Query Processing & Query Optimization

Query Processing: Overview, Measures of Query Cost, selection Operation, Sorting, Join operation, Other operations, Evaluation of Expressions.

1. Query Processing:

Query processing is a fundamental function performed by a DBMS to process user queries and retrive data from a DB.

It refers to the steps involved in processing users query (or request for data retrieval, modification (or deletion from a DB.

The process involves several stages, each designed to efficiently and accurately handle the users request.

Here is an overview of the query processing stages:

Parsing:

The first step is to analyze the query and break it down into its various components, such as keywords, table names, column names, conditions, and operators. This process is known as parsing.

Semantic Analysis:

After parsing, the system checks the query syntax and verifies if the tables, columns, and conditions mentioned in the query exist in the DB.

Query Optimization:

Query optimization is a critical step in the query processing.

The system analyzes various execution plans for the

query and selects the most efficient one based on factors like index usage, join order, and access methods.

Query Plan Execution:

Once the optimized query plan is determined, the DBMS executes the query against the DB.

Transaction Management:

Transaction management ensures that all operations in the transaction are completed successfully (or rolled back entirely in case of failure.

Concurrency Control:

In multi-user environments, concurrent execution of queries may lead to conflicts and inconsistencies. Concurrency control mechanisms are employed to maintain data consistency and prevent conflicts among simultaneous transactions.

Error Handling:

The DBMS handles any errors (or exceptions that may occur during the query processing phase. It provides appropriate error messages to the user (or) application and takes corrective actions if required.

Efficient query processing is essential for maintaining the performance and responsiveness of a DB system. Advanced DBMS implementations employ various techniques, such as indexing, caching, parallel processing and query optimization algorithms, to improve the overall query performance and reduce response times.

2. Measures of Query Cost:

Measures of query cost are used to evaluate the efficiency and performance of a query execution plan. These measures help DB administrators and developers to compare and select the most efficient query plan to minimize resource usage and response times. Some common measures of query cost include;

Execution Time:

Execution time is the elapsed time taken by the DBMS to process and execute the query. Lower execution time indicates better performance.

CPU Time:

CPU time represents the amount of time the CPU spends processing the query.

IO Cost:

IO cost measures the number of input and output operations performed during query execution.

Memory Usage:

Memory usage measures the amount of memory used by the query execution plan. Efficient use of memory is crucial for reducing disk IO and improving query performance.

Network Usage:

For distributed databases, network usage measures the data transferred b/w nodes during query processing.

Disk Space Utilization:
Disk space utilization measures the amount of disk space used to store temporary results (or) intermediate data during query execution.

Cardinality Estimation Errors:
Cardinality estimation errors refer to inaccuracies in estimating the number of rows returned by a query.
The DBMS uses cardinality estimates to choose the most efficient execution plan.

CPU Load:
CPU load measures the processing burden on the CPU during query execution.
High CPU load may lead to resource contention and impact overall system performance.

Response Time:
Response time is the time taken for the DBMS to process the query and return the results to the user.

Number of Joins and Nested loops:
The number of joins and nested loops in the query execution plan affects the complexity and efficiency of query processing.
By analyzing these measures, DB administrators can fine-tune query execution plans, optimize DB configurations, and allocate resources effectively to enhance the overall performance of the DB system.

5. Selection Operation in Query Processing:

The selection operation is performed by the file scan. The file scans are the search algorithms that are used for locating and accessing the data.
It is the lowest level operator used in query processing.
It is used to retrieve only the rows that meet the given criteria, reducing the data set size and improving query efficiency.
The selection operation is expressed using the SQL "WHERE" clause in a DB query.

Ex:-
```
select * from employees where salary > 5000;
```
Selection operations are essential for data retrieval and reporting tasks, as they allow users to narrow down the data set and focus on specific subsets of data that are relevant to their queries.

Selection operation with indexes:

In query processing, indexes play a crucial role in optimizing the selection operation.
The index based search algorithms are known as index scans. Such index structures are known as access paths. These paths allow locating and accessing the data in the file.
There are following algorithms that use the index in query processing.

