**SQL Indexes**

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](https://www.geeksforgeeks.org/sql-create-table/), [delete](https://www.geeksforgeeks.org/sql-delete-statement/), and use the INDEX in the database.

**Creating an Index**

**Syntax**

***CREATE INDEX****index*

***ON TABLE****column;*

where the **index** is the name given to that index **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,…..);*

**For 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**](https://www.geeksforgeeks.org/sql-drop-truncate/)**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’;*

**SQL Server Database**

**Syntax:**

***DROP INDEX****TableName.IndexName;*

**Why SQL Indexing is Important?**

Indexing is an important topic when considering advanced [MySQL](https://www.geeksforgeeks.org/mysql-common-mysql-queries/), 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:

* 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 condition.
* The order of the index is of huge importance as we can use the saitions, we can create an index for them in the same order to maximize the speed of the query. me 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.
* 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.
* 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.
* 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.

**Conclusion**

* Data retrieval is speed up by the usage of indexes.
* They function as a [database](https://www.geeksforgeeks.org/what-is-database/) row’s table of contents.
* Enhance query speed, however, inserts and updates might take longer.
* Bitmap, Hash, and B-tree indexes are examples of typical types.
* Indexes must be carefully selected and kept up-to-date in order for database operations to go smoothly.

**Types of SQL Indexes: Key to Faster Data Retrieval and Management**

*Take a deep dive into the world of SQL indexes with our comprehensive guide. This guide will help you understand how indexes such as Primary Key, Unique, Clustered, Non-Clustered, and Full-Text can optimize database performance and query efficiency. You will also discover the ideal scenarios for each index type and understand their impact on database operations.*

SQL indexes are essential for efficient database management, enabling quick data access and query execution. This article clarifies the types of indexes in SQL, from the commonly used Primary Key and Unique indexes to specific Clustered and Non-Clustered types. Understanding these indexes is crucial for optimal database performance.

**Table of Content**

* [What are Indexes in SQL?](https://www.shiksha.com/online-courses/articles/types-of-index-in-sql-blogId-148379#1)
* [Why Indexes in SQL are Used?](https://www.shiksha.com/online-courses/articles/types-of-index-in-sql-blogId-148379#2)
* [Types of Indexes](https://www.shiksha.com/online-courses/articles/types-of-index-in-sql-blogId-148379#3)
  + [Primary Key Index](https://www.shiksha.com/online-courses/articles/types-of-index-in-sql-blogId-148379#4)
  + [Unique Index](https://www.shiksha.com/online-courses/articles/types-of-index-in-sql-blogId-148379#5)
  + [Clustered Index](https://www.shiksha.com/online-courses/articles/types-of-index-in-sql-blogId-148379#6)
  + [Non-Clustered Index](https://www.shiksha.com/online-courses/articles/types-of-index-in-sql-blogId-148379#7)
* [SQL Indexes - Types, Uses, Advantages, Disadvantages, and Scenarios](https://www.shiksha.com/online-courses/articles/types-of-index-in-sql-blogId-148379#8)

**What are Indexes in SQL?**

Indexes in SQL are specialized lookup tables that are used by the database search engine to accelerate data retrieval.

In simple terms, an index in SQL is a tool used to quickly identify rows with specific column values. If there were no indexes, the SQL server would have to start with the first row and then go through the entire table until it discovers the relevant rows. This method is known as a full-table scan and can be highly inefficient for large tables.

**Stay updated with the latest blogs on online courses and skills**

Enter Mobile Number

Register Now

**Why Indexes in SQL are Used?**

* **Improved Query Performance**: The primary reason for using indexes is to accelerate query processing. Indexes can drastically reduce the amount of data the server needs to examine.
* **Efficient Data Access**: Indexes provide a quick way to access row data for SELECT statements. This is particularly beneficial for tables with a large number of rows.
* **Sorting and Grouping Speed**: Indexes improve the speed of data retrieval operations by providing a sorted version of the data, which is faster to process for ORDER BY and GROUP BY operations.
* **Unique Constraints**: Indexes can be used to enforce uniqueness for columns to ensure that no two rows of a table have duplicate values in a particular column or a combination of columns.
* **Optimized Join Operations**: In databases with multiple tables, indexes improve the speed of join operations by quickly locating the joining rows in each table.

**Note:**

Apart from these advantages of Indexes in SQL, they have some limitations too, like:

* **Overuse of Indexes**: While indexes speed up data retrieval, they can slow down data input, through INSERT, UPDATE, and DELETE statements. Each index needs to be updated when data is modified.
* **Storage Space**: Indexes consume additional disk space.
* **Maintenance Overhead**: Indexes need to be maintained and rebuilt over time, which can add overhead to database maintenance routines.

