Database Management System(303105203)

Computer Science & Engineering

Introduction:

- Query Processing
- Layers of Query Processing
- Measures of Query Cost
- File Scan(Linear & Binary Search)
- Materialized View & Pipelining.
- Query Optimization
- Equivalence Rules
- Cost-Based Query Optimization

Query Processing

- Query processing is a translation of high-level queries into low-level expression.
- It is a step wise process that can be used at physical level of the file system, query optimization and actual execution of query to get the result.
- Its refers to the range of activities that are involved in extracting data from the database.

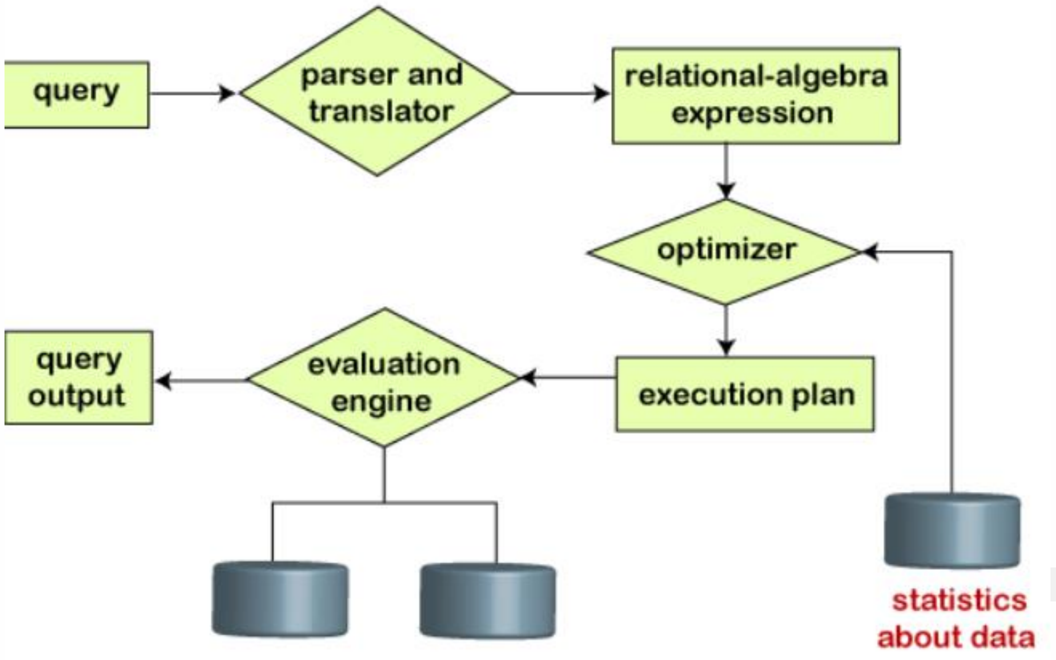
Need for Querying Processing

1. **Interpreting the Query**: Understanding and converting the query from a high-level language (like SQL) into a format that the database can process.
2. **Searching Through Data**: Accessing and retrieving the relevant data from the database storage that satisfies the query conditions.

Note: Overall, query processing involves transforming a user's request into actions that efficiently retrieve the desired information from a database.

Step in Query Processing:

1. Parsing and translation
2. Optimization
3. Evaluation



Process of SQL

1. **Parsing:** The parser takes the SQL query as input and checks it for syntax and semantic correctness.

- **Components:**

i. Lexical Analyzer: Breaks the query into tokens.

ii. Syntax Analyzer: Verifies the query structure.

iii. Semantic Analyzer: Checks the validity of the query against the database schema (e.g., table names, column names).

2. Optimization

- Query optimization is a crucial phase in query processing where the database management system (DBMS) seeks to determine the most efficient way to execute a given query.
- This process involves generating, evaluating, and selecting the best possible execution plan from many alternatives.