Primary Index equality on a key:
The equality comparison is performed on the key attribute carrying a primary key.

Primary Index equality on nonkey:
We can fetch multiple records through a primary key when the selection criteria specify the equality comparison on a nonkey.

Secondary Index, equality on key or nonkey:
Using secondary Index strategy, we can either retrieve a single record when equality is on key or multiple records when the equality condition is on nonkey.

Selection operation with comparisons:

Selection operations with comparisons are used to retrieve specific rows from a table based on certain conditions using comparison operators.

The most commonly used comparison operators are:

(i) Equal to (=):
The equal to operator checks if the values on both sides of the operator are equal.

(ii) Not equal to (<>), (!=)

(iii) Greater than (>)

(iv) Less than (<)

(v) Greater than or equal to (>=)

(vi) Less Than or equal to (<=)

Eg- select * from employees where price > 200;
select * from orders where order_stat = "shipped";

Indexes allow the DB system to quickly locate the rows that satisfy the conditions, reducing the need for full table scans and improving query response times.

---

1. Sorting :-

In query processing, the sorting method is used for performing various relational operations such as joins, etc efficiently.
It involves arranging the retrieved data in a specific order based on one or more columns.
Sorting is commonly used to present query results in a meaningful and organized way.

Order by clause:
It allows you to sort the retrieved rows based on one or more columns in either ascending or descending order.
Eg: select col1, col2 from table_name order by col1;

Sorting Algorithms:-
DBMS uses various sorting algorithms to arrange data efficiently. Some common sorting algorithms include Bubble sort, Insertion sort, Quick Sort, Merge Sort, and Heap Sort.

External sorting:

Sorting large data sets do not fit entirely in memory can be challenging.

External sorting is a technique where the DBMS sorts the data that exceeds available memory by dividing it into smaller chunks and then merging those chunks in a sorted order.

Sorting is a computationally intensive operation for large datasets.

To improve the performance of sorting, DB often use various optimization techniques, including parallel processing, caching, and efficient memory management.

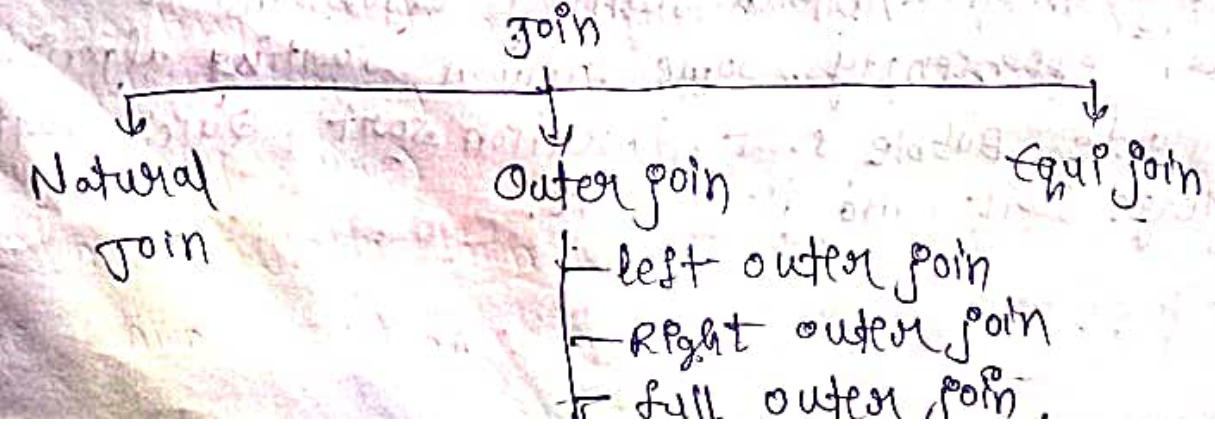
---

5. Join Operation and Other Operations:

Join processing uses the nested loop access method for all queries that contain joined columns.

A join operation combing related tuples from different relations, if and only if a given join condition is satisfied. It is denoted by ⋈.

Types of join operations.



Inner Join:
Retrieves rows from both tables that have matching values in the specified columns.

