SQL INDEXES

By : Katam Bhavya

Introduction to SQL Indexing

Definition:

- An index in SQL is a data structure that improves the speed of data retrieval operations on a database table.

Analogy:

- Think of it like the index of a book—making specific information quick to locate.

Why Use Indexes?

Key Benefits:

- Faster Data Retrieval: Speeds up query processing.
- Reduces Disk I/O Operations: Minimizes the amount of data read from disk.
- Efficient Query Performance: Particularly beneficial for large datasets.

Examples of Use Cases:

- Searching specific rows in large tables.
- Supporting unique constraints (e.g., primary keys).
- Optimizing sorting and grouping operations in queries.

How to Create an Index

Syntax:

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

Example:

```
CREATE INDEX idx_lastname ON employees(last_name);
```

Naming Tip: Use meaningful names for indexes that reflect their associated columns, e.g., idx_lastname for the last_name column.

Performance Note: While indexes boost retrieval speed, they can add storage overhead and may impact write operations. Use them judiciously.

How to Manage Indexes

View Indexes:

- To view the indexes on a table, use the following query:

```
SELECT * FROM sys.indexes WHERE object_id = OBJECT_ID('table_name');
```

Drop Indexes:

- To remove an index from a table:

```
DROP INDEX index_name ON table_name;
```

Rebuild an Index:

- To rebuild an existing index (helps with fragmentation):

```
ALTER INDEX index_name ON table_name REBUILD;
```

Types of Indexes

- **Clustered Index**
  - Stores data rows in sorted order based on the index key.
  - Only one clustered index can exist per table, as it determines the physical order of data.
  - **Example:** Primary key column.

- **Non-Clustered Index**
  - Creates a separate structure from the data rows, storing pointers to the actual data.
  - Multiple non-clustered indexes can exist per table.
  - **Example:** Frequently searched columns.

- **Unique Index**
  - Ensures that all values in a column are unique.
  - Can be created on any column, not just primary keys.
  - **Example:** Email or username columns.

- **Composite Index**
  - Index on multiple columns, improving performance for queries filtering on those columns.
  - **Example:** (first_name, last_name).

- **Full-Text Index**
  - Optimized for advanced text-based queries, such as searching for keywords within long text columns.
  - **Example:** Searching long text columns for keywords.

- **Bitmap Index (specific to certain databases)**
  - Efficient for columns with low cardinality (few distinct values).
  - **Example:** Gender or status fields (e.g., "active" vs. "inactive").

Best Practices

- **Index only the necessary columns:** Focus on columns that are frequently queried, involved in joins, or used for sorting and filtering.

- **Avoid excessive indexing**: Too many indexes can slow down insert, update, and delete operations due to the overhead of maintaining them.

- **Periodically analyze and optimize indexes**: Regularly check index performance to ensure they are still beneficial as your data and queries evolve.

- **Combine indexes with query optimization**: Use indexes alongside other optimization techniques like query rewriting or proper join strategies for the best performance.

Drawbacks of Indexing

- **Increased storage usage**: Indexes require additional storage space, which can grow with large tables.

- **Slower write operations**: Insert, update, and delete operations are slower due to the need to update the index every time data changes.

- **Requires careful planning and monitoring**: Effective indexing requires ongoing assessment and optimization to avoid unnecessary performance overhead.

Conclusion

Key Takeaway:

- SQL indexing is a powerful tool for optimizing database performance, especially for query-heavy operations.

Advice:

- Always balance the use of indexes with potential trade-offs like storage and maintenance overhead.
- Regularly analyze query performance to determine where indexing adds value.

THANK YOU