SQL indexes

MrinalVerma

An index is a schema object. It is used by the server to speed up the retrieval of rows by using a pointer. It can reduce disk I/O(input/output) by using a rapid path access method to locate data quickly. An index helps to speed up select queries and where clauses, but it slows down data input, with the update and the insert statements. Indexes can be created or dropped with no effect on the data. In this article, we will see how to create, delete, and uses the INDEX in the database.

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

Creating an Index:

Syntax:

```
CREATE INDEX index
ON TABLE column;
```

where the **index** is the name given to that index and **TABLE** is the name of the table on which that index is created and **column** is the name of that column for which it is applied.

For multiple columns:

Syntax:

```
CREATE INDEX index
ON TABLE (column1, column2,.....);
```

Unique Indexes: Unique indexes are used for the maintenance of the integrity of the data present in the table as well as for fast performance, it does not allow multiple values to enter into the table.

Syntax:

```
CREATE UNIQUE INDEX index
ON TABLE column;
```

When should indexes be created:

- A column contains a wide range of values.
- A column does not contain a large number of null values.
- One or more columns are frequently used together in a where clause or a join condition.

When should indexes be avoided:

- The table is small
- The columns are not often used as a condition in the query
- The column is updated frequently

Removing an Index: Remove an index from the data dictionary by using the **DROP INDEX** command.

Syntax:

```
DROP INDEX index;
```

To drop an index, you must be the owner of the index or have the **DROP ANY INDEX** privilege.

Altering an Index: To modify an existing table's index by rebuilding, or reorganizing the index.

```
ALTER INDEX IndexName
ON TableName REBUILD;
```

Confirming Indexes: You can check the different indexes present in a particular table given by the user or the server itself and their uniqueness.

Syntax:

```
select * from USER_INDEXES;
```

It will show you all the indexes present in the server, in which you can locate your own tables too.

Renaming an index: You can use the system-stored procedure sp_rename to rename any index in the database.

Syntax:

```
EXEC sp_rename
    index_name,
    new_index_name,
    N'INDEX';
```

Lastly, let us discuss why should we use indexing?

Indexing is an important topic when considering advanced MySQL, although most people know about its definition and usage they don't understand when and where to use it to change the efficiency of our queries or stored procedures by a huge margin.

Here are some scenarios along with their explanation related to Indexing:

1. When executing a query on a table having huge data (> 100000 rows), MySQL performs a full table scan which takes much time and the server usually gets timed out. To avoid this always check the explain option for the query within MySQL which tells us about the state of execution. It shows which columns are being used and whether it will be a threat to huge data. On basis of the columns repeated in a similar order in conditions, we can create an index for them in the same order to maximize the speed of the query.
2. The order of the index is of huge importance as we can use the same index in many scenarios. Using only one index we can utilize it in more than one query which different conditions. like for example, in a query, we make a join with a table based on customer_id wards we also join another join based on customer_id and order_date. Then we can simply create a single index by the order of customer_id, order_date which would be used in both cases. This also saves storage.
3. We should also be careful to not make an index for each query as creating indexes also take storage and when the amount of data is huge it will create a problem. Therefore, it's important to carefully consider which columns to index based on the needs of your application. In general, it's a good practice to only create indexes on columns that are frequently used in queries and to avoid creating indexes on columns that are rarely used. It's also a good idea to periodically review the indexes in your database and remove any that are no longer needed.
4. Indexes can also improve performance when used in conjunction with sorting and grouping operations. For example, if you frequently sort or group data based on a particular column, creating an index on that column can greatly improve performance. The index allows MySQL to quickly access and sort or group the data, rather than having to perform a full table scan.
5. In some cases, MySQL may not use an index even if one exists. This can happen if the query optimizer determines that a full table scan is faster than using the index. This can occur

when the table is small, the query is highly selective, or the table has a high rate of updates.

Last Updated : 10 Mar, 2023

90

Similar Reads

1. Multiple Indexes vs Multi-Column Indexes
2. SQL queries on clustered and non-clustered Indexes
3. Difference between Structured Query Language (SQL) and Transact-SQL (T-SQL)
4. Configure SQL Jobs in SQL Server using T-SQL
5. SQL vs NO SQL vs NEW SQL
6. Secondary Indexes on MAP Collection in Cassandra
7. Secondary Indexes on LIST Collection in Cassandra
8. Secondary Indexes on SET Collection in Cassandra
9. How to Compare Indexes of Tables From Two Different Databases in Oracle?
10. SQL | Procedures in PL/SQL

Previous Next

Article Contributed By :

MrinalVerma
MrinalVerma

Vote for difficulty

Current difficulty : Easy

Easy Normal Medium Hard Expert

Improved By : sunny94, khushboogoyal499, swchoi442, varshachoudhary, adichavan095

Article Tags : DBMS Indexing, DBMS-SQL, SQL-basics, DBMS, SQL

Practice Tags : DBMS, SQL