

SQLITE - INDEXES

http://www.tutorialspoint.com/sqlite/sqlite_indexes.htm

Copyright © tutorialspoint.com

Indexes are special lookup tables that the database search engine can use to speed up data retrieval. Simply put, an index is a pointer to data in a table. An index in a database is very similar to an index in the back of a book.

For example, if you want to reference all pages in a book that discuss a certain topic, you first refer to the index, which lists all topics alphabetically and are then referred to one or more specific page numbers.

An index helps speed up SELECT queries and WHERE clauses, but it slows down data input, with UPDATE and INSERT statements. Indexes can be created or dropped with no effect on the data.

Creating an index involves the CREATE INDEX statement, which allows you to name the index, to specify the table and which column or columns to index, and to indicate whether the index is in ascending or descending order.

Indexes can also be unique, similar to the UNIQUE constraint, in that the index prevents duplicate entries in the column or combination of columns on which there's an index.

The CREATE INDEX Command:

The basic syntax of **CREATE INDEX** is as follows:

```
CREATE INDEX index_name ON table_name;
```

Single-Column Indexes:

A single-column index is one that is created based on only one table column. The basic syntax is as follows:

```
CREATE INDEX index_name  
ON table_name (column_name);
```

Unique Indexes:

Unique indexes are used not only for performance, but also for data integrity. A unique index does not allow any duplicate values to be inserted into the table. The basic syntax is as follows:

```
CREATE UNIQUE INDEX index_name  
on table_name (column_name);
```

Composite Indexes:

A composite index is an index on two or more columns of a table. The basic syntax is as follows:

```
CREATE INDEX index_name  
on table_name (column1, column2);
```

Whether to create a single-column index or a composite index, take into consideration the columns that you may use very frequently in a query's WHERE clause as filter conditions.

Should there be only one column used, a single-column index should be the choice. Should there be two or more columns that are frequently used in the WHERE clause as filters, the composite index would be the best choice.

Implicit Indexes:

Implicit indexes are indexes that are automatically created by the database server when an object is created. Indexes are automatically created for primary key constraints and unique constraints.

Example

Following is an example where we will create an index on [COMPANY](#) table for salary column:

```
sqlite> CREATE INDEX salary_index ON COMPANY (salary);
```

Now, let's list down all the indices available on COMPANY table using **.indices** command as follows:

```
sqlite> .indices COMPANY
```

This will produce the following result, where *sqlite_autoindex_COMPANY_1* is an implicit index which got created when table itself was created.

```
salary_index  
sqlite_autoindex_COMPANY_1
```

You can list down all the indexes database wide as follows:

```
sqlite> SELECT * FROM sqlite_master WHERE type = 'index';
```

The DROP INDEX Command:

An index can be dropped using SQLite **DROP** command. Care should be taken when dropping an index because performance may be slowed or improved.

The basic syntax is as follows:

```
DROP INDEX index_name;
```

You can use following statement to delete previously created index:

```
sqlite> DROP INDEX salary_index;
```

When should indexes be avoided?

Although indexes are intended to enhance a database's performance, there are times when they should be avoided. The following guidelines indicate when the use of an index should be reconsidered:

- Indexes should not be used on small tables.
- Tables that have frequent, large batch update or insert operations.
- Indexes should not be used on columns that contain a high number of NULL values.
- Columns that are frequently manipulated should not be indexed.

Loading [MathJax]/jax/output/HTML-CSS/jax.js