**What is an Index in SQL?**

An **index** in SQL is a database object that improves the speed of data retrieval operations on a table at the cost of additional storage and slower write operations (INSERT, UPDATE, DELETE). It is similar to the index of a book, allowing you to quickly locate specific data without scanning the entire table.

**Why Do We Need Indexes?**

1. **Improves Query Performance:** Reduces the number of rows scanned by the database during a query.
2. **Efficient Searching:** Helps locate rows matching query conditions faster.
3. **Optimizes Sorting:** Speeds up ORDER BY and GROUP BY operations.
4. **Enhances JOIN Operations:** Speeds up table joins by quickly finding matching rows.

**Creating Indexes**

**Single-Column Index**

-- Create an index on a single column

CREATE INDEX idx\_customer\_name ON Customers(CustomerName);

**Multi-Column (Composite) Index**

-- Create an index on multiple columns

CREATE INDEX idx\_customer\_location ON Customers(City, Country);

**Example Scenarios**

**Without Index**

SELECT \* FROM Customers WHERE CustomerName = 'John';

* The query scans the entire table, which can be slow if the table is large.

**With Single-Column Index**

CREATE INDEX idx\_customer\_name ON Customers(CustomerName);

SELECT \* FROM Customers WHERE CustomerName = 'John';

The database uses the index to locate "John" quickly, skipping irrelevant rows.

**With Multi-Column Index**

CREATE INDEX idx\_customer\_city\_country ON Customers(City, Country);

SELECT \* FROM Customers WHERE City = 'New York' AND Country = 'USA';

The index helps to quickly locate rows that match both City and Country.

**Advantages of Multi-Column Index**

* Efficient for queries involving multiple columns.
* Reduces the need for multiple single-column indexes.

**types of indexes in SQL** with examples for each:

**1. Clustered Index**

* **Definition:** A clustered index sorts and stores the data rows in the table based on the key values. The table itself is physically organized based on the clustered index.
* **Key Points:**
  + Each table can have only one clustered index.
  + Improves query performance for range-based searches.

**Example**

-- Creating a clustered index on the 'CustomerID' column

CREATE CLUSTERED INDEX idx\_customer\_id ON Customers(CustomerID);

-- Query that benefits from the clustered index

SELECT \* FROM Customers WHERE CustomerID BETWEEN 100 AND 200;

**2. Non-Clustered Index**

* **Definition:** A non-clustered index contains pointers to the data rows instead of physically rearranging them.
* **Key Points:**
  + A table can have multiple non-clustered indexes.
  + Best for exact match queries or when multiple indexes are needed.

**Example**

-- Creating a non-clustered index on the 'CustomerName' column

CREATE NONCLUSTERED INDEX idx\_customer\_name ON Customers(CustomerName);

-- Query that benefits from the non-clustered index

SELECT \* FROM Customers WHERE CustomerName = 'John';

**When to Use Non-Clustered Index**

* Frequently queried columns that are not part of the clustered index.
* Columns used in WHERE, GROUP BY, or ORDER BY clauses.
* Columns used for joins in complex queries.

### ****Example with Non-Clustered Index****

#### **1. Table Creation**

-- Create a sample 'Customers' table

CREATE TABLE Customers (

CustomerID INT PRIMARY KEY,

FirstName NVARCHAR(50),

LastName NVARCHAR(50),

Email NVARCHAR(100),

City NVARCHAR(50)

);

#### **2. Create a Non-Clustered Index**

Suppose we often query customers by their City. Creating a non-clustered index on the City column will optimize these queries.

-- Create a non-clustered index on 'City'

CREATE NONCLUSTERED INDEX idx\_city ON Customers(City);

#### **3. Query Optimization**

Now, when we run a query to fetch all customers from a specific city, the database uses the non-clustered index.

-- Query that benefits from the non-clustered index

SELECT \* FROM Customers WHERE City = 'New York';

Without the index, the database would need to scan the entire Customers table. With the index, it can directly jump to the rows related to 'New York'.

