

Query Processing and Optimization

Concept of Query Processing: [Imp.]

Query processing is the process of executing a database query or request for information. It involves several steps that transform a high-level query written in a programming language (such as SQL) into instructions that can be understood and executed by database management system (DBMS).

Steps involved in query processing:

- i) Parsing and Validation: The DBMS first parses the query to check its syntax and ensure that it conforms to the language's grammar rules. It verifies the query's structure and validates it for correctness.
- ii) Semantic analysis: The DBMS performs semantic analysis to ensure that the query makes sense in the context of the database schema and the available data. It checks for things like table and column existence, data types etc.
- iii) Query Optimization: It explores different execution plans to determine the most efficient way to retrieve the requested data. Optimization is performed by optimizer which considers factors like indexes, statistics, join algorithms and other techniques.
- iv) Plan generation: It specifies the sequence of operations to execute query. This plan may involve accessing one or more tables, applying filters, performing joins, and other operations.
- v) Plan execution: Plan execution may involve disk I/O, memory management, and CPU operations to process the data efficiently.
- vi) Result retrieval: Finally, the DBMS retrieves the result set and returns it to the application that initiated the query. The result may be returned in various formats depending on the query and the application's requirements.

Query Trees: [Imp.]

Query trees, are a hierarchical representation of the steps and operations involved in executing a database query. They are used in query processing to outline the logical and physical operations required to retrieve the desired data from the database.

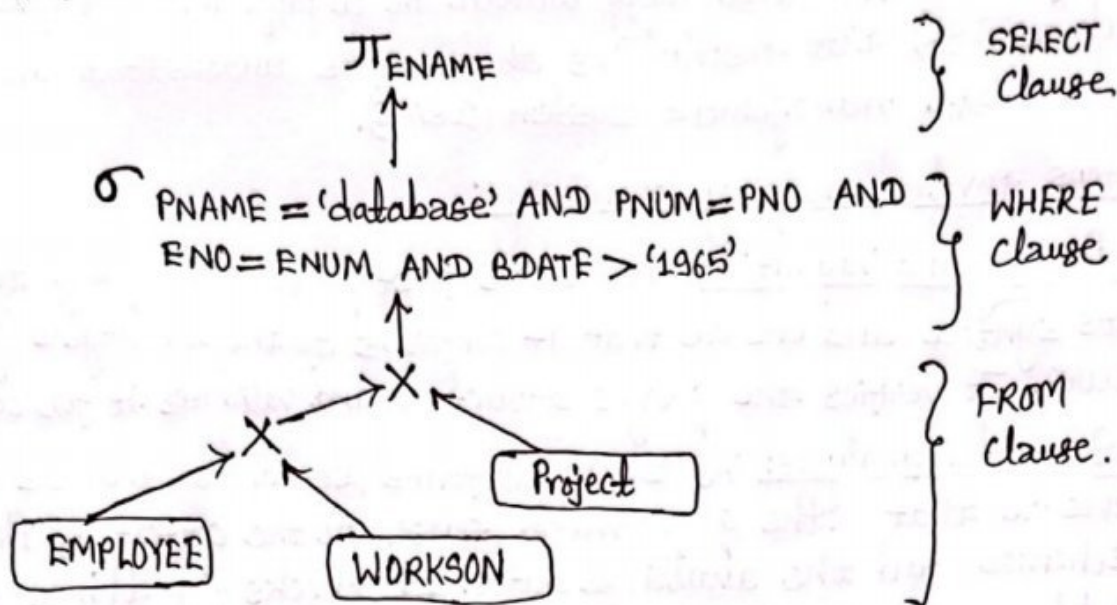


Fig: An example of query tree.

A query tree is typically represented as a tree-like structure, with each node representing an operation or transformation applied to the data. The nodes are connected by edges that indicate the flow of data between operations. The root node represents the final result of the query, and the leaf nodes represent the base tables or data sources being accessed.

⊗ Why do we need query trees in query processing?

Ans: Following are the reasons to demonstrate why query trees are needed in query processing:

1) Visualization and Understanding: Query trees provide a visual representation of the query execution plan, making it easier to understand and analyze the steps involved in processing a query.

2) Query optimization: By representing the query execution plan as a tree, the optimizer can efficiently explore and evaluate various plan alternatives to find the most efficient one.

28.

iii) Execution order and data flow: Query trees define the order in which the operations are performed and how data flows between them.

iv) Query plan caching and reuse: Query trees can be cached and reused for similar queries with different parameter values. This improves query performance by avoiding re-computation.

Query Optimization: [Imp]

Query Optimization refers to the process of improving the performance and efficiency of database queries. It involves the most effective way to retrieve data. The primary goal of query optimization is to minimize the execution time and resource consumption of a query such as CPU usage, disk I/O, and network traffic.

⊗ Why do we need query optimization in databases?

Ans: Query optimization is essential in database for following reasons:

→ Improved Performance: Query optimization helps improve the performance of database systems by reducing the execution time of queries.