Left Outer Join:
Retrieves all rows from the left table and the matching rows from the right table.

Right Outer Join:
Retrieves all rows from the right table and the matching rows from the left table.

Full Outer Join:
Retrives all rows from both tables, including non-matching rows.

Join Algorithms in DB:-
These are two algorithms to compute natural join and conditional join of two relations. In DB.

(i) Nested Loop Join:
for each tuple t_R in T_R do
for each tuple t_S in T_S do
compare (t_R it$_S$) if they satisfies the condition
add them in the result of the join.
end
end

(ii) Block Nested Loop Join:
for each tuple - -
for each - -
for each - -

for each ---

end

end

end

end.

Other Operations:-

Projection Operation:

The projection operation is used to select specific columns from a table and retrieve only those columns in the query result.

Selection Operation:

The selection operation filters rows from a table based on specified conditions using comparison operators like =, >, < etc.

Aggregation Operation:

The aggregation operation is used to group data based on one or more columns and perform aggregate functions such as SUM, COUNT, AVG, MAX or MIN on the grouped data. (Group by clause)

Sorting Operation:

The sorting operation arranges the retrieved data in a specific order based on one or more columns. (order by clause)

Union and Intersection Operations:

The union operation combines the results of two or

more select queries into a single result set.
The intersection operation returns only the common rows b/w two (or more select queries.

Subquery Operation:
It is also called Nested query, a query nested within another query.
The DBMS query optimizer determines the most efficient execution plan for the query, considering the order and combination of these operations to minimize resource usage and execution time.

---

6. Evaluation of Expressions:
In DBMS expressions are used to perform calculations, comparisons, and manipulations of data within queries (or other DB operations.
When a query is processed, the DBMS evaluates expressions to compute the results.
Expressions can be simple (or complex, involving one (or more operands and operators.

Now let us try to understand how DBMS evaluates the query written in SQL.
i.e, how it breaks them into pieces to get the records quickly.

There are 2 methods of evaluating the query.
(i) Materialization
(ii) Pipelining

Materialization:

In this method, queries are broken into individual queries and then the results of which are used to get the final result.

Ex: select * from Stu s, class c where
s.class_id = c.class_id and
c.class_name = s.class_name;

Cost = cost of individual select + cost of write into temporary table.

Pipelining:

In this method, DBMS do not store the records into temporary tables.
It will process the query one after the other and each will use the result of previous query for its processing.

In this method no extra cost of writing into temporary tables. It has only cost of evaluation of individual queries.
Hence, it has better performance than materialization.

There are 2 types of pipelining.
(i) Demand Driven (or) Lazy evaluation.
(ii) Producer Driven (or) Eager Pipelining.

Demand Driven:

In this method, the result of lower level queries are not passed to the higher level automatically.

It will be passed to higher level only when it is requested by the higher level

Producer Driven:

In this method, lower level query creates a buffer to store the results and the higher level queries pulls the results for its use.

The lower level queries eagerly pass the result to higher level queries.

It does not wait for the higher level queries to request for the results.

If the buffer is full, then the lower level query waits for the higher level query to empty it.

Hence, it is also called as PULL and PUSH pipelining

There are still more methods of pipelining like linear and non-linear methods of pipelining, left deep tree, right deep tree etc.

Efficient evaluation of expressions contributes to optimal query performance and overall DB efficiency.

Query Optimization: Overview, Transformation of Relational Expressions, Estimating statistics of Expression result, Choice of Evaluation plans, Materialized views, Advanced Topics in Query Optimization.

---

1. Query Optimization:-

Query optimization is of great importance for the performance of a relational DB, especially for the execution of complex SQL statements.

A query optimizer decides the best methods for implementing each query.

Query optimization is a key technology for every application, from operational systems to data warehouse and analytical systems to content management systems.

There are various principles of query optimization are as follows -

(i) The first phase of query optimization is understanding what the DB is performing. Different databases have different commands.
Ex:- In Oracle, "EXPLAIN PLAN FOR" to see the query plan.

(ii) Retrieve as little data as possible
Ex:- select * ...

