The index is very valuable in practice and also a popular area of interview questions. The index is responsible for the performance optimization of the database, so the proper use of the index can help the database improve performance substantially and the related user experience.

As such, the index is a very important piece of technology for daily work in the database field.

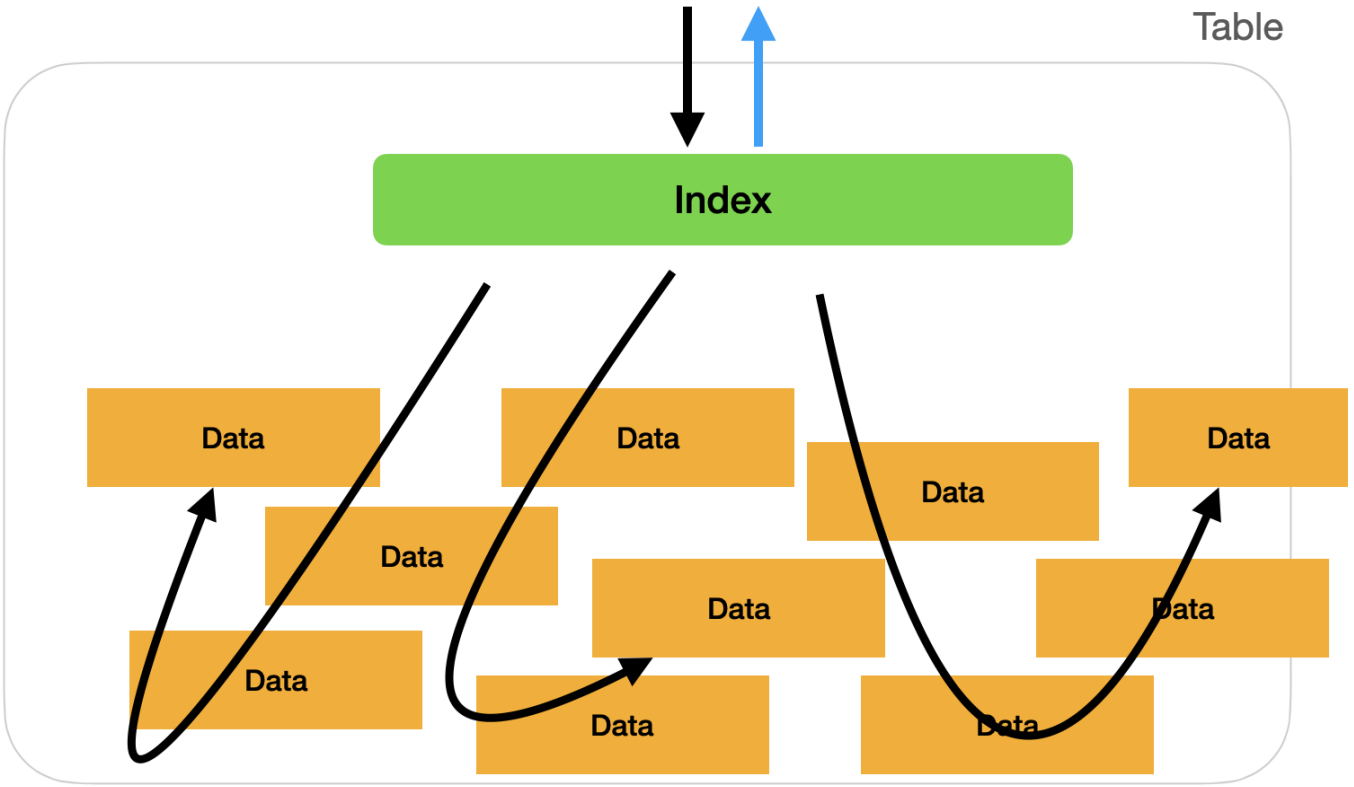
Understand Index

Concept & Pros

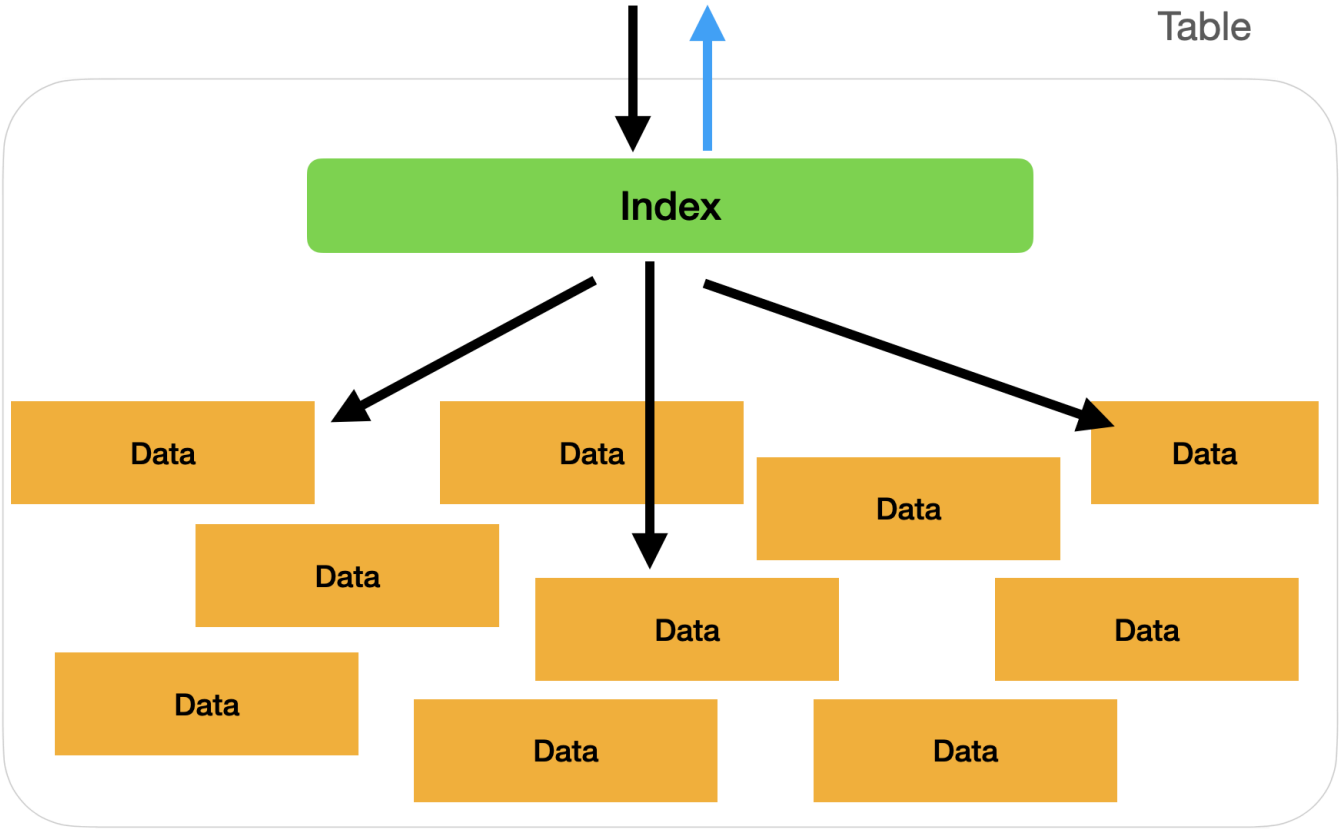
The core concept of index technology is to mark specific columns in the table and extract them to be stored additionally and in an organized manner. When an SQL statement is executed, you can directly search for data based on these indexes.