**Non-Clustered Index vs Clustered Index**

|  |  |  |
| --- | --- | --- |
| Feature | Non-Clustered Index | Clustered Index |
| Data Storage | Stored separately from table data. | Physically reorganizes table data. |
| Row Locator | Contains a pointer to the data row. | Points directly to the data (no separate pointer). |
| Number of Indexes | Multiple non-clustered indexes allowed. | Only one clustered index per table. |
| Query Type | Best for queries on specific columns. | Best for range-based queries. |
| Physical Order | Does not change the physical order of data. | Changes the physical order of data. |

**Advantages of Non-Clustered Index**

1. **Flexible Queries:**
   * Works well for queries involving specific columns that are not part of the clustered index.
2. **Multiple Indexes:**
   * You can create multiple non-clustered indexes on different columns in a table.
3. **Less Storage Impact:**
   * Since the data isn't physically rearranged, the table structure remains unaffected.

**Disadvantages of Non-Clustered Index**

1. **Slower for Large Range Queries:**
   * For range-based queries, a clustered index is more efficient.
2. **More Storage Needed:**
   * Non-clustered indexes require additional storage for the index structure and row locators.
3. **Maintenance Overhead:**
   * When data is updated, inserted, or deleted, maintaining multiple non-clustered indexes can slow down write operations.

**Real-Life Use Case**

**Scenario: E-commerce Product Search**

In an e-commerce database, products are stored in a Products table. Users frequently search products by Category and Brand.

* **Create Non-Clustered Indexes:**

CREATE NONCLUSTERED INDEX idx\_category ON Products(Category);

CREATE NONCLUSTERED INDEX idx\_brand ON Products(Brand);

* **Query Optimization:**

-- Query that benefits from non-clustered indexes

SELECT ProductName, Price FROM Products WHERE Category = 'Electronics' AND Brand = 'Samsung';

The non-clustered index ensures that the database quickly retrieves rows matching 'Electronics' and 'Samsung' without scanning the entire table.

**3. Unique Index**

* **Definition:** Ensures that all values in the indexed column(s) are unique.
* **Key Points:**
  + Prevents duplicate values in the column.
  + Can be clustered or non-clustered.

**Example**

-- Creating a unique index on the 'Email' column

CREATE UNIQUE INDEX idx\_unique\_email ON Customers(Email);

-- Inserting duplicate emails will result in an error

INSERT INTO Customers (CustomerID, Email) VALUES (1, 'john.doe@example.com');

INSERT INTO Customers (CustomerID, Email) VALUES (2, 'john.doe@example.com'); -- Error

**4. Composite Index (Multi-Column Index)**

* **Definition:** An index that includes multiple columns.
* **Key Points:**
  + Useful for queries that filter or sort based on multiple columns.
  + The order of columns in the index is important.

**Example**

-- Creating a composite index on 'City' and 'Country'

CREATE INDEX idx\_city\_country ON Customers(City, Country);

-- Query that benefits from the composite index

SELECT \* FROM Customers WHERE City = 'New York' AND Country = 'USA';

**5. Full-Text Index**

* **Definition:** Used to perform full-text searches on large text columns, like finding words or phrases in text data.
* **Key Points:**
  + Commonly used in document or blog systems.

**Example**

-- Creating a full-text index on the 'Description' column

CREATE FULLTEXT INDEX ON Products(Description);

-- Query using full-text search

SELECT \* FROM Products WHERE CONTAINS(Description, 'Laptop');

**6. Spatial Index**

* **Definition:** Used for indexing spatial data like geometry or geography data types.
* **Key Points:**
  + Useful for geospatial queries (e.g., finding points within a radius).

