examples. [1+2+2]

What are the different steps involved in Query Processing? Explain with a flow diagram. What are the different approaches for Query Optimization? Explain in brief. [4+4]

What do you mean by RAID? Brief

5. Explain the basic steps in query processing with a diagram. What is pipelining evaluation in a query? Explain with an example. [5+3]

. Define query processing and briefly explain its steps. How is pipeline approach different from the materialization approach? [1+1+6] [4+4]

. Why does RAID Level 6 give better data

Describe the basic steps in query processing. Explain how pipelining can be used to imporve query evalution efficiency. [5+3]

6. Explain the basic steps in query processing with diagram? What is pipelining in query evaluation. Explain with an example. [4+2] [5+3]

7. a) What do you mean by hashing and indexing? Differentiate between dense

. Describe the basic steps in query processing. Discuss the methods used for evaluation of entire expression tree. [2+3] [4+3]

a) What is the difference

. Explain with diagram about process of query processing in RDBMS. How are equivalence rules for relational algebra helpful for query optimization? Explain with example. [5+3]

**What is the task of evaluation engine in query processing? Explain cost based query optimization and Heuristic optimization. [4+4]**

. Define query processing. Explain the various approaches used to evaluate any expression with suitable example. [2+6]

Explain the steps involved in query processing. What is the significance of materialized views? [6+2]

Write about fixed length record and variable length

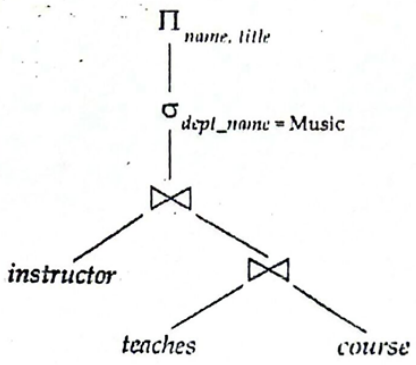
6. Explain how a DBMS chooses an appropriate query execution plan for optimized query execution. Explain the difference between materialization and pipelining methods for query evaluation? [5+3]

6. Explain the difference between cost-based and heuristics-based methods for query optimization. How can you optimize the following query? [3+5]

$$\Pi_{name,\ title}(\sigma_{dept\_name=\text{"Music"}}(instructor \bowtie \Pi_{course\_id,\ title}(teaches \bowtie course)))$$

5. Explain the process how a query is evaluated in RDBMS systems. How are equivalence rules for relation algebra helpful for query optimization? Explain with example. [3+5]

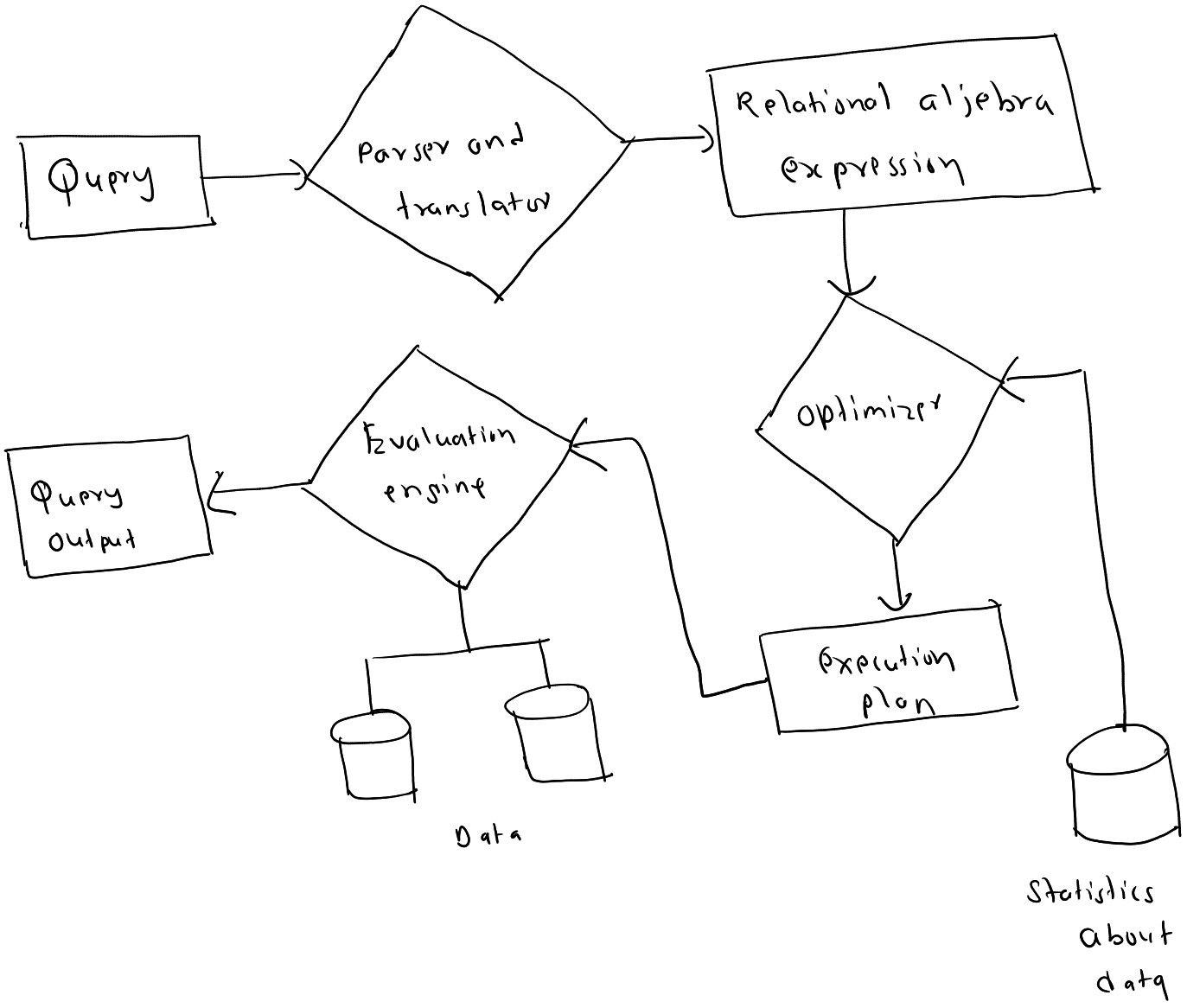
6. Explain the basic steps in query processing. Optimize the following query expression. [6+2]



