Search...
Sign In

MySQL Indexes

Last Updated : 23 Jul, 2025

MySQL **indexes** are designed tools to increase the speed and efficiency of data retrieval operations within a database. Similar to a

Databases SQL MySQL PostgreSQL PL/SQL MongoDB SQL Cheat She Sign In

having to go page after page, **MySQL indexes** let the **database** quickly find and retrieve your required data without scanning the whole table.

You can execute high-performance database queries that render faster and, subsequently, a more **responsive application** with the strategic **implementation of indexes**. Different types of **MySQL indexes**, how they work, and some best practices for using them effectively will be the main topics in this article.

What is a MySQL Index

A **MySQL index** is a data structure that improves the speed of data retrieval operations on a database table. Similar to an index in a book that helps you quickly find specific information without having to read through the entire book, a **MySQL index** allows the database to **locate** and **access rows in a table** much faster than it could **without an index**.

Types of MySQL Indexes

MySQL supports a variety of indexes so that they can perform different functions in the event of different use cases. Understanding these types will help you choose the right index for any of your database needs.

Primary Index

A primary index is automatically created when a primary key is defined on the table. It uniquely identifies each row in the **TABLE**, hence making sure that the values of the primary key are both unique and cannot be null. There can be only one primary index per table.

Unique Index

The unique index normally ensures that all values within a column are unique. Contrary to the **primary index**, a table may contain several **unique indexes**. This type of index comes in handy in scenarios when you want to enforce uniqueness on a column that doesn't form the primary key.

Full-Text Index

This index type is used with columns storing large quantities of text and facilitates very fast recovery of data based on complicated text-based searches.

Full-text indexes can support full-text search. They are implemented for fast locating of words or phrases within the text columns. Full-text indexes usually come with the **MATCH** and **AGAINST clauses** in SQL queries, and find very pertinent applications in applications like search engines.

Composite Index

A **composite index** is an **index** created on more than one column. Performance is aided in queries that filter on more than one column, and it takes the first place in the column used in the query. For instance, suppose you always search by **last_name** and **first_name**; then, a composite index on (**last_name, first_name**) would come in handy.

Spatial Index

Spatial indexes are applied to geographic data types and enable the effective management and querying of spatial data. Indexes like this are particularly useful in applications that involve mapping, location-based services, and geographic information systems.

Creating and Managing Indexes in MySQL

Proper creation and maintenance of indexes are some of the biggest helps when doing **MySQL** database performance optimization. The proper ways of creating, dropping, and viewing indexes, as well as related best practices for their maintenance, will be discussed in this chapter.

Creation of an Index

An index is created by the statement **CREATE INDEX**. Here's a very simple example of how to use this statement:

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

For the composite index, where several columns are involved, the **syntax** looks as follows:

```
CREATE INDEX idx_composite ON table_name (column1, column2);
```

Unique Indexes

To provide that values in a column are unique, you create a unique index:

```
CREATE UNIQUE INDEX idx_unique_column_name ON table_name
(column_name);
```

Full-Text Indexes

In support of a full-text search, you create a full-text index:

```
CREATE FULLTEXT INDEX idx_fulltext_column_name ON table_name
(text_column_name);
```

Spatial Indexes

If you are dealing with geographic data, then you may want to create a spatial index:

```
CREATE SPATIAL INDEX idx_spatial_column_name ON table_name
(spatial_column_name);
```

Drop an Index

To delete an index, you would execute a **DROP INDEX** statement:

```
DROP INDEX idx_column_name ON table_name;
```

Showing All Existing Indexes

To display the indexes for a table you can use the **SHOW INDEX** command:

```
SHOW INDEX FROM table_name;
```

Using the MySQL Indexes with EXPLAIN Statement

The EXPLAIN statement just helps you understand how MySQL is executing your queries and if:

```
EXPLAIN SELECT * FROM table_name WHERE column_name = 'value';
```

Rebuilding Indexes

Indexes tend to get fragmented over time leading to degradation of performance. Rebuild Indexes help to retain the optimal performance:

```
OPTIMIZE TABLE table_name;
```

Conclusion

MySQL indexes are among the most important elements of database performance optimization, particularly in data retrieval operations. The proper knowledge and implementation of the different types of **indexes**, such as **primary**, **unique**, **full-text**, composite, and spatial indexes, will significantly improve the **speed** and **efficiency** of your queries.

Indexes work because a data structure is created that allows **MySQL** to quickly **locate** and **access** the required rows, hence avoiding having to scan **entire tables**. However, it is important to use indexes judiciously since they will also add some overhead to write operations and use disk space.

Comment N nsachi... + Follow 1

Article Tags : Databases MySQL