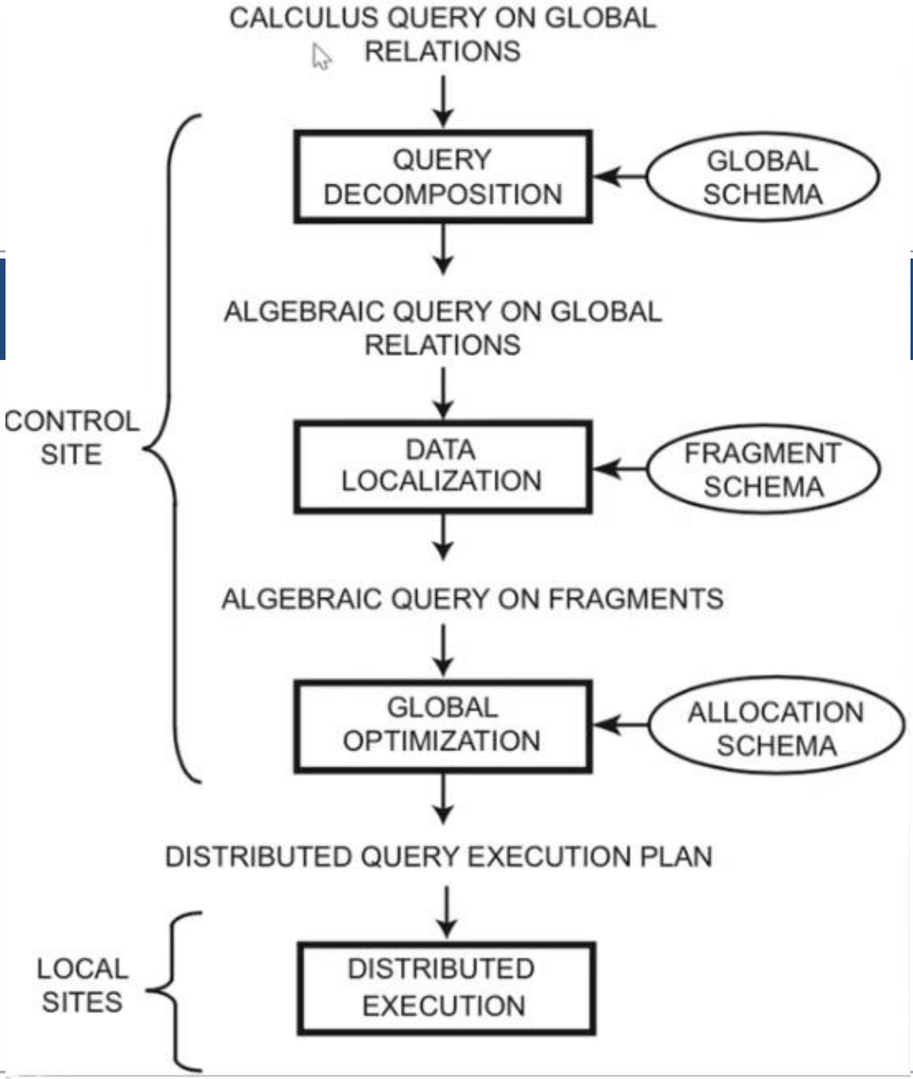
Key Objectives of Query Optimization

- **Minimize Response Time**: Reduce the time it takes to return the query results.
- **Minimize Resource Utilization**: Efficiently use system resources like CPU, memory, and disk I/O.
- **Ensure Scalability**: Maintain performance as the size of the database and the number of queries increase.

3. **Evaluation:** Finally runs the query and display the required result.