## Query Processing in DBMS

Query processing converts high-level queries (e.g., SQL) into low-level instructions that a database engine can execute. The general process can be broken down into three major phases:

1. Parsing and Translation
2. Optimization
3. Evaluation (Execution)



## 1. Parsing and Translation

-> The DBMS first checks the query for correctness in terms of SQL syntax. If there are syntax errors, the query is rejected immediately, and an error is returned to the user.

-> After confirming the syntax is valid, the DBMS ensures the query's components (tables, columns, functions, etc.) exist and that the user has the necessary privileges to access them.

-> The valid SQL query is transformed into a parse tree (or AST). This tree structure represents the logical components of the query.

-> The DBMS then converts the parse tree into a relational algebra expression.

Example:

SELECT Ename from Employee WHERE Salary > 5000

↓

$\Pi_{Ename}(\sigma_{salary>5000}(Employee))$

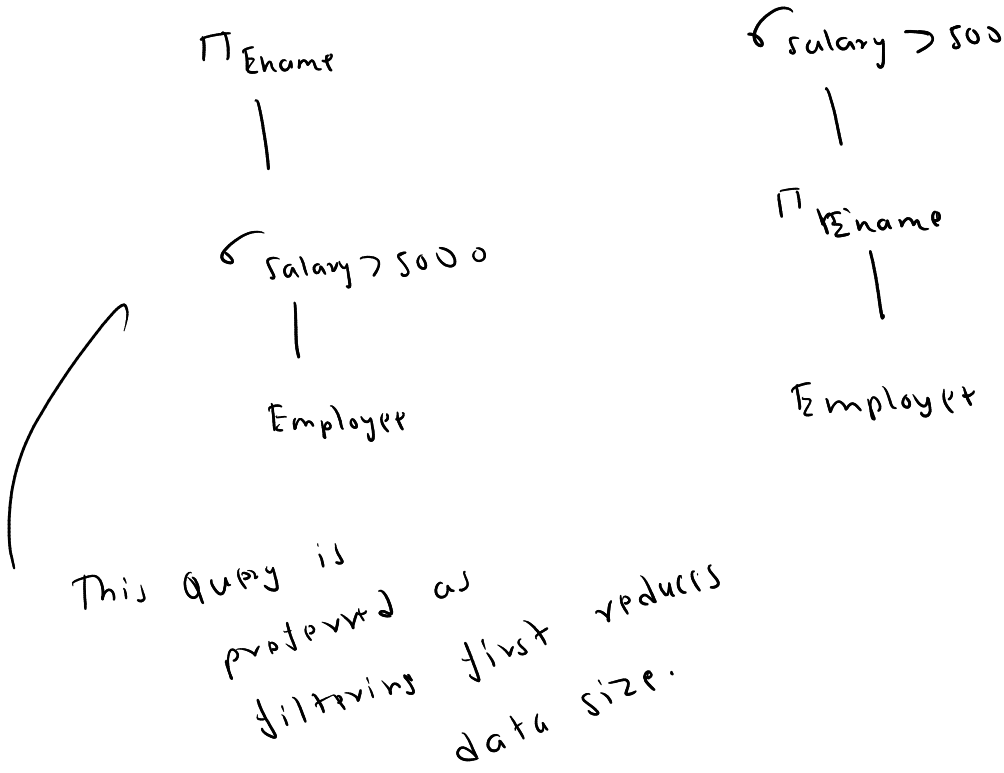
or $\sigma_{salary>5000}(\Pi_{Ename}(Employee))$

## 2. Query Optimization

-> The system applies a set of rules to simplify or restructure the relational algebra expression without changing its meaning.