(iii) store intermediate results: Some times logic for a query plan can be quite complex. It is possible to produce the desired outcomes through the use

of subqueries, inline views and UNION type statements.

There are various query optimization strategies are as follows :-

(i) Use Index : It can be using an index to speed up a query.

(ii) Aggregate Table : It can be used to pre-populating tables at higher levels so less amount of information is required to be parsed.

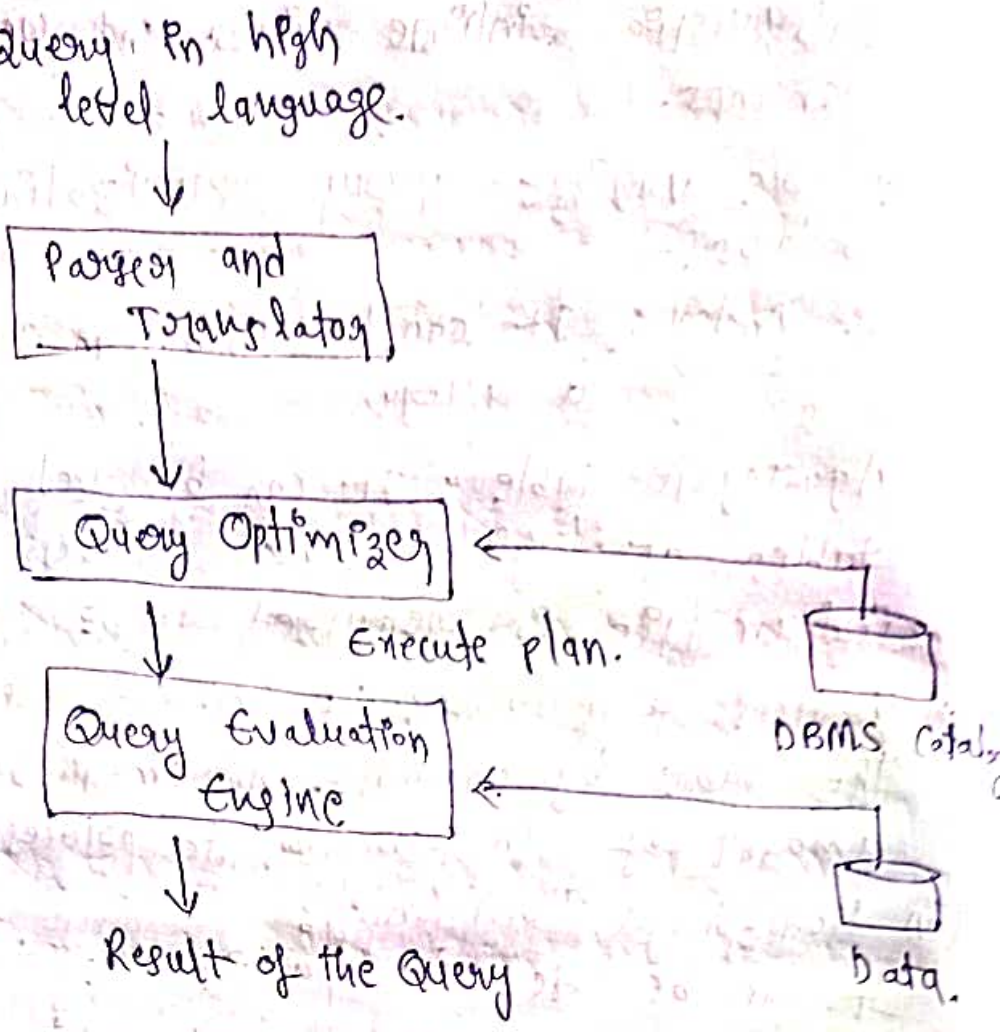
(iii) Vertical Partitioning : It can be used to partition the table by columns. This method reduces the amount of information to process.

(iv) Horizontal Partitioning :
It can be used to partion the table by data value. This method reduces the amount of information required to process.

(v) De-normalization :
The process of de-normalization combines multiple tables into a single table.

