Query Optimisation:[$]

What are the steps in query processing?

High Level Query	→ Scanning and Syntax checking	→ Intermediate Form
Intermediate Form	→ **Query Optimizer**	→ Execution Plan
Execution Plan	→ Query Code Generator	→ Code to Execute Query
Code to Execute Query	→ Run time Database Program	→ Result

What is a query optimizer?

The **query optimizer** is the component of a database management system that attempts to determine the most efficient way to execute a query. The optimizer considers the possible query plans for a given input query, and attempts to determine which of those plans will be the most efficient. Cost-based query optimizers assign an estimated "cost" to each possible query plan, and choose the plan with the least cost. Costs are used to estimate the runtime cost of evaluating the query, in terms of the number of I/O operations required, the CPU requirements, and other factors. The set of query plans examined is formed by examining the possible access paths (e.g. index scan, sequential scan) and join algorithms (e.g. sort-merge join, hash join, nested loops). The search space can become quite large depending on the complexity of the SQL query.

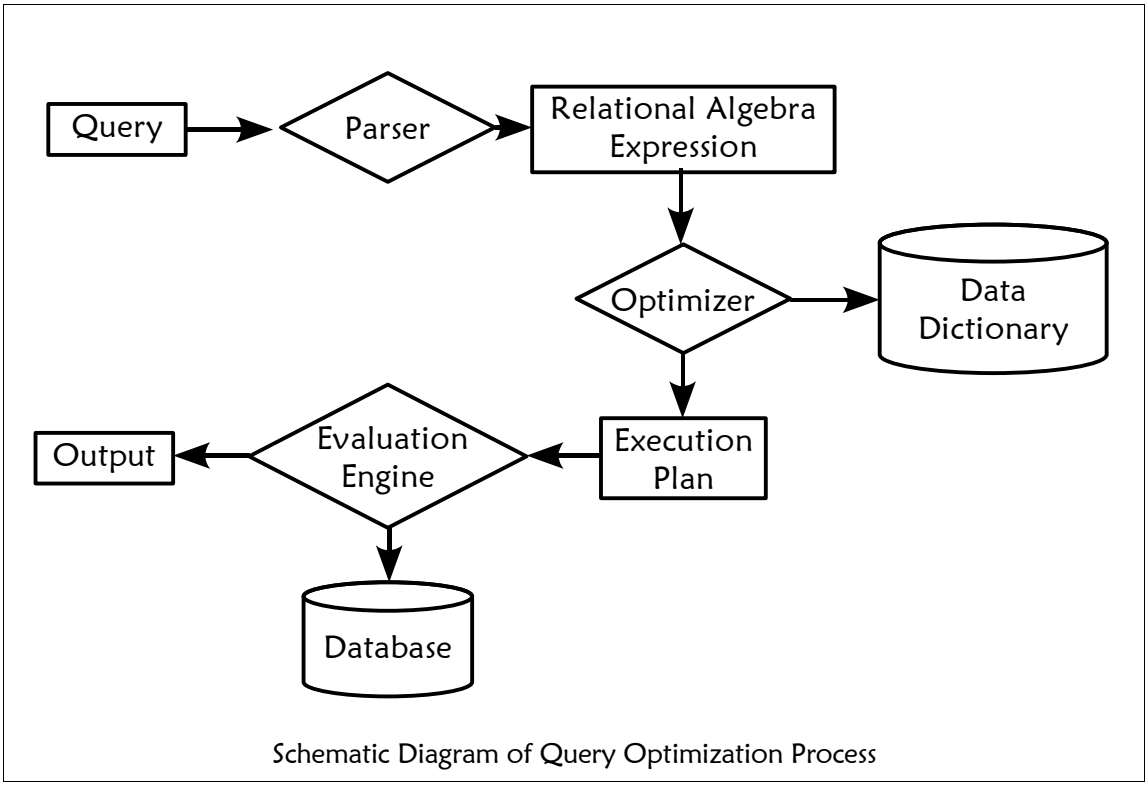
The query optimizer cannot be accessed directly by users. Instead, once queries are submitted to database server, and parsed by the parser, they are then passed to the query optimizer where optimization occurs.

What is Query Optimization Process?
Query optimisation process is the procedure of selecting the best plan or strategy to be used in responding to a database request.
To process a query depends upon the following steps:

1. Query: Supplied by external user to process a request.
2. Parsing and translation: The user query is then convert into the relational algebra by the help of a translator. In this step system use the operator tree (also query tree) to convert the query into relational algebra form.
3. Relational algebra expression: The query is expressed in relational algebraic form by taking the help of operators like pi,sigma,cross,join etc.
4. Optimizer: The optimizer check whether the related data is present in the database or not. If data present then the optimizer search the best plan to evaluate the query.
5. Evaluation plan: To find the best solution, the next step is the evaluation plan. Here the optimizer choose one of the best strategy and then the optimizer pass that plan to the evaluation engine.
6. Evaluation Engine: The query evaluation engine access the required data from the database.

