A blue background with white text and numbers

Description automatically generated

In the world of databases, efficiency is key when it comes to retrieving data quickly and accurately. Just like a book index helps us find specific chapters without flipping through every page, an index in SQL serves a similar purpose. By creating an index on a database table, we can significantly enhance the speed and efficiency of data retrieval operations. In this article, we will explore the concept of an index in SQL, its benefits, and how it can revolutionize the way we access and manage large amounts of data.

**What is index?**

index is a **database object** that improves the **speed and efficiency of data retrieval operations**. It is created on one or more columns of a table and acts as a reference or pointer to the actual data in the table.

When an index is created on a column, the database system creates a separate data structure that organizes the values in that column. This data structure allows the database engine to **locate specific rows in the table** more quickly, without having to **scan the entire table**.

**Why are indexes needed?**

Imagine walking into the Library of Congress and being given the task to find a specific publishing within 10 minutes. Would you be able to complete this task within the given time frame? The Library of Congress is considered the largest library in the world and it houses approximately [170 million items](https://www.loc.gov/about/fascinating-facts/#:~:text=The%20Library%20of%20Congress%20is,more%20than%20170%20million%20items.). Now, the Library of Congress is not a regular library where the public can check out books at will, but if you are like us, you know the challenge should not be too difficult. In fact, the first thing we would do is ask for access to the library’s index because indexes contain all the necessary information needed to access items quickly and efficiently.

In the same manner, a database index contains all the necessary information to access data quickly and efficiently. In today’s society, the business of data is rapidly advancing. In fact, some tech giants process several hundred petabytes (1000⁵ bytes) of data per day. Storing all of this data in a database is great, but for a data company, being able to access that data efficiently is paramount to success. Just like the Library of Congress example, one way of solving the access issue when it comes to large amounts of data is through the use of indexes. Indexes serve as lookup tables that efficiently store data for quicker retrieval.

**How are indexes created?**

**Example of index:**

At the moment, the Employees table, does not have an index on SALARY column.

A table with numbers and letters

Description automatically generated

Employees table



Query

To find all the employees, who has salary greater than 5000 and less than 7000, the query engine has to check each and every row in the table, resulting in **a table scan**, which can adversely affect the performance, especially if the table is large. Since there is no index, to help the query, **the query engine performs an entire table scan.**

**Create index:**



Create index

The index stores salary of each employee, in the ascending order as shown below. The actual index may look slightly different

A table with yellow and black text

Description automatically generated

Employee Table And Index Table

Now, when the SQL server has to execute the same query, it has an index on the salary column to help this query. Salaries between the range of 5000 and 7000 are usually present at the bottom, since the salaries are arranged in an ascending order. SQL server picks up the row addresses from the index and directly fetch the records from the table, rather than scanning each row in the table. This is called as Index Seek.

**Exploring Index Types in SQL?**

In SQL databases, indexes play a crucial role in optimizing query performance by facilitating rapid data retrieval operations. Various types of indexes are utilized, each offering unique benefits and functionality to suit different database requirements.

1. Clustered Index
2. Non-clustered Index
3. Unique Index
4. Composite Index
5. Filtered Index
6. Spatial Index
7. Full-text Index
8. Covering Index
9. Bitmap Index
10. Hash Index

In this article, we'll delve specifically into the nuances of clustered and non-clustered indexes, exploring their roles in optimizing database performance. Stay tuned for insights tailored to these essential indexing techniques!

* **clustered index**

A clustered index determines the physical order of data in a table. For this reason, a table can have only one clustered index.