-> The optimizer consults statistics (e.g., table sizes, data distribution, available indexes) stored in the database catalog. This helps it estimate the cost of various access methods (e.g., index scan vs. full table scan) and join algorithms (e.g., nested loop join, merge join, hash join) and makes a Query Execution Plan (QEP).

-> For each possible execultion plan, the optimizer estimates a "cost" in terms of I/O operations, CPU usage, memory usage. It chooses the plan that is estimated to have the lowest overall cost to execute.




### 3. Evaluation (Execution)

-> The Evaluation engine takes the optimized QEP and executes it step by step.

-> Performs operations (e.g., selection and projection) as per the chosen QEP.

-> Return the final result set to the user.

## Evaluation of Expressions in Query Processing

-> Two primary evaluation methods: Materialization and Pipelining.

### 1. Materialization

-> Materialization involves computing the output of a query operator and storing (or "materializing") the entire result in a temporary structure (often in memory or on disk) before it is consumed by the next operator in the query plan.

## How It Works:

-> The first operator computes its result set.

-> The entire result is stored as an intermediate relation.

-> The next operator reads from this stored intermediate result.

Example :

(1) $\Pi_{ename}$ ( $\sigma_{salary > 50000}$ (Employees ⋈ Departments))

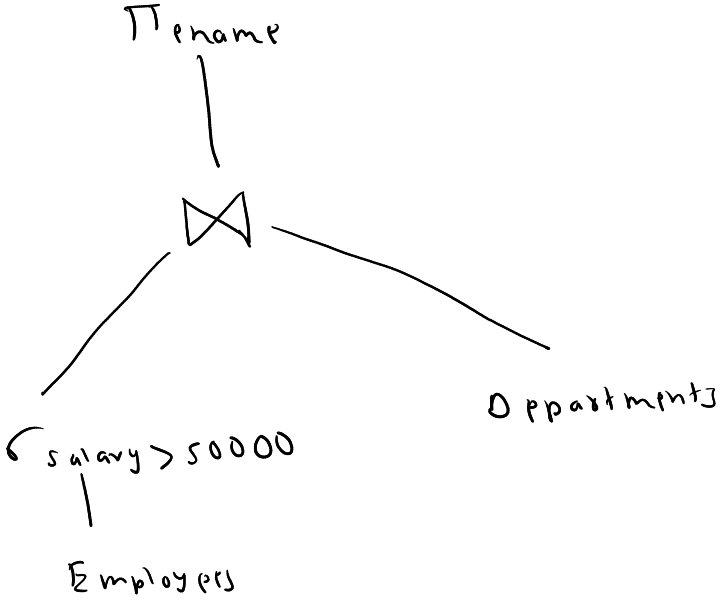



fig : pictorial representation

1. Apply selection ( $\sigma$ )
2. Save result on disk
3. Join result with another table Depart
4. Materialize the join result
5. perform the projection on the final result ($\Pi$ ename)

With materialization:

-> The database first computes the selection and writes all qualifying Employees into a temporary table.

-> Then, the join operator reads the entire temporary table and performs the join with Departments.

Advantages:

-> Easy to implement and reason about.

-> Changes in one operator do not affect another since the intermediate result is fixed once stored.

-> The same intermediate result can be used for multiple subsequent operations

Disadvantages:

-> Writing and reading from temporary storage can be costly in terms of I/O operations, especially with large datasets.

-> It requires additional storage space for the materialized intermediate results

Pipelining

Pipelining allows operators to process data in a streaming fashion. The output of one operator is passed immediately to the next operator without having to store the entire result set. Data flows as a "pipeline" of tuples from one operator to the next.

How It Works:

-> The first operator begins processing and outputs tuples one at a time.

-> As soon as a tuple is produced, it is passed directly to the next operator.

-> This process continues in a pipeline until the final result is produced.