**Example**

-- Creating a spatial index on a 'Location' column

CREATE SPATIAL INDEX idx\_location ON Locations(GeoCoordinates);

-- Query using spatial data

SELECT \* FROM Locations WHERE ST\_Within(GeoCoordinates, @searchArea);

**7. Bitmap Index (Specific to some databases)**

* **Definition:** A special type of index that maps values to bitmaps and is efficient for low-cardinality columns (columns with few unique values).
* **Key Points:**
  + Often used in data warehousing.

**Example**

-- Creating a bitmap index (Oracle database example)

CREATE BITMAP INDEX idx\_gender ON Customers(Gender);

-- Query benefits from bitmap index

SELECT \* FROM Customers WHERE Gender = 'Male';

**Key Differences Between Index Types**

|  |  |  |  |
| --- | --- | --- | --- |
| Type | Supports Multiple Indexes | Physical Row Order Affected | Best For |
| Clustered Index | No | Yes | Range queries, primary keys |
| Non-Clustered Index | Yes | No | Exact matches, multiple conditions |
| Unique Index | Yes | Depends | Enforcing uniqueness |
| Composite Index | Yes | Depends | Multi-column queries |
| Full-Text Index | Yes | No | Text search queries |
| Spatial Index | Yes | No | Geospatial queries |
| Bitmap Index | Yes | No | Low-cardinality data (e.g., Boolean) |

**Indexing Best Practices**

1. **Index Columns Used in WHERE, JOIN, and ORDER BY Clauses:**
   * Helps queries with these operations.
2. **Avoid Excessive Indexing:**
   * Indexes consume storage and slow down write operations.
3. **Analyze Query Performance:**
   * Use tools like EXPLAIN or QUERY PLAN to see if indexes are being utilized.( MySQL)

SQL SERVER

SET STATISTICS PROFILE ON;

SELECT \* FROM products WHERE description = 'smartphone';

SET STATISTICS PROFILE OFF;

1. **Use Composite Indexes Judiciously:**
   * Useful only when the columns are queried together frequently.
2. **Drop Unused Indexes:**
   * Remove indexes that are not improving performance to save storage.
3. **Viewing Existing Indexes**

-- For MySQL

SHOW INDEX FROM Customers;

-- For SQL Server (Currently Jolly is using SQLSERVER MANAGEMENT STUDIO)

EXEC sp\_helpindex 'Customers';

-- For PostgreSQL

SELECT indexname, indexdef FROM pg\_indexes WHERE tablename = 'customers';

Indexes are widely used in real-world database systems to improve the efficiency of various operations. Below are some **real-time use cases of indexes** and their significance:

**1. E-commerce Applications**

* **Scenario:** Searching for products by name or category.
* **How Index Helps:**

A non-clustered index on the ProductName and Category columns speeds up product searches.

Example:

CREATE NONCLUSTERED INDEX idx\_product\_name ON Products(ProductName);

SELECT \* FROM Products WHERE ProductName LIKE 'Laptop%';

* **Result:** Quick product retrieval from a large inventory database.

**2. Banking Systems**

* **Scenario:** Retrieving customer account details using account numbers.
* **How Index Helps:**
  + A clustered index on the AccountNumber column ensures efficient retrieval of customer details.

Example:

CREATE CLUSTERED INDEX idx\_account\_number ON Accounts(AccountNumber);

SELECT \* FROM Accounts WHERE AccountNumber = '1234567890';

* **Result:** Faster account lookups in large transactional databases.

**3. Social Media Platforms**

* **Scenario:** Fetching user profiles based on usernames or email addresses.
* **How Index Helps:**
  + A unique index on the Username or Email column prevents duplicates and enables quick lookups.

Example:

CREATE UNIQUE INDEX idx\_unique\_username ON Users(Username);

SELECT \* FROM Users WHERE Username = 'john\_doe';

* **Result:** Fast profile access and duplicate prevention.

**4. Healthcare Systems**

* **Scenario:** Searching patient records by ID or date of birth.
* **How Index Helps:**
  + A composite index on PatientID and DOB enables quick retrieval of patient records.

