Jaspreet Singh

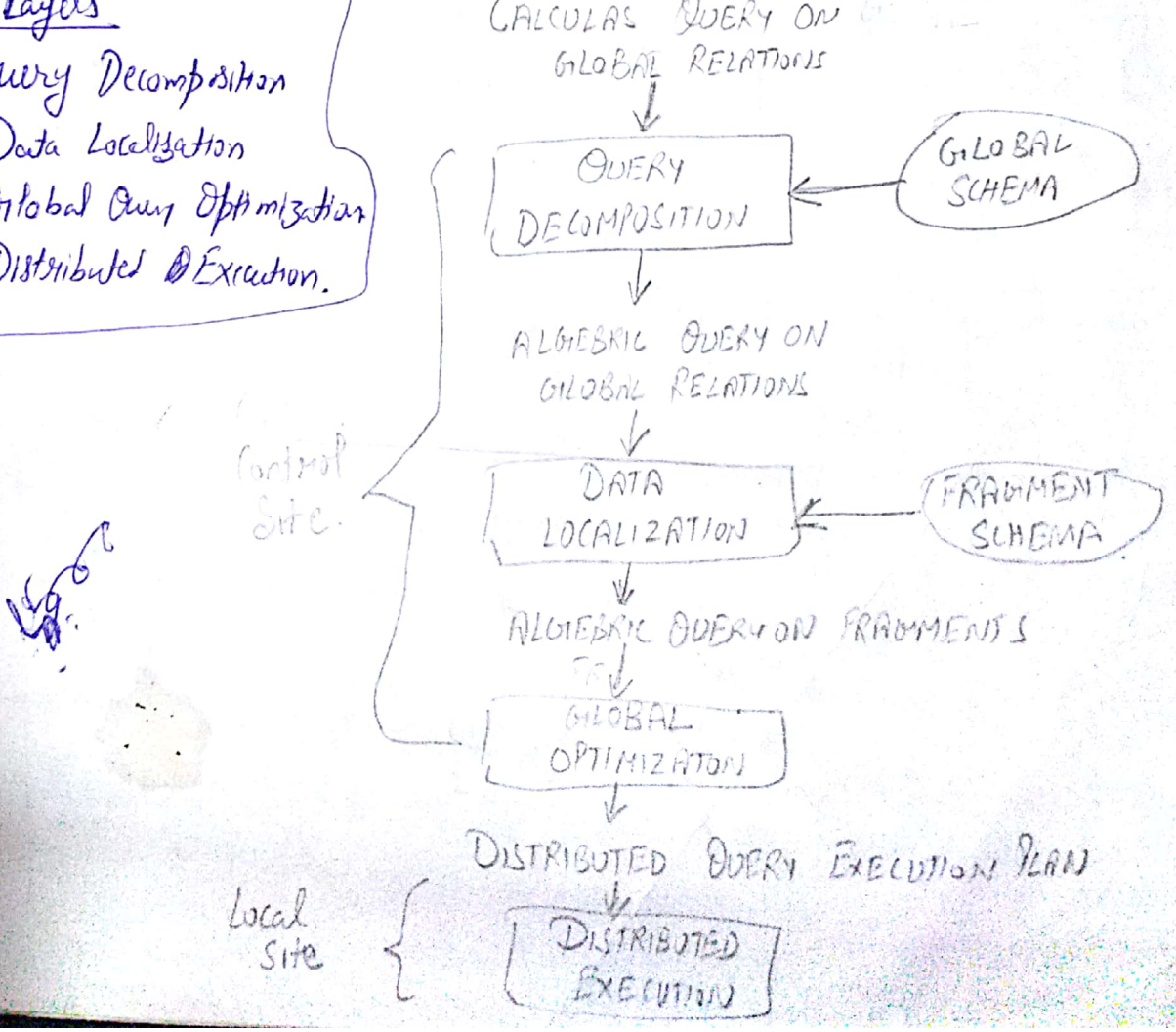
Chapter ⇒ Query Processing

Layers of Query Processing.

1. Processing of query in distributed DBMS instead of Local (centralized) DBMS.
2. is very difficult, because there are many (elements) factors parameters involves in it which affect the overall performance of distributed queries.
3. That is why query processing problem is divided into several subproblems / layers, which are easier to solve individually.
4. Main function of Query Processor is to transform a high level query (calculus) into an equivalent low-level query (algebra).

Main Layers of Query Processing

- 4 Layers
1) Query Decomposition
2) Data Localization
3) Global Query Optimization
4) Distributed Execution.