Layers of Query Processing:

1. Query Decomposition
2. Data Localization
3. Global Query Optimization
4. Distribution Query Execution



Query Decomposition: The first layer decomposes the calculus query into an algebraic query on global relations. The information needed for this transformation is found in the global conceptual schema describing the global relations.
Query decomposition can be viewed as four successive steps.
i) Normalization
ii) Analysis
iii) Simplification
iv) Restructure

- **First,** the calculus query is rewritten in a normalized form that is suitable for subsequent manipulation.
- **Second,** the normalized query is analyzed semantically so that incorrect queries are detected and rejected as early as possible.
- **Third,** the correct query is simplified. One way to simplify a query is to eliminate redundant predicates.
- **Fourth,** the calculus query is restructured as an algebraic query and transform it in order to find a go.

Data Localization: The input to the second layer is an algebraic query to localize the query's data using data distribution information in the fragment schema.

- **Generating a query on fragments is done in two steps:**
  - First, the query is mapped into a fragment query.
  - Second, the fragment query is simplified and restructured to produce another "good" query.

Global Query Optimization: The input to the third layer is an algebraic query on fragments.

The goal of query optimization is to find an execution strategy for the query which is close to optimal.

- Query optimization consists of finding the "best" ordering of operators in the query that minimize a cost function.

Distribution Query Execution: The last layer is performed by all the sites having fragments involved in the query.

Measures of Query Cost

- Query Cost is a cost in which the enhancer considers what amount of time your query will require.

- It involves query processing time i.e.; time taken to parse and translate the query, optimize it, evaluate, execute and return the result to the user is called cost of the query.

➢ The key factors influencing query cost include:

1.CPU Time

2.I/O Operations

3. Memory Usage

4. Network Latency

5. Disk Access

➢ The cost of query evaluation can be measured in terms of:

i)The number of disk accesses.

ii)Time of Execution taken by the CPU to execute a query.

iii)The involved Communication costs in either distributed or parallel database systems.

File Scans

- File scans refer to the methods used to locate and retrieve records from a database file. When a query is executed, the DBMS may need to scan the file to find the relevant records.

- The efficiency of these scans can significantly impact query performance.

- There are two scan algorithms to implement:

1. Linear Search
2. Binary search

1. **Linear Search:** also known as sequential search.

It is a fundamental algorithm used in DBMS to find a specific record within a database file.

- It operates by checking each record in sequence until the desired record is found or the end of the file is reached.

Steps of Linear Search in a DBMS

1. **Start at the Beginning:** The search process begins with the first record in the database file.
2. **Check Each Record:** Each record is examined one by one to see if it matches the search criteria (e.g., a specific ID, name, or other attributes).
3. **Compare the Record:** The search key (the value you are looking for) is compared with the corresponding attribute in the current record.

4.Continue Sequentially: If the current record does not match the search key, the search moves to the next record in the sequence.

5.Repeat Until Found or End: This process is repeated until either:

- ➢ The desired record is found (a match is found).
- ➢ The end of the file is reached without finding a match.

6.Return Result: If a match is found, the record is returned.

->If no match is found after scanning all records, the search concludes that the record does not exist in the file.

2. Binary search: Binary search is a more efficient search method that requires the data to be sorted. It works by repeatedly dividing the search interval in half.

- **Preconditions for Binary Search:**

1. **Sorted Data**: The data must be sorted based on the search key. Binary search is only applicable to ordered datasets.

2. **Access Method**: The DBMS must be able to quickly access the middle element of the current search interval, which is usually facilitated by indexing or direct access methods.

Steps of Binary Search in a DBMS:

1. Initialize Search Interval:Start with the entire dataset as the initial search interval. This means you consider the first record to the last record.
2. Find Middle Record: Calculate the middle index of the current search interval.
3. Compare and Narrow Down: Compare the search key with the key of the middle record.

- If the search key matches the middle record's key, the search is successful, and the record is returned.
- If the search key is less than the middle record's key, narrow the search interval to the left half (all records before the middle record).

- If the search key is greater than the middle record's key, narrow the search interval to the right half (all records after the middle record).

4. **Repeat**: Continue dividing the interval and comparing until the search key is found or the interval is empty.

5. **Return Result**: If the search key is found, return the corresponding record.

->If the search interval becomes empty (no more records to check), conclude that the record is not present in the dataset.