Example:

CREATE INDEX idx\_patient\_id\_dob ON Patients(PatientID, DOB);

SELECT \* FROM Patients WHERE PatientID = 12345 AND DOB = '1990-01-01';

* **Result:** Efficient handling of large patient databases.

**5. Logistics and Supply Chain**

* **Scenario:** Tracking shipment details using tracking numbers.
* **How Index Helps:**
  + A non-clustered index on the TrackingNumber column speeds up shipment tracking.

Example:

CREATE NONCLUSTERED INDEX idx\_tracking\_number ON Shipments(TrackingNumber);

SELECT \* FROM Shipments WHERE TrackingNumber = 'TRK12345678';

* **Result:** Real-time updates and tracking.

**6. Online Booking Systems**

* **Scenario:** Fetching available slots or reservations by date and time.
* **How Index Helps:**
  + A composite index on BookingDate and TimeSlot improves query performance.

Example:

CREATE INDEX idx\_booking\_date\_time ON Bookings(BookingDate, TimeSlot);

SELECT \* FROM Bookings WHERE BookingDate = '2024-12-01' AND TimeSlot = '10:00 AM';

* **Result:** Fast access to booking data, enabling seamless user experience.

**7. Data Warehousing**

* **Scenario:** Running analytics and reports on large datasets.
* **How Index Helps:**
  + Bitmap indexes are used for columns with low cardinality (e.g., Gender, MaritalStatus).

Example:

CREATE BITMAP INDEX idx\_gender ON Customers(Gender);

SELECT Gender, COUNT(\*) FROM Customers GROUP BY Gender;

* **Result:** Improved performance for aggregation queries.

**8. Content Management Systems (CMS)**

* **Scenario:** Full-text search in articles or blog posts.
* **How Index Helps:**
  + A full-text index on the Content column enables efficient keyword searches.

Example:

CREATE FULLTEXT INDEX ON Articles(Content);

SELECT \* FROM Articles WHERE CONTAINS(Content, 'SQL Index');

* **Result:** Instant content discovery.

**9. Financial Applications**

* **Scenario:** Retrieving stock prices or transaction histories based on dates.
* **How Index Helps:**
  + A clustered index on the TransactionDate column ensures fast date-based queries.

Example:

CREATE CLUSTERED INDEX idx\_transaction\_date ON Transactions(TransactionDate);

SELECT \* FROM Transactions WHERE TransactionDate BETWEEN '2024-11-01' AND '2024-12-01';

* **Result:** Accelerated access to time-sensitive financial data.

**10. Real-Time Monitoring Systems**

* **Scenario:** Monitoring system logs and alerts.
* **How Index Helps:**
  + Indexes on Timestamp and LogLevel improve query performance for real-time log monitoring.

Example:

CREATE INDEX idx\_logs ON SystemLogs(Timestamp, LogLevel);

SELECT \* FROM SystemLogs WHERE LogLevel = 'ERROR' AND Timestamp > '2024-12-01 10:00:00';

* **Result:** Rapid identification of critical system issues.

**Summary of Benefits in Real-Time Use**

* **Fast Query Execution:** Minimizes the time to retrieve relevant rows.
* **Efficient Resource Usage:** Reduces CPU and I/O overhead during searches.
* **Optimized User Experience:** Enables seamless performance in applications.
* **Scalability:** Helps handle large datasets effectively.

Indexes are indispensable in applications with high data volume and frequent query operations.

some multiple-choice questions (MCQs) focused on **Indexes** in SQL:

**1. What is the primary purpose of an index in SQL?**

A) To store all the data in a table  
B) To speed up the retrieval of rows from a table  
C) To enforce referential integrity between tables  
D) To delete unnecessary rows from a table

**Answer**: B) To speed up the retrieval of rows from a table

**2. Which of the following is true about clustered indexes in SQL?**

A) A table can have multiple clustered indexes.  
B) Clustered indexes store data rows in the table.  
C) Clustered indexes are always created automatically.  
D) Clustered indexes can only be created on non-primary keys.