- The input is a query on global data expressed in relational calculus.
- The first three layers convert the input query into low-level query (algebric query) and map it into optimized distributed execution plan.
- The first three layers are performed by a central control site. and use Schema, metadata. information stored in global directory.
- Fourth layer performs distributed Query execution by executing the plan and return the answer to the query it is done by local sites..

Layers

1 **Query Decomposition**

- The first layer decomposes the calculus query into an algebric query on global. The information needed for this transformation is found in Global Schema.
- Query decomposition can be viewed as four successive steps.

1] Normalization → (CNF) Conjuctive Normal Form
→ (DNF) Disjunctive Normal Form. [Rewrite Query into Normal Form]

2] Analysis → [Normalized Query is analyzed semantically]

3] Elimination of Redundancy → [Simplified the query by eleminating redundant data]

4] Rewriting → [The calculus query is re constructed as an algebric query
Several algebric queries can be derived from the same calculus query

2] Data Localisation

- Output of first layer is an algebric query on global relation which is input to the second layer.
- The main role of this layer is to localize the query's data using data distribution information. (Fragment Schema).
- We know that each fragment is stored at different site.

⇒ This layer determines which fragments are involved in the query and transforms the ~~distributed~~ query into fragment query.

3] Global Query Optimization

- The input of the third layer is a fragment algebric query.
- The goal of this layer is to find an execution strategy for the algebric fragment query which is closer to optimal.
- Query Optimization consists of:
  1] Finding the best ordering of operations in the fragment query.
  2] Find the communication operations which minimizes a cost function.

4] Distributed Execution

- The last layer is performed by all the sites having fragments involved in the query.
- Each subquery called a local query, is executing at one site using local scheme.

1 Query Decomposition :

1 Normalization

- The input query may be arbitrary complex
- Transform the query to a normalized form to facilitate further processing.

⇒ • With relational languages such as SQL, The most important transformation is that of query qualification (Where clause)

There are to possible normal forms.

1 Conjuctive NF
giving precedence to AND (∧)

Ex $(P_{11} \vee P_{12} \vee \dots \vee P_{1n}) \wedge \dots \wedge (P_{m1} \vee P_{m2} \wedge \dots \vee P_{mn})$

Removing Redundancy

2) Disjunctive NF
give Precedence to (∨) OR.

$(P_{11} \wedge P_{12} \wedge \dots \wedge P_{1n}) \vee \dots \vee (P_{m1} \wedge P_{m2} \wedge \dots \wedge P_{mn})$

In disjunctive NF, the query can be processed as independent conjuctive subqueries, linked by OR

The transformation of predicates using well known equivalence rules for logical operations ($\wedge$, $\vee$ and $\neg$);

1) $P_1 \wedge P_2 \Leftrightarrow P_2 \wedge P_1$ — Commutative laws
2) $P_1 \vee P_2 \Leftrightarrow P_2 \vee P_1$
3) $P_1 \wedge (P_2 \wedge P_3) \Leftrightarrow (P_1 \wedge P_2) \wedge P_3$ — Associative law
4) $P_1 \vee (P_2 \vee P_3) \Leftrightarrow (P_1 \vee P_2) \vee P_3$
5) $P_1 \wedge (P_2 \vee P_3) \Leftrightarrow (P_1 \wedge P_2) \vee (P_1 \wedge P_3)$ — Distributed laws
6) $P_1 \vee (P_2 \wedge P_3) \Leftrightarrow (P_1 \vee P_2) \wedge (P_1 \vee P_3)$
7) $\neg (P_1 \wedge P_2) \Leftrightarrow \neg P_1 \vee \neg P_2$ — Negation
8) $\neg (P_1 \vee P_2) \Leftrightarrow \neg P_1 \wedge \neg P_2$
9) $\neg (\neg P) \Leftrightarrow P$

Example : Find the names of employees who have been working on Project P_1 for 12 or 24 months.

- SQL Query :-

```
SELECT  ENAME
FROM    EMP, ASGT
WHERE   EMP.ENO = ASGT.ENO
AND     ASGT.PNO = "P1"
AND     DUR = 12 OR DUR = 24
```

ASG1, ASG2.

Emp(Eno, ename, title)
Asg(Eno, Pno, resp, dur)
Proj(Pno, Pname, budget)

1 CNF

The qualification in CNF:

EMP.ENO = ASG.ENO ∧ ASG.PNO = "P1" ∧ DUR = 12 ∨ DUR = 24)

2 DISJUNCTIVE NF (DNF)

The qualification in DNF

(EMP.ENO = ASG.ENO ∧ ASG.PNO = "P1" ∧ DUR = 12) ∨
(EMP.ENO = ASG.ENO ∧ ASG.PNO = "P1" ∧ DUR = 24)

(X)

Step 2: Analysis

- Query Analysis enables rejection of normalized queries for which further processing is either impossible or unnecessary.
- The main reason for rejection of query (type incorrect) Syntactically or ① ② semantically incorrect.