$ Parts adapted from Lecturer Pelve Mogin

7. Query Output: After evaluating the query the output is produced.



Schematic Diagram of Query Optimization Process

Example:

Q: Find those employee records where the salary of the employee is greater than 2000 and deptno is 20.

Step1 – Finding out the Query

Let's make an unoptimized query; First find out the deptno from emp where empno is 20, some sort of a redundant query and then employees working in deptno 20 and sal>2000.

SELECT * FROM EMP WHERE SAL>2000 AND DEPTNO=(SELECT DISTINCT DEPTNO FROM EMP WHERE DEPTNO=20);

A query can always be written in more than one way. So the user can express the query in any form but it is the responsibility of the system to convert it to an efficient one before execution. The previous query should normally have been written as:

SELECT * FROM EMP WHRE SAL>2000 AND DEPTNO=20;

However it is not our task, but the task of the optimizer.

Step2 – Parsing the Query

Let's take the first form of the query and put it in an operator tree (query tree)

σsal>2000
↓
σdeptno=20
↓
Emp

Step3 – Relational Algebra Expression

σsal>2000(σdeptno=20 **(Emp)**)

Step4 – Finding the Best Solution

The cascading selections are added up together using an AND to obtain the following

σsal>2000 ∧ deptno=20

Step5 – till Output

The second strategy obtained is an optimized one so we can expect that it will be passed on for processing and output is obtained.

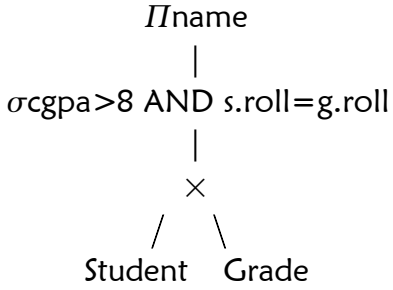
Example:

Q: Find the student whose CGPA is greater than 8 from the following schema
Student(Name,Roll,Address) Grade(Roll,CGPA)

Step1

SELECT NAME FROM STUDENT S, GRADE G WHERE S.ROLL=G.ROLL AND CGPA>8

Step2

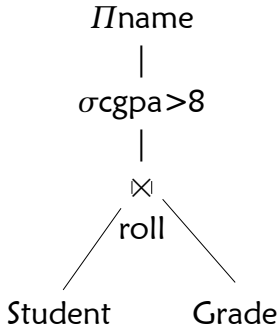


Step3

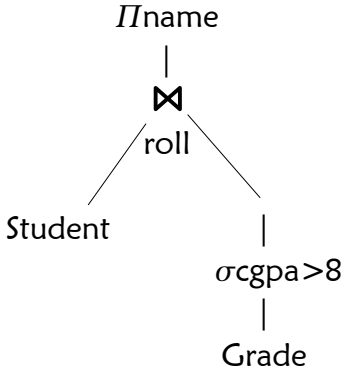
Πname(σcgpa>8 AND s.roll=g.roll**(Student×Grade)**)

Step4

Finding an Optimal Solution 1



2nd solution



Now we convert it to: Πname Student$\bowtie$(σcgpa>8**(Grade)**)
Step 5 till 7
As the obtained query Πname Student$\bowtie$(σcgpa>8**(Grade)**) is the optimal one it will be passed for processing and output obtained.

NOTE: In finding out the optimal solution to a query, the most important point is to find out the equivalences for existing algebraic expressions.

Transformations of an expression to its Equivalent Expressions

1. Combining a cascade Selection
2. Combining a cascade Projection
3. Commute Selection and Projection
4. Use associative and commutative rules for joins and Cartesian product
5. Perform selections before join or Cartesian product
6. Perform modified projection before a join
7. Commuting Projection with a Cartesian Product
8. Commuting Projection with Union
9. Commuting Selection with Union
10. Commuting Selection with difference

1. Combining a cascade Selection

σcond1(σcond2(σcond3(R))) ≡ σcond1∧cond2∧cond3(R)

2. Combining a cascade Projection

ΠcolsX(ΠcolsY(ΠcolsZ(R))) ≡ ΠcolsX(R) where X ⊆ Y ⊆ Z

3. Commute Selection and Projection

σcond(ΠcolsX(R)) ≡ ΠcolsX(σcond(R))

4. Use associative and commutative rules for joins and Cartesian product

R1 ⋈ R2 = R2 ⋈ R1
(R1 ⋈ R2) ⋈ R3 = R1 ⋈ (R2 ⋈ R3)
(R1 × R2) × R3 = R1 × (R2 × R3)