Example -> Same Query

-> The selection operator starts scanning the Employees table.
-> As soon as it finds an employee with a salary above 50,000, it sends that tuple directly to the join operator.
-> The join operator processes each incoming tuple on the fly, matching it with tuples from Departments without waiting for the full set of selected employees to be computed.
-> result is projected as soon it is available

Advantages:
-> Results are available sooner since there is no waiting for an entire intermediate result to be computed and stored.
-> Minimizes the need to write large intermediate results to disk, leading to faster query execution in many cases.
-> Since only a small part of the data is processed at a time, it can be more memory-efficient.

Disadvantages:
-> If an error occurs, the pipeline might need to be re-executed or rolled back.
-> Once a tuple has passed through the pipeline, it isn't stored. If multiple operators need the same intermediate result, it may need to be recomputed.

Query Optimization

1. Cost-Based Query Optimization (CBO)

Cost-based optimizers evaluate multiple execution plans for a given query by estimating the "cost" of each plan. The cost is a numeric value that represents the expected resource usage (I/O operations, CPU cycles, memory consumption, etc.) associated with executing a plan. The optimizer uses statistics about the database, such as table sizes, index cardinality, and data distribution, to compute these costs.

## Example

-> Consider two tables: Employees and Departments.

```
SELECT E.name, D.dept_name
FROM Employees E
JOIN Departments D ON E.dept_id = D.id
WHERE D.location = 'New York';
```

-> The optimizer knows how many rows are in each table and how selective the condition D.location = 'New York' is.
-> Plan A: Filter Departments by location first, then join with Employees using an index on dept_id.
-> Plan B: Join Employees and Departments first and then apply the filter.
-> If filtering Departments first greatly reduces the number of rows to join, the cost model might estimate a lower cost for Plan A.
-> The optimizer selects the plan with the lowest cost (likely Plan A).

## Heuristic-Based Query Optimization

-> Heuristic optimizers use a set of rules or "heuristics" rather than detailed cost calculations to simplify the optimization process. These rules are derived from common-sense approaches and general best practices rather than statistical analysis.

## Common Heuristics

-> Apply filtering conditions as early as possible in the execution plan to reduce the number of rows processed in later stages.

-> Generally, smaller tables or highly selective filters are joined first.

-> Rewrite the query to eliminate redundancies (e.g., removing unnecessary subqueries or expressions).

## Example

```
SELECT E.name, D.dept_name
FROM Employees E
JOIN Departments D ON E.dept_id = D.id
WHERE D.location = 'New York';
```

-> The heuristic rule would push the WHERE D.location = 'New York' clause to the point where the Departments table is accessed. This immediately reduces the size of Departments.

-> Heuristically, the optimizer may decide that since Departments is filtered by a specific location, it is beneficial to use this smaller, filtered set to drive the join with Employees.

-> Without detailed cost calculations, the heuristic optimizer builds a plan that first retrieves the filtered Departments and then performs the join with Employees.

While heuristics are faster (because they do not require detailed cost estimations), they may not always produce the optimal plan in every scenario, especially when the underlying data distribution is complex.

## Advantages of Cost-Based Optimization

-> Uses detailed statistics to select the plan with the lowest estimated cost.

-> Can adapt to complex queries with multiple joins, subqueries, and non-standard operations.

-> As data distributions change, the optimizer can choose different plans based on updated statistics.

## Advantages of Heuristic Optimization

-> Faster decision-making since it avoids heavy computations.

-> Uses a fixed set of rules that are easier to implement and understand.

-> In many cases, common-sense rules lead to reasonably efficient plans without the overhead of cost calculations.

## Query decomposition (Not asked Juts read it once or twice just in case)

Query decomposition is the process of breaking a complex SQL query into smaller, manageable subqueries or operations

Stages of Query Decomposition:

1. Normalization: Convert the query into a standard format.
2. Analysis: Identify and validate query constraints and requirements.
3. Simplification: Remove redundant conditions and optimize expressions.
4. Partitioning: Break the query into subqueries or smaller tasks.
5. Optimization: Improve query execution using indexes, joins, and other techniques.

Benefits:

->Improves performance by reducing query execution time.

-> Makes complex queries easier to manage and debug.

-> Allows parallel execution of subqueries for faster results.

-> Enhances database optimization by reducing resource consumption.

$$\Pi_{name,\ title}(\sigma_{dept\_name=\text{"Music"}}(instructor \bowtie \Pi_{course\_id,\ title}(teaches \bowtie course)))$$

**Optimized Query Expression (Combined)**

Putting it all together, one succinct way to write it is:

$$\Pi_{name,\ title}\left((\pi_{ID,\ name}(\sigma_{dept\_name='Music'}(\text{instructor}))) \bowtie (\pi_{ID,\ course\_id}(\text{teaches})) \bowtie (\pi_{course\_id,\ title}(\text{course}))\right)$$

→ explain with logic from notes