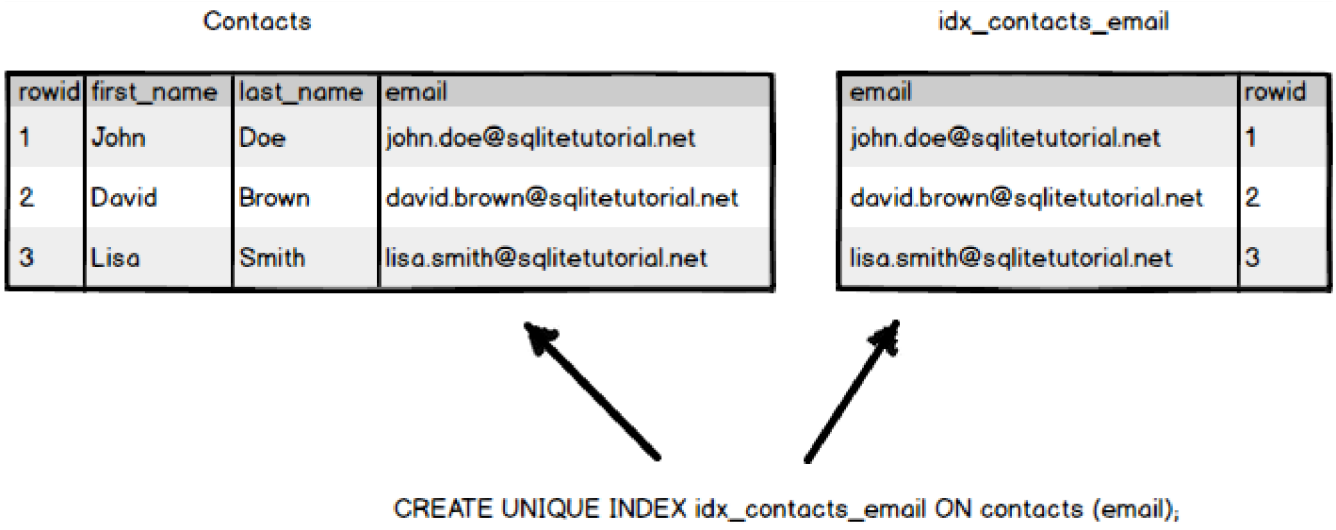
SQLite Index

Summary: in this tutorial, you will learn how to use SQLite indexes to query data faster, speed up sort operation, and enforce unique constraints.

What is an index?

In relational databases, a table is a list of rows. In the same time, each row has the same column structure that consists of cells. Each row also has a consecutive rowid (https://www.sqlitetutorial.net/sqlite-autoincrement/) sequence number used to identify the row. Therefore, you can consider a table as a list of pairs: (rowid, row).

Unlike a table, an index has an opposite relationship: (row, rowid). An index is an additional data structure that helps improve the performance of a query.



SQLite uses B-tree for organizing indexes. Note that **B** stands for balanced, B-tree is a balanced tree, not a binary tree.

The B-tree keeps the amount of data at both sides of the tree balanced so that the number of levels that must be traversed to locate a row is always in the same approximate number. In addition, querying using equality (=) and ranges (>, >=, <,<=) on the B-tree indexes are very efficient.

How does an index work

Each index must be associated with a specific table. An index consists of one or more columns, but all columns of an index must be in the same table. A table may have multiple indexes.

Whenever you create an index, SQLite creates a B-tree structure to hold the index data.

The index contains data from the columns that you specify in the index and the corresponding `rowid` value. This helps SQLite quickly locate the row based on the values of the indexed columns.

Imagine an index in the database like an index of a book. By looking at the index, you can quickly identify page numbers based on the keywords.

SQLite CREATE INDEX statement

To create an index, you use the `CREATE INDEX` statement with the following syntax:

```
CREATE [UNIQUE] INDEX index_name
ON table_name(column_list);
```

To create an index, you specify three important information:

- The name of the index after the `CREATE INDEX` keywords.
- The name of the table to the index belongs.
- A list of columns of the index.

In case you want to make sure that values in one or more columns are unique like email and phone, you use the `UNIQUE` option in the `CREATE INDEX` statement. The `CREATE UNIQUE INDEX` creates a new unique index.

SQLite UNIQUE index example

Let's create a new table (https://www.sqlitetutorial.net/sqlite-create-table/) named `contacts` for demonstration.

```
CREATE TABLE contacts (
        first_name text NOT NULL,
        last_name text NOT NULL,
```

```
        email text NOT NULL
);
```

Try It ❯

Suppose, you want to enforce that the email is unique, you create a unique index as follows:

```
CREATE UNIQUE INDEX idx_contacts_email
ON contacts (email);
```

Try It ❯

To test this.

