SQLite Primary Key

Summary: in this tutorial, you will learn how to use SQLite `PRIMARY KEY` constraint to define a primary key for a table.

Introduction to SQLite primary key

A primary key is a column or group of columns used to identify the uniqueness of rows in a table. Each table has one and only one primary key.

SQLite allows you to define primary key in two ways:

First, if the primary key has only one column, you use the `PRIMARY KEY` column constraint to define the primary key as follows:

```
CREATE TABLE table_name(
   column_1 INTEGER NOT NULL PRIMARY KEY,
   ...
);
```

Second, in case primary key consists of two or more columns, you use the `PRIMARY KEY` table constraint to define the primary as shown in the following statement.

```
CREATE TABLE table_name(
   column_1 INTEGER NOT NULL,
   column_2 INTEGER NOT NULL,
   ...
   PRIMARY KEY(column_1,column_2,...)
);
```

In SQL standard, the primary key column must not contain `NULL` values. It means that the primary key column has an implicit `NOT NULL` constraint.

However, to make the current version of SQLite compatible with the earlier version, SQLite allows the primary key column to contain `NULL` values.

SQLite primary key and rowid table

When you create a table without specifying the `WITHOUT ROWID` option, SQLite adds an implicit column called `rowid` that stores 64-bit signed integer. The `rowid` column is a key that uniquely identifies the rows in the table. Tables that have `rowid` columns are called `rowid` tables.

If a table has the primary key that consists of one column, and that column is defined as `INTEGER` then this primary key column becomes an *alias* for the `rowid` column.

Notice that if you assign another integer type such as `BIGINT` and `UNSIGNED INT` to the primary key column, this column will not be an alias for the `rowid` column.

Because the `rowid` table organizes its data as a B-tree, querying and sorting data of a `rowid` table are very fast. It is faster than using a primary key which is not an alias of the `rowid` .

Another important note is that if you declare a column with the `INTEGER` type and `PRIMARY KEY DESC` clause, this column will not become an alias for the `rowid` column:

```
CREATE TABLE table(
   pk INTEGER PRIMARY KEY DESC,
   ...
);
```

Creating SQLite primary key examples

The following statement creates a table named `countries` which has `country_id` column as the primary key.

```
CREATE TABLE countries (
   country_id INTEGER PRIMARY KEY,
   name TEXT NOT NULL
);
```

Try It ›

Because the primary key of the `countries` table has only one column, we defined the primary key using `PRIMARY KEY` column constraint.

It is possible to use the `PRIMARY KEY` table constraint to define the primary key that consists of one column as shown in the following statement:

```
CREATE TABLE languages (
   language_id INTEGER,
   name TEXT NOT NULL,
   PRIMARY KEY (language_id)
);
```

Try It ›

However, for tables that the primary keys consist of more than one column, you must use `PRIMARY KEY` table constraint to define primary keys.

The following statement creates the `country_languages` table whose primary key consists of two columns.

```
CREATE TABLE country_languages (
        country_id INTEGER NOT NULL,
        language_id INTEGER NOT NULL,
        PRIMARY KEY (country_id, language_id),
        FOREIGN KEY (country_id) REFERENCES countries (country_id)
            ON DELETE CASCADE ON UPDATE NO ACTION,
        FOREIGN KEY (language_id) REFERENCES languages (language_id)
            ON DELETE CASCADE ON UPDATE NO ACTION
);
```

Try It ›

Adding SQLite primary key example

Unlike other database systems e.g., MySQL and PostgreSQL, you cannot use the `ALTER TABLE` statement to add a primary key to an existing table.

You need to follow these steps to work around the limitation:

1. First, set the foreign key constarint check off.
2. Next, rename the table to another table name (old_table)

3. Then, create a new table (table) with exact structure of the table that you have been renamed.
4. After that, copy data from the old_table to the table.
5. Finally, turn on the foreign key constraint check on

See the following statements:

```
PRAGMA foreign_keys=off;

BEGIN TRANSACTION;

ALTER TABLE table RENAME TO old_table;

-- define the primary key constraint here
CREATE TABLE table ( ... );

INSERT INTO table SELECT * FROM old_table;

COMMIT;

PRAGMA foreign_keys=on;
```

Try It ›

The `BEGIN TRANSACTION` starts a new transaction. It ensures that all subsequent statements execute successfully or nothing executes at all.

The `COMMIT` statement commits all the statements.

Let's create a table name `cities` without a primary key.

```
CREATE TABLE cities (
   id INTEGER NOT NULL,
   name text NOT NULL
);

INSERT INTO cities (id, name)
VALUES(1, 'San Jose');
```

Try It ›

In order to add the primary key to the `cities` table, you perform the following steps:

```
PRAGMA foreign_keys=off;

BEGIN TRANSACTION;

ALTER TABLE cities RENAME TO old_cities;

CREATE TABLE cities (
   id INTEGER NOT NULL PRIMARY KEY,
   name TEXT NOT NULL
);

INSERT INTO cities
SELECT * FROM old_cities;

DROP TABLE old_cities;

COMMIT;

PRAGMA foreign_keys=on;
```

Try It ›

If you use SQLite GUI tool, you can use the following statement to show the table's information.

```
PRAGMA table_info([cities]);
```

Try It ›

In this tutorial, you have learned use the SQLite `PRIMARY KEY` constraint to define the primary key for a table.