A screenshot of a computer code

Description automatically generated

Creating Table tblEmployee

**Note** that, Id column is marked as primary key. Primary key, constraint create clustered indexes automatically if no clustered index already exists on the table.

to confirm: **Execute sp\_helpindex tblEmployee**



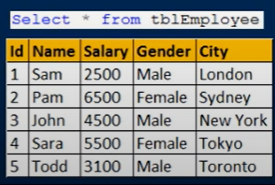
clustered index automatic created

**note that, the values for id column are not in a sequential order.**

A white background with red and blue text

Description automatically generated

inserting rows



Query 1

**Clustered Index**

A clustered index is analogous to a telephone directory, where the data is arranged by the last name. We just learnt that, a table can have only one clustered index. However, the index can contain multiple columns (a composite index), like the way a telephone directory is organized by last name and first name.

**Create a composite lustered index on the Gender and Salary columns:**



Create Clustered index



Query 2

**In comparing the results of Query 1, ordered by ID, and Query 2, ordered by gender in descending order and salary in ascending order, it's apparent that the data is stored in specific orders dictated by the respective clustered indexes.**

**NonClustered Index**



Create NonClustered Index

A table with different cities

Description automatically generated

Data Table || Index Table

A **nonclustered** index is analogous to an index in a textbook The data is stored in one place, the index in another place. The index will have pointers to the storage location of the data.

Since, the **nonclustered** index is stored separately from the actual data, a table can have more than one non clustered index, just like how a book can have an index by Chapters at the beginning and another index by common terms at the end..

In the index itself, the data is stored in an ascending or descending order of the index key, which doesn't in any way influence the storage of data in the table.

Difference Between **Clustered** and **NonClustered** Index

**1. Only one clustered index per table**, where as you can have more than one non clustered index

**2.** **Clustered index is faster** than a non clustered index, because, the clustered index has to refer back to the table, if the selected column is not present in the index.

**3.** **Clustered index determines the storage order** of rows in the table, and hence doesn't require additional disk space, but where as a Non Clustered index is stored separately from the table, additional storage space is required.

**Essential Best Practices for SQL Indexing**

Indexing in SQL involves several best practices to ensure optimal performance. Here are some key recommendations:

1. **Identify High-Use Queries**: Analyze your most frequently executed queries and identify the columns involved in the WHERE, JOIN, ORDER BY, and GROUP BY clauses. These columns are prime candidates for indexing.
2. **Choose Index Columns Wisely**: Select columns that are frequently used for filtering, sorting, or joining data. Consider columns with high selectivity (a wide range of values) to ensure efficient data retrieval.
3. **Avoid Over-indexing**: While indexes can improve read performance, they come with overhead during data modification operations (INSERT, UPDATE, DELETE). Avoid creating indexes on columns that are rarely queried or have low selectivity.
4. **Use Composite Indexes**: For queries involving multiple columns, consider creating composite indexes covering these columns. Composite indexes can improve query performance by allowing the database engine to satisfy queries directly from the index.
5. **Regularly Review and Maintain Indexes**: Periodically review your indexes to ensure they remain relevant and beneficial. Remove unused or redundant indexes, and consider rebuilding or reorganizing fragmented indexes to maintain optimal performance.
6. **Consider Index Fragmentation**: Monitor index fragmentation levels and take appropriate actions to address fragmentation, such as rebuilding or reorganizing indexes.
7. **Understand Index Types**: Understand the differences between clustered and non-clustered indexes and choose the appropriate type based on your query patterns and data access requirements.
8. **Test Performance Impact**: Before implementing indexes in a production environment, thoroughly test their impact on query performance and overall database performance.

By following these best practices, you can effectively utilize indexing to improve query performance and optimize database operations in SQL.

**Conclusion**

In conclusion, implementing effective indexing strategies is crucial for optimizing query performance and enhancing the overall efficiency of SQL databases. By carefully selecting and maintaining indexes based on query patterns, monitoring index usage, and periodically reviewing and refining indexing strategies, database administrators can ensure that their databases perform optimally even as data volumes grow and query complexity increases. By adhering to best practices in SQL indexing, organizations can unlock the full potential of their database systems, resulting in improved application responsiveness, better user experience, and ultimately, enhanced business outcomes.