(vi) Server Tuning :
Each server has its parameters and provides tuning server parameters.

The goal of query optimization is to generate an optimal query execution plan that minimizes the overall resource usage and reduces the query response time.



2. Transformation of Relational Expressions :-

In query optimization, one of the essential tasks is the transformation of relational expressions.

The goal of transformation is to rewrite the original query expression into an equivalent but more efficient form.

This process involves applying various algebraic rules and logical transformations based on the properties of relational algebra.

Two relational algebra expressions are equivalent if both the expressions produce the same set of tuples on each legal DB instance.

A legal DB instance refers to that DB system which satisfies all the integrity constraints specified in the DB schema.

Equivalence Rules:

The equivalence rule says that expressions of two forms are the same (or equivalent because both expressions produce the same outputs on any legal DB instance.

The optimizer uses various equivalence rules on relational algebra expressions for transforming the relational expressions.

$\theta, \theta_1, \theta_2$ = Predicates

L_1, L_2, L_3 = List of attributes

E, E_1, E_2 = Relational algebra Expressions.

Rule-1: Cascade of σ

$$\sigma_{\theta_1 \wedge \theta_2}(E) = \sigma_{\theta_1}(\sigma_{\theta_2}(E))$$

Rule-2: Commutative Rule.

$$\sigma_{\theta_1}(\sigma_{\theta_2}(E)) = \sigma_{\theta_2}(\sigma_{\theta_1}(E))$$

$$E_1 \bowtie_\theta E_2 = E_2 \bowtie_\theta E_1$$

$\downarrow \bowtie_\theta$ = θ is in subscript with the join symbol

Rule-3: Cascade of Π

$$\Pi L_1 (\Pi L_2 (\cdots (\Pi L_n (E)) \cdots)) = \Pi L_1 (E)$$

Rule-4: We can combine the selections with Cartesian products as well as theta joins.

$$\sigma_\theta (E_1 \times E_2) = E_1 \bowtie_\theta E_2$$

$$\sigma_{\theta_1}(E_1 \bowtie_{\theta_2} E_2) = E_1 \bowtie_{\theta_1 \wedge \theta_2} E_2$$

Rule-5: Associative Rule.

$(E1 \bowtie E2) \bowtie E3 = E1 \bowtie (E2 \bowtie E3)$

$(E1 \bowtie_{\theta 1} E2) \bowtie_{\theta 2 \wedge \theta 3} E3 = E1 \bowtie_{\theta 1 \wedge \theta 3} (E2 \bowtie_{\theta 2} E3)$

Rule-6: Distribution of the selection operation over the theta join.

$\sigma_{\theta_0}(E_1 \bowtie_{\theta} E_2) = (\sigma_{\theta_0}(E_1)) \bowtie_{\theta} E_2$

$\sigma_{\theta_1 \wedge \theta_2}(E_1 \bowtie_{\theta} E_2) = (\sigma_{\theta_1}(E_1)) \bowtie_{\theta} (\sigma_{\theta_2}(E_2))$

Rule-7: Distribution of the projection operation over the theta join.

$\Pi_{L1 \cup L2}(E_1 \bowtie_{\theta} E_2) = (\Pi_{L1}(E_1)) \bowtie_{\theta} (\Pi_{L2}(E_2))$

Rule-8: The union and intersection set operations are commutative.

$E_1 \cup E_2 = E_2 \cup E_1$

$E_1 \cap E_2 = E_2 \cap E_1$

Rule-9: The union and intersection set operations are associative.

$(E1 \cup E2) \cup E3 = E1 \cup (E2 \cup E3)$

$(E1 \cap E2) \cap E3 = E1 \cap (E2 \cap E3)$

Rule-10: Distribution of selection operation on the intersection, union and set difference operations.

$\sigma_p(E_1 - E_2) = \sigma_p(E_1) - \sigma_p(E_2)$

$\sigma_p(E_1 - E_2) = \sigma_p(E_1) - E_2$

Rule-11: Distribution of the projection operation over the union operation.

$$\Pi_L (\epsilon_1 \cup \epsilon_2) = (\Pi_L(\epsilon_1)) \cup (\Pi_L(\epsilon_2))$$

3. Estimating Statistics of Expression Results:

Estimating statistics of expression results is an important aspect of query optimization in a DBMS.