**Types of Indexes**

**Primary Key Index**

A primary key is a field or a combination of fields in a database table that uniquely identifies each record (row) in that table. A primary key index is an automatically generated index associated with the primary key column(s) to enhance data retrieval and enforce data uniqueness.

**Importance of Primary Key Index**

* **Data Uniqueness:** The primary key index enforces the uniqueness constraint on the designated column(s).
  + i.e., no two records in the table can have the same values in the primary key column(s). It prevents duplicate records, ensuring data accuracy.
* **Data Retrieval Efficiency:** By creating a primary key index, the database management system (DBMS) generates a data structure that allows for rapid data retrieval. Instead of scanning the entire table, the DBMS can use the primary key index to pinpoint the exact location of a specific record, significantly improving query performance.
* **Join Operations:** Primary keys are often used in join operations, where data from multiple tables is combined. The primary key index ensures quick and efficient matching of records during these operations, reducing processing time.

**Use Cases**

* **Identification:** Primary keys are commonly used to identify records in a table uniquely.
  + For example, in an "Employees" table, the employee ID might serve as the primary key, allowing each employee to uniquely identify by their ID.
* **Relationships:** Primary keys are essential when establishing relationships between tables in a relational database. They serve as foreign keys in related tables, ensuring referential integrity.
* **Data Integrity:** Primary keys guarantee data integrity by preventing the insertion of duplicate records, ensuring that each record is unique.

**Unique Key Index**

A unique index in a relational database is a data structure that enforces the uniqueness constraint on one or more columns within a table. Its primary purpose is to ensure that values stored in the indexed column(s) are unique across all records in the table.

**Role in Maintaining Unique Values:**

* A unique index serves as a safeguard against duplicate data entries. It ensures that the data integrity of a table is maintained by preventing the insertion of rows with duplicate values in the indexed column(s).
* When a unique index is created on a column, the database management system (DBMS) automatically checks for duplicate values whenever a new record is inserted or an existing record is updated in the table.
* If an insertion or update operation would result in a duplicate value in the indexed column(s), the DBMS raises an error, and the operation is rejected, thereby preventing the introduction of duplicate data.

**Must Read:** [Subqueries in SQL](https://www.shiksha.com/online-courses/articles/subqueries-in-sql/)

**Must Read:** [SQL CREATE TABLE](https://www.shiksha.com/online-courses/articles/how-to-create-table-in-sql/)

**Difference Between Primary Key Indexes and Unique Index**

|  |  |  |
| --- | --- | --- |
| **Index Attribute** | **Primary Key** | **Unique Index** |
| **Uniqueness Constraint** | A primary key enforces uniqueness and serves as the primary identifier. It must contain non-null values and uniquely identify each row. | A unique index enforces uniqueness but does not require serving as the primary identifier. Null values are allowed as long as non-null values are unique. |
| **Number of Columns** | There can be only one primary key per table, consisting of one or more columns. | Multiple unique indexes can be created within a single table, each enforcing uniqueness on different sets of columns. |
| **Use in Relationships** | Primary keys are often used as foreign keys in related tables to establish relationships. | Unique indexes can also be used in relationships but do not have the same semantics as primary keys. They are typically used when uniqueness is needed without the requirement of being a primary identifier. |

**Use Cases**

* **Email Addresses:** In a user database, using a unique index on the email address column ensures that each user has a unique email, preventing multiple accounts with the same email.
* **Identification Numbers:** When storing identification numbers like social security or passport numbers, a unique index ensures that no two individuals share the same identifier within the database.
* **Product SKUs:** Unique indexes can be applied to product SKU (Stock Keeping Unit) columns to prevent duplicate SKUs in an inventory database.
* **Membership IDs:** In a membership system, unique indexes on membership IDs guarantee that each member has a distinct identification number.
* **Invoice Numbers:** In financial systems, unique indexes on invoice numbers ensure that each invoice is uniquely identified, avoiding billing errors.

**Clustered Index**

A clustered index sorts and stores the rows of a table based on the values in one or more specified columns. Each table can have only one clustered index, and the choice of the clustering column(s) significantly impacts how data is stored and retrieved.

**Importance of Clustered Index**

* **Physical Data Organization:** The primary purpose of a clustered index is to physically order the data rows in the table based on the values in the indexed column(s). This arrangement allows for efficient data retrieval when queries request data in the same order as the clustered index.
* **Optimized Data Retrieval:** Clustered indexes are particularly useful for improving query performance when selecting, sorting, or filtering data based on the columns included in the clustered index. They eliminate the need for a separate data lookup process, as the data rows are already stored in the desired order.
* **Sequential Access:** When queries involve range scans or retrieving a range of data values, a clustered index is highly efficient. It allows for sequential access, reducing disk I/O operations and enhancing query speed.