→ Resource Utilization: Query optimization aims to minimize resource consumption such as CPU usage, disk I/O, and network traffic.

→ Cost Reduction: Query optimization helps reduce costs by minimizing resource usage and improving overall efficiency of query execution.

→ Scalability: Optimization techniques such as indexing, caching, and parallel processing help ensure that the system can handle growing workloads and provide consistent performance.

→ Complex Query Support: ~~And By~~ Query optimization techniques consider various factors, such as query rewriting, join ordering, and index selection, to generate effective execution plans for complex queries.

→ Application Responsiveness: It provides faster responses, leading to better user experiences.

Heuristics for Query Optimization:

Heuristics refer to rules or guidelines that guide decision-making process of the query optimizer. Heuristics are practical techniques based on general principles and assumptions about the behaviour of database systems and the characteristics of queries and data. They are not guaranteed to always find the optimal solution but aim to find good solutions within a reasonable amount of time and resources.

⊗ Heuristic Rules:

- They are used as an optimization technique to modify the internal representation of query.
- Usually, these rules are in the form of query tree or query graph data structure, to improve its performance.
- One of the main heuristic rule is to apply SELECT operation before applying the JOIN or other binary operations.
- SELECT and PROJECT reduces the size of the file and hence, should be ~~use~~ applied before JOIN or other binary operation.
- Heuristic query optimizer transforms the initial query tree into final query tree using equivalence transformation rules.

⊗ Some General transformation rules:

i) Cascade of σ :

$$\sigma_{C_1} \text{ AND } \sigma_{C_2} \text{ AND } \dots \text{ AND } \sigma_{C_n}(R) \equiv \sigma_{C_1}(\sigma_{C_2}(\dots(\sigma_{C_n}(R))\dots))$$

ii) Commutativity of σ :

$$\sigma_{C_1}(\sigma_{C_2}(R)) \equiv \sigma_{C_2}(\sigma_{C_1}(R))$$

iii) Commutating σ with π :

$$\pi_{A_1, A_2, \dots, A_n}(\sigma_C(R)) \equiv \sigma_C(\pi_{A_1, A_2, \dots, A_n}(R))$$

iv) Commutating of \bowtie AND \times :

$$R \bowtie_C S \equiv S \bowtie_C R$$

$$R \times S \equiv S \times R$$

v) Commutating of set operations ($\cup, \cap, -$):

$$R \cup S \equiv S \cup R, \quad R \cap S \equiv S \cap R, \quad R - S \neq S - R$$

Query Optimization Strategies: [V. Imp.]

Query Optimization Strategies for lowering the execution time or increasing the performance of queries include following:

i) Cost Based Optimization:

Cost based optimization is a process of selecting lower cost mechanism and is based on the cost of query. The query can use different paths based on indexes, constraints, sorting methods etc. This method mainly uses statistics like record size, no. of records, table size, no. of blocks, size of columns and their uniqueness etc.

Features:

- ↳ It is based on cost of the query to be optimized.
- ↳ Query can use a lot of paths: indexes, constraints etc.
- ↳ Aim is to choose the most efficient path of implementing query at possible lowest minimum cost in form of an algorithm.
- ↳ It depends upon cardinality of input.

ii) Heuristic Based Optimization:

Heuristic based optimization transforms the query-tree by using a set of rules that typically improves execution performances. Some common rules are:

- ↳ Perform selection early (reduces the no. of tuples).
- ↳ Perform projection early (reduces the no. of attributes).
- ↳ Perform most restrictive selection and join operations.

iii) Semantic Based Optimization:

Semantic based optimization uses constraints specified on the database schema, such as unique attributes and other more complex constraints, in order to modify one query into another query that is more efficient to execute.

E.g:

```
SELECT e.lname, m.lname
FROM EMPLOYEE as e, EMPLOYEE as m
WHERE e.super_ssn = m.ssn and e.salary > m.salary;
```

Choice of Query Execution Plans: [Imp]

The "Choice of Query Execution Plans" refers to the process of selecting the most efficient and effective plan for executing a database query in a relational database management system. When a query is submitted to a database, the database engine analyzes the query and generates multiple possible execution plans. The goal is to select a plan that minimizes the overall cost of executing the query, which typically involves factors such as disk I/O, CPU utilization, and memory consumption.

The optimizer aims to estimate the cost of each plan and selects the one with the lowest estimated cost. Factors that influence the choice of query execution plans include table and index statistics, database configuration settings, available system resources, join conditions etc. By choosing the optimal execution plan, database systems can significantly improve query performance, reduce resource usage, and provide faster response times for users.

Q. How can you choose the evaluation plans?

Ans: Write above topic "Choice of Query Execution Plans".

same above topic
Query execution plan
को सही evaluation
plan को मिलाने
लेना।

Q. How can you measure the cost of query?

Ans: Cost of query is the time taken by the query to hit the database and return the result. Query cost considers the no. of different resources such as: no. of disk accesses, no. of block transfer, size of table, time taken by CPU for executing the query etc. The time taken by CPU is negligible in most system when compared with the no. of disk accesses. If we consider the no. of block transfer as main component in calculating the cost of query, it would include more subcomponents such as Rotational latency, seek time, Sequential I/O, Random I/O etc.

$$\text{Query Cost} = b \times t_T + s \times t_S$$

where, b = block transfer

s = seeks

t_T = time to transfer 1 block

t_S = time for one seek.

