Query Optimization: A single query can be executed through different algorithms or re-written in different forms and structures. Hence, the question of query optimization comes into the picture – Which of these forms or pathways is the most optimal? The query optimizer attempts to determine the most efficient way to execute a given query by considering the possible query plans.

Importance: The goal of query optimization is to reduce the system resources required to fulfill a query, and ultimately provide the user with the correct result set faster.

- First, it provides the user with faster results, which makes the application seem faster to the user.
- Secondly, it allows the system to service more queries in the same amount of time, because each request takes less time than unoptimized queries.
- Thirdly, query optimization ultimately reduces the amount of wear on the hardware (e.g. disk drives), and allows the server to run more efficiently (e.g. lower power consumption, less memory usage).

There are broadly two ways a query can be optimized:

- Cost based: This was developed by IBM. The optimizer estimates the cost of each processing method of the query and chooses the one with the lowest estimate. Presently, most systems use this.
- Heuristic: Rules are based on the form of the query. Oracle used this at one point. Presently, no system uses this.

The query optimizer has the job of selecting the appropriate indexes for acquiring data, classifying predicates used in a query, performing simple data reductions, selecting access paths, determining the order of a join, performing predicate transformations, performing Boolean logic transformations, and performing subquery transformations—all in the name of making query processing more efficient.

Heuristic and rule based optimizers:

Heuristic based optimization uses rule-based optimization approaches for query optimization. These algorithms have polynomial time and space complexity, which is lower than the exponential complexity of exhaustive search-based algorithms. However, these algorithms do not necessarily produce the best query plan.

Some of the common heuristic rules are −

- Perform select and project operations before join operations. This is done by moving the select and project operations down the query tree. This reduces the number of tuples available for join.

- Perform the most restrictive select/project operations at first before the other operations.
- Avoid cross-product operation since they result in very large-sized intermediate tables.

Transaction

A transaction is an action or series of actions that are being performed by a single user or application program, which reads or updates the contents of the database.

A transaction can be defined as a logical unit of work on the database. This may be an entire program, a piece of a program, or a single command (like the SQL commands such as INSERT or UPDATE), and it may engage in any number of operations on the database. In the database context, the execution of an application program can be thought of as one or more transactions with non-database processing taking place in between.

Example of a Transaction in DBMS

A simple example of a transaction will be dealing with the bank accounts of two users, let say Scott and Smith. A simple transaction of moving an amount of 5000 from Scott to Smith engages many low-level jobs. As the amount of Rs. 5000 gets transferred from the Scott's account to Smith's account, a series of tasks gets performed in the background of the screen.

This straightforward and small transaction includes several steps: decrease Smith's bank account from 5000:

```
Open_Acc (Scott)

OldBal = Scott.bal

NewBal = OldBal - 5000

Ram.bal = NewBal

CloseAccount(Scott)
```

You can say, the transaction involves many tasks, such as opening the account of Scott, reading the old balance, decreasing the specific amount of 5000 from that account, saving new balance to an account of Scott, and finally closing the transaction session.

For adding amount 5000 in Smith's account, the same sort of tasks needs to be done:

OpenAccount(Smith)

Old_Bal = Smith.bal

NewBal = OldBal + 1000

Ahmed.bal = NewBal

CloseAccount(B)

Properties of Transaction

There are properties that all transactions should follow and possess. The four basic are in combination termed as ACID properties. ACID properties and its concepts are as follows:

- **Atomicity**: The 'all or nothing' property. A transaction is an indivisible entity that is either performed in its entirety or will not get performed at all. This is the responsibility or duty of the recovery subsystem of the DBMS to ensure atomicity.
- **Consistency:** A transaction must alter the database from one steady-state to another steady state. This is the responsibility of both the DBMS and the application developers to make certain consistency. The DBMS can ensure consistency by putting into effect all the constraints that have been mainly on the database schema such as integrity and enterprise constraints.
- **Isolation:** Transactions that are executing independently of one another is the primary concept followed by isolation. In other words, the frictional effects of incomplete transactions should not be visible or come into notice to other transactions going on simultaneously. It is the responsibility of the concurrency control sub-system to ensure adapting the isolation.
- **Durability:** The effects of an accomplished transaction are permanently recorded in the database and must not get lost or vanished due to subsequent failure. So this becomes the responsibility of the recovery sub-system to ensure durability.