**Assignment 5:** Demonstrate the creation of an index on a table and discuss how it

improves query performance. Use a DROP INDEX statement to remove the index and

analyze the impact on query execution.

Creating an index on a table in a database can significantly improve query performance, especially when dealing with large datasets. An index is a data structure that allows for efficient lookup, sorting, and retrieval of rows in a table based on the indexed columns. Let's demonstrate how to create an index and discuss its impact:

**Example Scenario**

Suppose we have a table called Products that stores information about various products, including their **'ProductID', 'ProductName', 'CategoryID'**, **'Price'**, and **'Stock'**.

**Creating an Index**

Let's create an index on the **'CategoryID**' column of the **'Products'** table, assuming that we frequently query products based on their category:

-- Create an index on CategoryID column

CREATE INDEX idx\_category\_id ON Products(CategoryID);

**Impact on Query Performance**

By creating an index on '**CategoryID'**, the database system creates a separate data structure that maps each '**CategoryID'** value to the corresponding rows in the '**Products'** table. This allows the database to quickly locate all rows that match a given '**CategoryID'** without having to scan the entire table sequentially.

For example, if you execute a query like:

SELECT \* FROM Products WHERE CategoryID = 5;

With the index '**idx\_category\_id'** in place, the database can quickly navigate to the subset of rows where '**CategoryID'** equals 5, resulting in faster query execution compared to a situation where no index exists and the database would have to scan through all rows of the '**Products'** table.

**Dropping the Index**

Now, let's see the impact of dropping the index:

-- Drop the index

DROP INDEX idx\_category\_id ON Products;

When you drop an index, the database no longer has the optimized data structure for '**CategoryID'**. Consequently, queries that depend on this index will likely see degraded performance, especially those involving filtering or sorting based on '**CategoryID'**.

**Analysis**

**Query Performance:** Queries that benefit from the index (**'CategoryID'** in our example) will likely see improved response times when the index is present.

**Storage and Maintenance:** Indexes consume additional storage space and incur overhead during data modification operations (like INSERT, UPDATE, DELETE), as the database needs to maintain the index structure.

**When to Use Indexes:** Indexes are beneficial for columns frequently used in WHERE clauses for filtering, ORDER BY clauses for sorting, and JOIN conditions for joining tables.

In summary, indexes are powerful tools for enhancing query performance by providing efficient data access paths. However, they should be used judiciously based on the specific access patterns and workload characteristics of your database application.