If $t_T = 0.1\text{ms}$, $t_S = 4\text{ms}$ then
block size = 4 KB

transfer rate = 40 MB per second.

Q. How can you express the query to optimized/tree transformation.

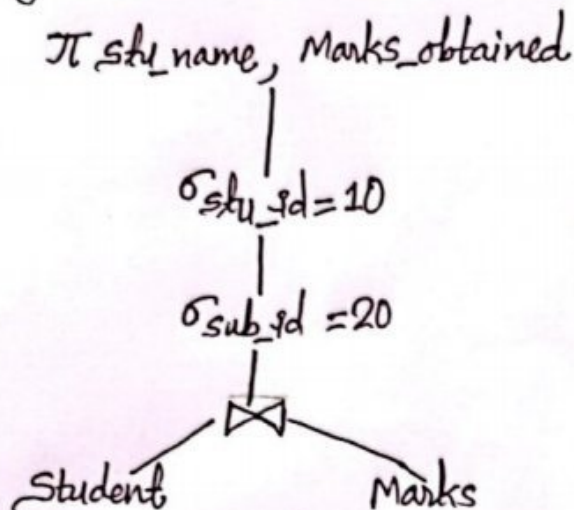
Ans: After a high-level query has been parsed into an equivalent relational algebra expression, the query optimizer can perform heuristic rules on the expression and tree, to transform the expression and tree into equivalent but optimized forms.

E.g: $\text{SELECT stu_name, Marks_obtained}$
 $\text{FROM student, Marks}$
 $\text{WHERE stu_id} = 10 \text{ and sub_id} = 20;$

Now, the corresponding relational algebra expression is:

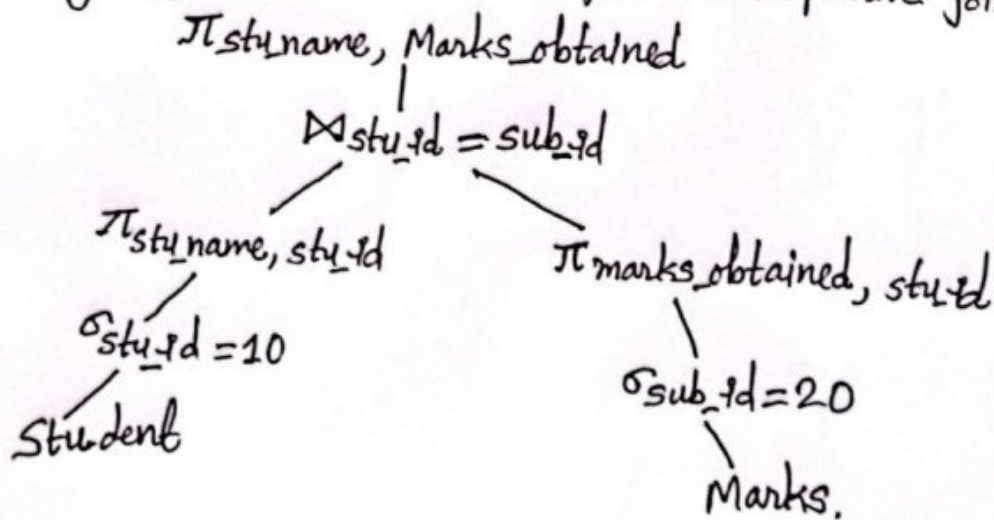
$\pi \text{ stu_name, Marks_obtained} (\sigma \text{ stu_id} = 10 (\sigma \text{ sub_id} = 20 (\text{Student} \bowtie \text{Marks})))$

Now, Query tree is as follows:



Join operation

Here, Cartesian product can be reduced. If $\sigma \text{ stu_id} = 10$ operation is performed first. We can also combine the $\sigma \text{ stu_id} = 20$ and cartesian product operations into a more efficient join operation as well as eliminating any unneeded columns before the expensive join performed.



Heuristic query optimization vs. Cost-based query optimization: ^[Impl]

	Heuristic Query Optimization	Cost-based query optimization
Approach	Relies on predefined rules or heuristics.	Considers actual cost and statistical information.
Decision-Making	Based on common knowledge and intuition.	Based on estimated costs and statistical analysis.
Complexity	Simple and easy to implement.	More complex to implement and maintain.
Adaptability	Does not adapt well to changing data or query workloads.	Can adapt to changing workloads based on updated stats.
Dependency	Independent of accurate statistical data.	Relies heavily on accurate statistical data.
Overhead	Low Computational overhead	Higher computational overhead due to cost estimation.
Plan exploration.	Limited exploration of alternative plans.	Explores various plans and join orders.