Materialized View

- A materialized view in a database management system (DBMS) is a database object that contains the results of a query.
- Unlike a regular view, which is a virtual table computed dynamically upon access, a materialized view stores the query result physically on disk.
- This allows for quicker access to complex query results, as the data is pre computed and stored, but it also requires a management to keep the data up to date.

NOTE:

- Materialized views are a powerful tool in DBMS for improving query performance, especially for complex and frequently executed queries.

- Properly managing refresh strategies is crucial to ensure the data in materialized views remains accurate and up to date.

Pipelining

- Pipelining in the query treatment means the method is based on the approach of splitting the query processor into multiple mini-processes, which help to perform parallel tasks and, as a result, increase the efficiency of the queries.

- Pipelining is a technique where multiple instructions are overlapped during execution.

- Pipelining techniques are crucial for improving query performance and resource utilization in modern database management systems.

Parul® University

❖ How pipelining works in the context of DBMS:

1.Parallel Execution: Different stages of a query can be processed in parallel.

2. Reduced I/O Wait Times: By overlapping operations, pipelining can reduce the time spent waiting for disk I/O operations.

3.Improved Resource Utilization: Pipelining can make better use of CPU, memory, and disk resources by keeping all parts of the system busy.

4.**Examples in SQL Queries**: In an SQL query, pipelining might be used when combining multiple operations such as joins, selections, and projections. For instance, if a query involves joining two tables and then filtering the results, pipelining allows the filtering to start as soon as the first rows of the join are produced.

5.**Pipeline Execution Plans**: Modern database systems often create execution plans that take advantage of pipelining. These plans specify the order of operations and how data flows between them to maximize efficiency.

Pipelining	Materialization
It is a modern approach to evaluate multiple operations.	It is a traditional approach to evaluate multiple operations.
It does not use any temporary relations for storing the results of the evaluated operations.	It uses temporary relations for storing the results of the evaluated operations. So, it needs more temporary files and I/O.
It is a more efficient way of query evaluation as it quickly generates the results.	It is less efficient as it takes time to generate the query results.
It requires memory buffers at a high rate for generating outputs. Insufficient memory buffers will cause thrashing.	It does not have any higher requirements for memory buffers for query evaluation.
Poor performance if trashing occurs.	No trashing occurs in materialization. Thus, in such cases, materialization is having better performance.

Query Optimization

- Query optimization in a Database Management System (DBMS) is the process of improving the execution efficiency of a query.
- The goal is to find the most efficient way to execute a given query by considering various possible query execution plans and selecting the one with the lowest estimated cost.

Parul® University

Key Concepts:

1. **Query Execution Plan (QEP)**: A QEP is a sequence of operations that the DBMS will perform to execute a SQL query. It includes choices like the order of table joins, the use of indexes, and the selection of specific algorithms for operations like sorting and filtering.

2. **Cost Model**: The optimizer uses a cost model to estimate the resource consumption (CPU, memory, I/O) of different query plans. The plan with the lowest estimated cost is chosen.

Equivalence Rules

- Equivalence rules in a Database Management System (DBMS) are fundamental principles used during query optimization to transform queries into different but equivalent forms.
- These transformations help the query optimizer generate alternative execution plans and choose the most efficient one.

Key Equivalence Rules:

1.Commutativity of Join:

->The order of joining tables can be switched without affecting the result.

->Rule: $R \bowtie S \equiv S \bowtie R R \bowtie S \equiv S \bowtie R$

->Examples:

SELECT * FROM employee e JOIN department d ON e.dept_id = d.id;

is equivalent to:

SELECT * FROM department d JOIN employee e ON d.id = e.dept_id;

2. **Associativity of Join**:

->The grouping of joins can be changed without affecting the result.