Assuming no indexes are set, SQL may have to go around the entire table to find the data. The following are two images that demonstrate the differences in query operation with and without index:

Without INDEX: In the index layer, no index is found, so many redundant query paths may be taken to retrieve data.



With index: In the index layer, if an index is found and it has relative information to your query, then you can directly know which data to fetch without taking a detour.



However, the index is not effective in all situations, mainly when data exceeds 100,000 records in a single table, the user will gradually feel the difference in performance. Or, in a transaction or function with a huge number of SQL statements, the difference in performance will again be noticeable to the user.

Cons

Using an index is not a silver bullet. There is one significant defect we should be aware of.

Assuming that our table will have high-frequency data writing actions, it will cause the database to build an index for the table frequently, which requires operation time.

Therefore, assuming a table with such characteristics, it is recommended to make a good evaluation and make a trade-off to avoid the dilemma that the query speed is 1 second faster, but inserting new data is 2 seconds slower.

Syntax

The syntax for adding an index is very simple, similar to the syntax for adding `FOREIGN KEY`:

```
ALTER TABLE `new_schema`.`users`
    ADD INDEX `name_index` (`name`);
```

The first line denotes the table to add the index to, and the second line starts the code of the index. `name_index` is the name of the index, and `name` is the column to be indexed.

The sample code above can also be written in another format with the exact same functionality:

```
CREATE INDEX `name_index` ON `new_schema`.`users` (`name`);
```

When you want to delete an index, the syntax is even simpler, you don't even need to bring a column, just specify the index name with `DROP` keyword:

```
ALTER TABLE `new_schema`.`users`
    DROP INDEX `name_index`;
```

Categories

There are many categories of Index, if we do not specify it, in MySQL, it will be set to the `INDEX` category by default. In addition to `INDEX`, we will introduce the three most commonly used categories.

PRIMARY KEY

We are already familiar with the `PRIMARY KEY` (which we will refer to as PK). In the Chapter 1-2 SQL Data Structure: Table, we learned that it represents the unique key of a table and cannot be repeated. We also reserved a foreshadowing in this chapter to mention that PK is very important for the efficiency of searching for data. The reason will be revealed in this section.

When setting the PK for the database, an index is also created, and `PRIMARY KEY` is used as the index category. The advantage of using this category is that when we use the `JOIN` to combine two tables together, it will effectively increase the processing speed.

You may have noticed that when we are concatenating, it is usually id to id. At this time, the index of PK will take effect, which brings extremely high speed. Therefore, when using `JOIN`, try to connect with PK to ensure the execution speed!

UNIQUE INDEX

Using `UNIQUE INDEX` is similar to the column unique setting we learned in the past, and it is often created together. After using `UNIQUE INDEX`, a constraint will be created to limit the column to only store unique values, avoid duplication, and improve the efficiency of index usage.

Also, please note that `PRIMARY KEY` is one category of `UNIQUE INDEX`.

FULL TEXT

`FULL TEXT` is an index specially designed for keyword search. If you want to do a keyword search engine related application, you will have the opportunity to use the full text index.

Especially, not all languages support full text indexes, and it is necessary to check the supported languages of each different database software.

Six Principles About Index Best Pratices

When using an index, we recommend mastering these six principles so that the index can play a high-quality role in our code. 1. Try to choose `UNIQUE INDEX`, otherwise, use `KEY`. 2. Add an index for the column that is often referenced by complex operations, such as `GROUP BY`. 3. Index the hotspot column that is frequently queried in `WHERE`, but make sure that the differences in the column values are large enough. For example, adding an index to the `identify_number` column is better than adding an index to `country_of_birth`. 4. The index actually uses the disk space, so pay attention to the space used in the hard disk. 5. Index the column with a small value. For example, the index `title` of the article is better than the index `content` of the article 6. Try to avoid index on columns with NULL, which may affect the query efficiency.

The above is the syntax of the index and the 6 basic principles that need to be mastered. I hope it can help you use the index more correctly!