Statistics provide valuable information about the data distribution and cardinality of the results of various expressions in the DB.

These statistics are used by the query optimizer to make informed decisions about the most efficient query execution plan.

Cardinality Estimation:

Cardinality refers to the number of distinct values (or rows in a table (or expression result.

Histograms:

Histograms are used to represent the data distribution of column values in a table (or) expression result.

Histogram help the optimizer in making more accurate cardinality estimates and selecting appropriate join strategies.

Selectivity Estimation:

Selectivity refers to the proportion of rows that satisfy a particular condition in a where clause, or predicate expression.

The DBMS estimates the selectivity of expressions using statistics on the data distribution.

Sampling:

In some cases, collecting statistics on the entire data set can be resource intensive.
To overcome this, the DBMS may use sampling techniques to estimate statistics.

Join Cardinality Estimation:

Estimating the cardinality of join operations is crucial for selecting the optimal join strategies, such as nested loop, hash join, or merge join.

Cost Estimation:

Once the optimizer has estimated the statistics of expression results, it uses cost based optimization techniques to evaluate different query plans and select the one with the lowest estimated cost.

Ex: Suppose we have a DB table named "employees" with millions of records, and it contains the following columns: "emp_id", "first_name", "last_name", "age", and "dept_id".

Query: SELECT first_name, last_name
FROM employees
WHERE age > 30 AND dept_id = 101;

4. Choice of Evaluation Plans:-

The choice of evaluation plan in DBMS is a critical step in query optimization.

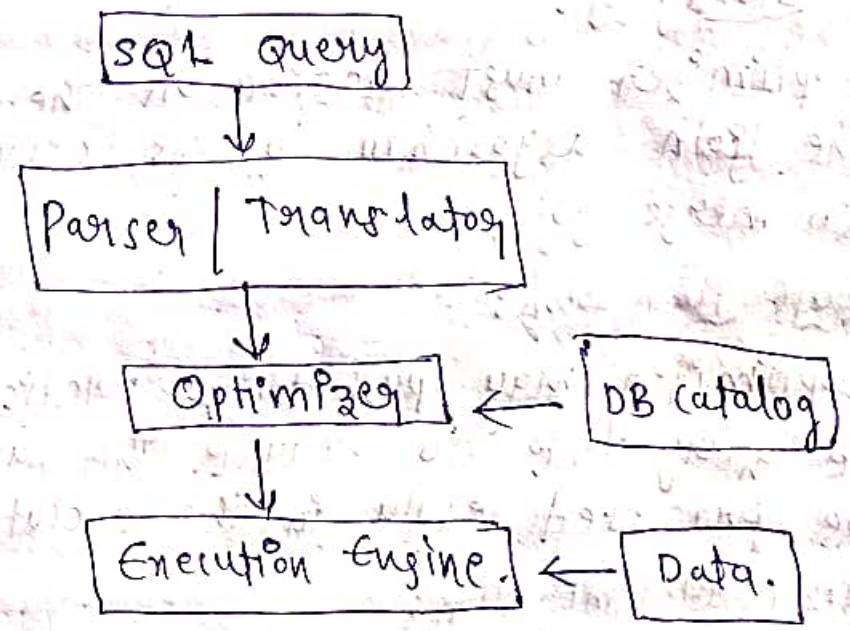
The goal is to select the most efficient execution plan to process a given query.

The query optimizer considers various factors and explores different plan alternatives to determine the optimal plan.

Query Evaluation Plan:

It is nothing but a program for an abstract machine inside the DBMS. It is produced by the query optimizer.

Query evaluation plans are very much similar to relational algebra expressions in most of the systems.



Here are some key aspects that influence the choice of an evaluation plan in a DBMS.