**Use Cases of Clustered Index**

* **Primary Key:** A common use of a clustered index is to define it on the primary key column(s). This ensures that the table's data is physically ordered according to the primary key values, facilitating fast retrieval of specific records.
* **Date and Time Data:** In tables where date and time information is critical, a clustered index on a timestamp column allows for efficient retrieval of data based on chronological order.
* **Sequential Data:** For tables that store sequentially generated data, such as transaction logs or sequential invoice numbers, a clustered index can optimize the retrieval of data in chronological or sequential order.

**Non-Clustered Index**

A non-clustered index is a type of index used in relational databases to improve the efficiency of data retrieval operations. Unlike clustered indexes, which affect the physical order of data rows within a table, non-clustered indexes create separate data structures to allow fast access to specific data subsets. This means that non-clustered indexes do not rearrange the physical organization of data, but rather create a separate structure to facilitate quicker access to the data.

**Importance of Non-Clustered Index**

* **Faster Data Retrieval:** Non-clustered indexes significantly improve query performance by allowing the database management system (DBMS) to quickly locate and retrieve specific data rows based on the indexed column(s).
* **Reduced Disk I/O:** Non-clustered indexes reduce the need for full table scans when querying data. This leads to fewer disk input/output (I/O) operations, resulting in faster query execution.
* **Support for Multiple Indexes:** Unlike clustered indexes, which limit a table to one, non-clustered indexes can be created on multiple columns, enabling efficient retrieval for various query patterns.

**Must Read:** [SQL LIMITS](https://www.shiksha.com/online-courses/articles/sql-limit/)

**Must Check:** [SQL Online Course and Certifications](https://www.shiksha.com/online-courses/sql-certification)

**Difference Between Clustered and Non-Clustered Index**

|  |  |  |
| --- | --- | --- |
| **Index Type** | **Non-Clustered Index** | **Clustered Index** |
| **Physical Order** | Does not determine the physical order of data rows. | Determines the physical order of data rows based on indexed column(s). |
| **Data Structure** | Creates a separate data structure containing index entries with pointers to the corresponding data rows. | Organizes the actual data rows in the table in the specified order. |
| **Multiple Indexes** | Allows for multiple non-clustered indexes on a single table. | Restricts a table to having only one clustered index. |
| **Query Optimization** | Suitable for optimizing query performance when the query does not align with the physical data order. | Ideal for queries that frequently retrieve data in the same order as the clustering column(s). |

**SQL Indexes - Types, Uses, Advantages, Disadvantages, and Scenarios**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Index Type** | **Use** | **Advantages** | **Disadvantages** | **Ideal Scenarios** |
| **Primary Key Index** | Automatically created with the primary key to enforce uniqueness | Fast data retrieval; Ensures data integrity | Additional storage; Slower insert/update operations | Unique identifier for each row (e.g., UserID) |
| **Unique Index** | Enforces uniqueness on a column not part of the primary key | Prevents duplicate values; Improves search performance | Slower write operations; Additional storage requirement | Columns requiring unique data but not suitable as a primary key (e.g., Email) |
| **Clustered Index** | Determines the physical storage order of data in the table | Fast data retrieval for range queries; Efficient use of disk space | Only one per table; Update operations can be slow due to reordering | Columns frequently used in sorting and range queries (e.g., Dates) |
| **Non-Clustered Index** | Provides a separate structure from the data rows and includes a pointer | Faster access than table scan; Multiple non-clustered indexes allowed per table | Increased storage; Slower write operations due to index updates | Frequently searched fields not in clustered index (e.g., FirstName) |
| **Composite Index** | Index on two or more columns | Improves performance on queries involving multiple columns | More complex, Increased storage; Slower writes | Multi-column searches and sorting (e.g., FirstName, LastName) |
| **Full-Text Index** | Used for full-text searches in text data | Facilitates complex queries on text data; Faster than LIKE searches | Takes up significant storage space; Specialized use-case | Large text fields (e.g., Product Descriptions, Articles) |
| **Bitmap Index** | Efficient for columns with a low cardinality (few unique values) | Small storage space; Fast for read-intensive tasks | Not suitable for frequently changing data; Performance issues with high cardinality data | Columns with limited unique entries (e.g., Gender, Marital Status) |
| **Spatial Index** | For indexing spatial data types | Improves performance for queries involving spatial data | Specific use-case; Additional complexity | Geographical data, location-based queries (e.g., Maps, Regions) |

**Conclusion**

Understanding SQL indexes is crucial for database management and optimization. Each type of index serves a specific purpose and enhances database performance. As technology evolves, SQL indexing advances, promising improvements in data storage and retrieval. Staying informed about these developments is essential for maintaining efficient databases in a growing digital world.