5. Perform selections before join or Cartesian product

σ_{cond}(R1 ⋈ R2) will be converted to (σ_{cond}R1) ⋈ R2

6. Perform a projection before thejoin

Π_{colsX}(R1 ⋈ R2) will be converted to (Π_{colsX}R1) ⋈ R2

7. Commuting Projection with a Cartesian Product

Π_{colsX}(R1 × R2) will be converted to Π_{colsX1}R1 × Π_{colsX2} R2

8. Commuting Projection with Union

Π_{colsX}(R1 ∪ R2) will be converted to Π_{colsX}R1 ∪ Π_{colsX}R2

9. Commuting Selection with Union

σ_{cond}(R1 ∪ R2) will be converted to σ_{cond}R1 ∪ σ_{cond}R2

10. Commuting Selection with difference

σ_{cond}(R1 — R2) will be converted to σ_{cond}R1 — σ_{cond}R2

What is a Query Tree?

Query tree is a tree data-structure like presentation which corresponds to a relational algebra statement/expression. All the input relations here are the leaf nodes and relational algebraic operators are the internal nodes.

The execution of the query tree is from bottom to top and when the root node is processed, the execution terminates and result is obtained.

Points:-

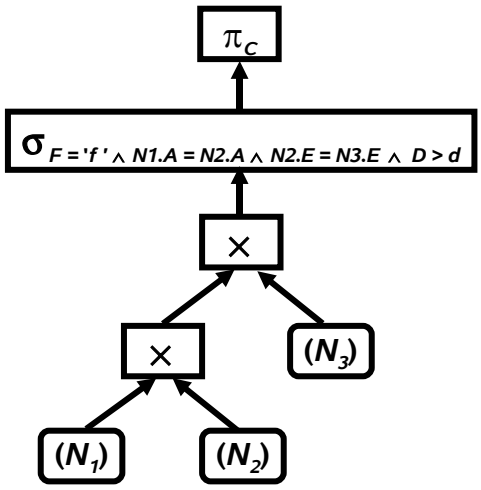
- Whereas declarative query languages (including SQL) offer a great comfort for users, they place a considerable burden on a Query Processor.
- A Query Processor is responsible to produce an execution plan that will guarantee an acceptable response time.
- Choosing a query execution plan is called Query Optimization and it mainly means making decisions about data access methods.
- For each SQL query, query parser generates an initial query tree of logical operators.
- It is also called the canonical query and it is not optimized.
- In most cases, direct execution of a canonical query would be very inefficient.

Let's take a Problem:
Consider the following relational schema

N1 ({A, B, C, D }, {A }),
N2 ({A, E, Q }, {AE }),
N3 ({E, F, G }, {E, F })

and the SQL query

```
SELECT C
FROM N1, N2, N3
WHERE F = 'f ' AND N1.A = N2.A AND N2.E = N3.E AND D > d ;
```



Generally This is the initial structure of the Query Tree

- A query tree of logical operators is a binary tree
- The nodes of that tree are logical (relational algebra) operators
- Lower level nodes, starting from leaves, contain Cartesian product operators
- These are applied onto relations from the SQL FROM clause
- After that are (relational) select and join conditions from SQL WHERE clause to upper tree nodes applied
- Finally is project operator of the SELECT clause

Now Analyze the previous Question

Question
Suppose each of r (N1), r (N2), and r (N3) has 1000 tuples

How many tuples will have the intermediate query result after the two Cartesian products?
Answers:

1. Thousand
2. Million
3. Billion

Question
How many N3 tuples are asked by the example query:

1. Many
2. Only God knows
3. At most one

Question
What is the rational behind an attempt to apply a restrictive select operation as early as possible:

- All further operations will use less tuples and thus perform faster
- The result will be more accurate
- The optimization will be more complex to understand

o Performing project operations as early as possible brings an improvement in the efficiency of a query execution Because tuples get shorter after projection, the query execution will be faster
o Heuristic optimization converts a declarative query to a canonical algebraic query tree, that is then gradually transformed using certain rules
o The main heuristics is to perform unary relational operations (selection and projection) before binary operations (joins, set theoretic), and aggregate functions with (or without) grouping

Solution

- According to the structure of the initial query tree, two Cartesian products should be executed first
- But this query asks for only a few tuples from r (N1) (D > d), and even for at most one tuple from r (N3) (F = 'f ')
- The main heuristic rule is to apply unary operations select and project before binary operations like join and set theoretic operations, and before aggregate functions
- Hence, move select operations down the tree

- Further improvement can be achieved by replacing each Cartesian product followed by a select according to a join condition with a join operator

- Next improvement can be achieved by switching the positions of N1 and
- N3, so that the very restrictive select operation
- σF = 'f '
- could be applied as early as possible

- Final improvement can be achieved by keeping in intermediate relations only the attributes needed by subsequent operations
- This can be accomplished by applying defined, or even introducing new - undefined (but logically implied) project (π) operations as early as possible