First, insert a row into (https://www.sqlitetutorial.net/sqlite-insert/) the `contacts` table.

```
INSERT INTO contacts (first_name, last_name, email)
VALUES('John','Doe','john.doe@sqlitetutorial.net');
```

Try It ❯

Second, insert another row with a duplicate email.

```
INSERT INTO contacts (first_name, last_name, email)
VALUES('Johny','Doe','john.doe@sqlitetutorial.net');
```

Try It ❯

SQLite issued an error message indicating that the unique index has been violated. Because when you inserted the second row, SQLite checked and made sure that the email is unique across of rows in `email` of the `contacts` table.

Let's insert two more rows into the `contacts` table.

```
INSERT INTO contacts (first_name, last_name, email)
VALUES('David','Brown','david.brown@sqlitetutorial.net'),
      ('Lisa','Smith','lisa.smith@sqlitetutorial.net');
```

Try It ❯

If you query data (https://www.sqlitetutorial.net/sqlite-select/) from the `contacts` table based on a specific email, SQLite will use the index to locate the data. See the following statement:

```
SELECT
        first_name,
        last_name,
        email
FROM
        contacts
WHERE
        email = 'lisa.smith@sqlitetutorial.net';
```

Try It ❯

To check if SQLite uses the index or not, you use the `EXPLAIN QUERY PLAN` statement as follows:

```
EXPLAIN QUERY PLAN
SELECT
        first_name,
        last_name,
        email
FROM
        contacts
WHERE
        email = 'lisa.smith@sqlitetutorial.net';
```

Try It ❯

SQLite multicolumn index example

If you create an index that consists of one column, SQLite uses that column as the sort key. In case you create an index that has multiple columns, SQLite uses the additional columns as the second, third, … as the sort keys.

SQLite sorts the data (https://www.sqlitetutorial.net/sqlite-order-by/) on the multicolumn index by the first column specified in the `CREATE INDEX` statement. Then, it sorts the duplicate values by the second column, and so on.

Therefore, the column order is very important when you create a multicolumn index.

To utilize a multicolumn index, the query must contain the condition that has the same column order as defined in the index.

The following statement creates a multicolumn index on the `first_name` and `last_name` columns of the `contacts` table:

```
CREATE INDEX idx_contacts_name
ON contacts (first_name, last_name);
```

Try It ❯

If you query the `contacts` table with one of the following conditions in the `WHERE` (https://www.sqlitetutorial.net/sqlite-where/) clause, SQLite will utilize the multicolumn index to search for data.

1) filter data by the `first_name` column.

```
WHERE
        first_name = 'John';
```

2)filter data by both `first_name` and `last_name` columns:

```
WHERE
        first_name = 'John' AND last_name = 'Doe';
```

However, SQLite will not use the multicolumn index if you use one of the following conditions.

1)filter by the `last_name` column only.

```
WHERE
        last_name = 'Doe';
```

2) filter by `first_name` OR `last_name` columns.

```
last_name = 'Doe' OR first_name = 'John';
```

SQLite Show Indexes

To find all indexes associated with a table, you use the following command:

```
PRAGMA index_list('table_name');
```

For example, this statement shows all the indexes of the `contacts` table:

```
PRAGMA index_list('playlist_track');
```

Here is the output:

To get the information about the columns in an index, you use the following command:

```
PRAGMA index_info('idx_contacts_name');
```

This example returns the column list of the index `idx_contacts_name` :

Another way to get all indexes from a database is to query from the `sqlite_master` table:

```
SELECT
   type,
```

```
    name,
    tbl_name,
    sql
FROM
    sqlite_master
WHERE
    type= 'index';
```

SQLite DROP INDEX statement

To remove an index from a database, you use the `DROP INDEX` statement as follows:

```
DROP INDEX [IF EXISTS] index_name;
```

In this syntax, you specify the name of the index that you want to drop after the `DROP INDEX` keywords. The `IF EXISTS` option removes an index only if it exists.

For example, you use the following statement to remove the `idx_contacts_name` index:

```
DROP INDEX idx_contacts_name;
```

Try It ❯

The `idx_contacts_name` index is removed completely from the database.

In this tutorial, you have learned about SQLite index and how to utilize indexes for improving the performance of query or enforcing unique constraints.