Cost Based Optimization:
Many modern DBMS use cost based optimization, where the optimizer estimates the cost of different query execution plans.
The cost includes factors like CPU usage, memory consumption, disk IO, and network traffic.
Query Statistics:
The query optimizer relies on statistics about the DB and its contents such as cardinality of columns, data distribution, and selectivity of conditions
Index Selection:
The optimizer evaluates the available indexes and considers whether to use them for efficient data access (or table scans.
Join Strategies:
The optimizer must decide on the order of joins and the join algorithm to use (nested loop, hash join or merge join).
Predicate Push down:
The optimizer may push down predicates (conditions) in the query tree to reduce the amount of data to be processed early in the execution.
Subquery Optimization:
Handling subqueries efficiently is vital for query performance.
Parallelism:
In some cases, the optimizer may explore parallel

execution plans, especially for queries that can be divided into independent tasks.

The choice of an evaluation plan is a complex process involving a trade off b/w multiple factors, the optimizer aims to find a plan that minimizes the overall resource usage and maximizes query performance.

5. Materialized Views:

A materialized view is a view whose contents are computed and stored.

A materialized view is also a logical virtual table. The result of the query is stored in the table on disk.

The performance of the materialized view is better than normal view.

It is also called indexed views, since the table created after the query is indexed and can be accessed faster and efficiently.

Eg:- create view loan(branch, total) as select branch, sum(amount) from bank group by branch;

The task of keeping a materialized view up-to-date with the underlying data is known as materialized view maintenance. It can be maintained by recompilation on every update.

View maintenance can be done by following:

→ Manually defining triggers on insert, delete and

update of each relation in the view definition.
→ Manually written code to update the view.
→ Supported directly by the DB.

Materialized views, also known as materialized tables or materialized query tables, are DB features in DBMS that allow the storage of pre-computed results of queries as physical tables.

Materialized views are created using the CREATE MATERIALIZED VIEW statement.

Once created, the materialized view holds the query results as a physical table.

To ensure that the materialized views data remains up-to-date, it requires regular refreshing. Refreshing can be done manually or automatically.

Advantages:

(i) Improved Query Performance:
Materialized views reduce query execution time by providing precomputed results.

(ii) Reduced DB load:
By using materialized views, the workload on the DB server is reduced

(iii) Offline Query Support:
Materialized views allow running queries against the pre computed results even when the original source data is temporarily unavailable.

Disadvantages and Considerations:

(i) Storage Overhead:
Materialized views consume additional storage space to hold the precomputed results.

(ii) Data Staleness:
Depending on the refresh frequency, there might be a time gap b/w when the source data changes and when the materialized view is updated, leading to potential data staleness.

(iii) Maintenance Overhead:
Materialized views require regular maintenance to keep the data up-to-date.

In summary, materialized views provide a powerful mechanism for improving query performance and reducing DB workload by storing precomputed results of complex queries.

---

6 Advanced Topics in Query Optimization:-

The advanced topics are designed to handle specific challenges and scenarios that arise in large-scale and complex DB systems.

Here are some of the advanced topics in query optimization:

Multi-Query Optimization:
Multiquery optimization involves optimizing a group of related queries together to improve overall performance.

Join Ordering Optimization:
Join ordering optimization focuses on finding the best join order for queries involving multiple tables.

Advanced Indexing Strategies:

Advanced indexing techniques, like bitmap indexes, function-based indexes, and compressed indexes, aim to improve query performance for specific query patterns and data distributions.

Join algorithms and Join Reordering:

Beyond the basic join algorithms (nested loop, hash join, merge join), advanced topics explore hybrid join algorithms, adaptive join algorithms, and advanced join reordering heuristics to handle different types of data distributions and query conditions.

Subquery Optimization:

Advanced techniques include subquery unnesting, subquery materialization, and correlated subquery optimization to improve subquery processing.

Parallel Query Execution:

Parallel query execution involves breaking down a query into smaller tasks and executing them concurrently on multiple CPU cores or nodes.

Query Optimization in Distributed Databases:

In distributed DBs, query optimization becomes more complex due to data distribution, network latency, and coordination among distributed nodes.

ML based Optimization:

Some advanced query optimizers use ML techniques to learn from historical query performance.