**Answer**: B) Clustered indexes store data rows in the table.

**Example:**

**Default clustered index on primary key**:

CREATE TABLE employees (

employee\_id INT PRIMARY KEY, -- Automatically creates a clustered index on employee\_id

name VARCHAR(100)

);

**Explicitly creating a clustered index** on a non-primary key:

CREATE CLUSTERED INDEX idx\_name

ON employees(name);

**Summary:**

A **clustered index** is created automatically **only for the primary key**. If you want a clustered index on other columns, you must create it manually

**3. In SQL, what is the default index type created on a primary key column?**

A) Unique index  
B) Full-text index  
C) Clustered index  
D) Non-clustered index

**Answer**: C) Clustered index

**4. What type of index is most suitable for columns that have unique values and are frequently used in WHERE clauses?**

A) Full-text index  
B) Clustered index  
C) Unique index  
D) Non-clustered index

**Answer**: C) Unique index

**5. Which SQL command is used to create an index on a table?**

A) CREATE TABLE  
B) CREATE INDEX  
C) CREATE UNIQUE  
D) ALTER INDEX

**Answer**: B) CREATE INDEX

**6. Which of the following is true about non-clustered indexes?**

A) They store the actual data of the table.  
B) A table can have only one non-clustered index.  
C) Non-clustered indexes store pointers to the actual data rows.  
D) Non-clustered indexes cannot be created on columns with NULL values.

**Answer**: C) Non-clustered indexes store pointers to the actual data rows.

**7. Which of the following columns would benefit the most from indexing?**

A) A column with a high number of NULL values  
B) A column that is frequently used in filtering and sorting operations  
C) A column that is rarely used in any query  
D) A column that is used for table joins but has few distinct values

**Answer**: B) A column that is frequently used in filtering and sorting operations

**8. What is the impact of using too many indexes on a table?**

A) Faster query performance and faster data retrieval  
B) Slower query performance but faster INSERT/UPDATE operations  
C) Slower query performance and slower INSERT/UPDATE operations  
D) No impact on query performance

**Answer**: C) Slower query performance and slower INSERT/UPDATE operations

**9. Which of the following is a disadvantage of using indexes?**

A) Indexes consume disk space.  
B) Indexes make INSERT and UPDATE operations slower.  
C) Indexes can be created on any column of the table.  
D) Both A and B

**Answer**: D) Both A and B

**10. Which of the following is NOT a type of index in SQL?**

A) Clustered index  
B) Non-clustered index  
C) Full-text index  
D) Data index

**Answer**: D) Data index

**11. In which scenario is a composite index most useful?**

A) When a query filters by multiple columns  
B) When a table has a small number of rows  
C) When a column is used only for sorting  
D) When a column has a large number of NULL values

**Answer**: A) When a query filters by multiple columns

**12. What does an index on a database column do?**

A) It sorts the data in the column.  
B) It speeds up queries that search or filter by that column.  
C) It prevents duplicate values in the column.  
D) It deletes the data in the column that does not match a given condition.

**Answer**: B) It speeds up queries that search or filter by that column.

**13. Which of the following is NOT a valid index type in SQL Server?**

A) Clustered index  
B) Unique index  
C) Hash index  
D) Full-text index

**Answer**: C) Hash index

**14. When should you consider dropping an index?**

A) When the index is frequently updated and slowing down write operations  
B) When the index is rarely used and consuming unnecessary resources  
C) When the query performance is unaffected by the index (if Index on Unused Columns, Small Tables)  
D) All of the above

**Answer**: D) All of the above

**15. Which of the following operations is NOT directly impacted by indexes?**

A) SELECT query performance  
B) INSERT operation  
C) DELETE operation  
D) TRUNCATE operation

**Answer**: D) TRUNCATE operation