->Rule: $(R \bowtie S) \bowtie T \equiv R \bowtie (S \bowtie T)$(R⋈S)⋈T≡R⋈(S⋈T)

->Example:

SELECT * FROM (employee e JOIN department d ON e.dept_id = d.id) JOIN location l ON d.location_id = l.id;

is equivalent to:

SELECT * FROM employee e JOIN (department d JOIN location l ON d.location_id = l.id) ON e.dept_id = d.id;

Parul® University

3. **Selection Distribution over Join**:

->A selection condition can be applied before or after a join.

->Rule: $\sigma\theta(R \bowtie S) \equiv R \bowtie \sigma\theta(S)\sigma\vartheta(R \bowtie S) \equiv R \bowtie \sigma\vartheta(S)$ if $\theta\vartheta$ only involves attributes of SS.

->Example:

SELECT * FROM employee e JOIN department d ON e.dept_id = d.id WHERE d.location = 'New York';

Is equivalent to:

SELECT * FROM employee e JOIN (SELECT * FROM department WHERE location = 'New York') d ON e.dept_id = d.id;

4. Projection Distribution over Join:

->Projection can be applied before or after a join.

->Rule: $\Pi A(R \bowtie S) \equiv \Pi A(\Pi B(R) \bowtie \Pi C(S))$$\Pi A(R \bowtie S) \equiv \Pi A(\Pi B(R) \bowtie \Pi C(S))$, where $B \subseteq A \cup KR$$B \subseteq A \cup KR$ and $C \subseteq A \cup KS$$C \subseteq A \cup KS$.

->Example:

SELECT e.name, d.name FROM employee e JOIN department d ON e.dept_id = d.id;

Is equivalent to:

SELECT e.name, d.name FROM (SELECT id, name FROM employee) e JOIN (SELECT id, name FROM department) d ON e.dept_id = d.id;

Parul® University

5. Selection Distribution over Union:

->Selection can be distributed over a union operation.

->Rule: $\sigma\theta(R \cup S) \equiv \sigma\theta(R) \cup \sigma\theta(S) \sigma\vartheta(R \cup S) \equiv \sigma\vartheta(R) \cup \sigma\vartheta(S)$

->*Example:*

SELECT * FROM (SELECT * FROM employee WHERE dept_id = 10) UNION (SELECT * FROM employee WHERE dept_id = 20);

Is equivalent to:

SELECT * FROM employee WHERE dept_id = 10 UNION SELECT * FROM employee WHERE dept_id = 20;

Parul® University

DIGITAL LEARNING CONTENT

6. Projection Distribution over Union:

->Projection can be distributed over a union operation.

->Rule: $\Pi A(R \cup S) \equiv \Pi A(R) \cup \Pi A(S) \Pi A(R \cup S) \equiv \Pi A(R) \cup \Pi A(S)$

->Example:

SELECT name FROM (SELECT name, dept_id FROM employee WHERE dept_id = 10 UNION SELECT name, dept_id FROM employee WHERE dept_id = 20);

Is equivalent to:

SELECT name FROM employee WHERE dept_id = 10 UNION SELECT name FROM employee WHERE dept_id = 20;

Cost-Based Query Optimization

- Cost-based query optimization is an overall process of choosing the most efficient means of executing a SQL statement based on overall cost of the query.

- The efficient execution is the execution with minimum cost.

- In this optimization technique, all of the possible ways or scenarios in which a query can be executed will be assigned a ‘cost’, which indicates how efficiently that query can be run.

- The optimizer will pick the scenario that has the least cost and execute the query using that scenario, because the query with least cost is the most efficient way to run the query.
- Some features of the cost-based optimization:

1. The query can use a lot of paths based on the value of indexes, available sorting methods, constraints, etc.
2. The aim of query optimization is to choose the most efficient path of implementing the query at the possible lowest minimum cost in the form of an algorithm.
3. The cost of executing the algorithm needs to be provided by the query Optimizer so that the most suitable query can be selected for an operation.

Thank You!

DIGITAL LEARNING CONTENT

Parul® University

www.paruluniversity.ac.in