② A query is semantically incorrect if its component do not contribute to generation of the result.

Ex ①

① Type incorrect means

Ex ②

```
SELECT E#
FROM EMP
WHERE ENAME > 200
```

① attribute E# is not declared in Scheme

② Operation >200 is incompatible with the type string of Ename

Example (2)

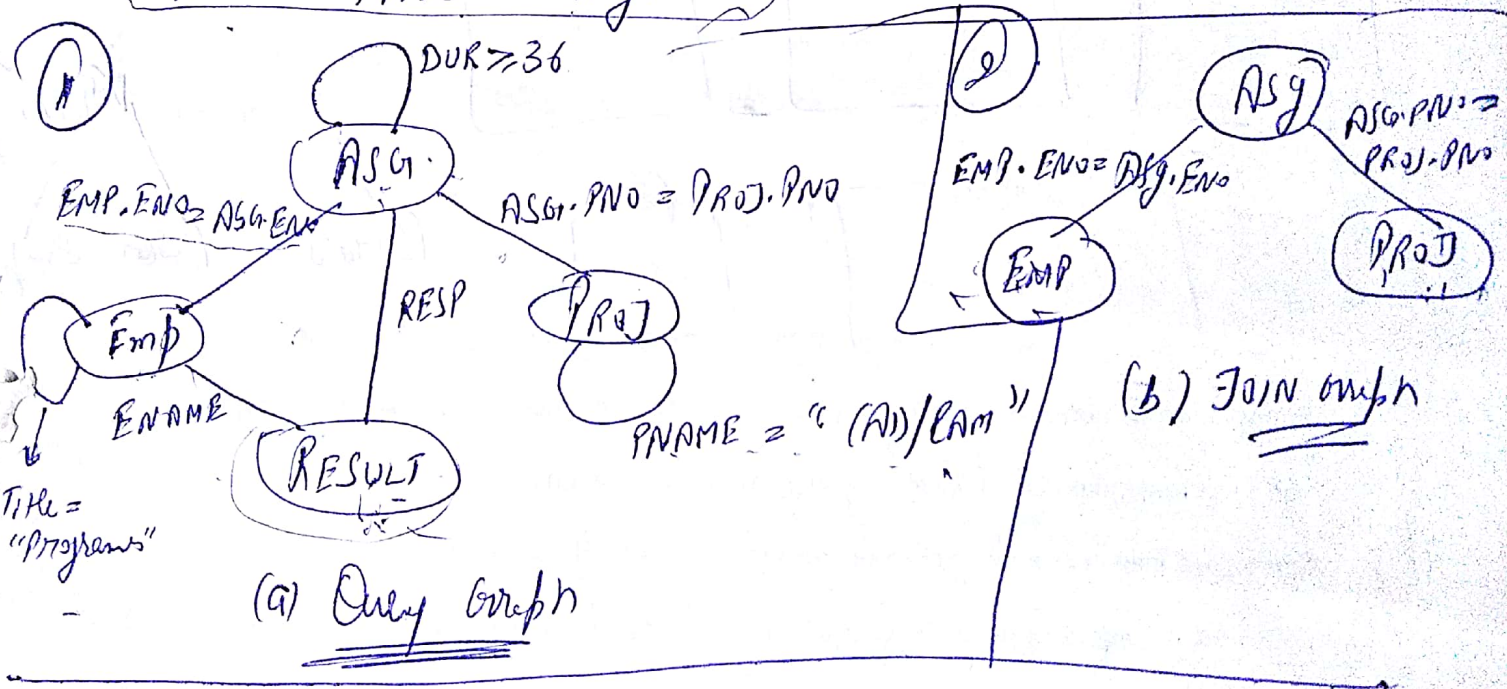
Semantically Check → (1) Query Graph
→ (2) Join graph

Find name, responsibility of programmer who have been working on CAD/CAM project for more than 3 years.

Example

```
SELECT  ENAME, RESP
FROM    EMP, ASG, PROJ
WHERE   EMP.ENO = ASG.ENO
AND     ASG.PNO = PROJ.PNO
AND     PNAME   = "CAD/CAM"
AND     DUR >= 36
AND     TITLE = "Programmer"
```

Asg (PNO, E, DUR)



- In Query graph (connection graph) one node indicates result relation. and any other node ~~skip~~ indicates operand relation.
- Query is semantically incorrect if any node does not contribute in result
- In general join predicate is missing ~~NO PROG~~ and the query should be rejected.

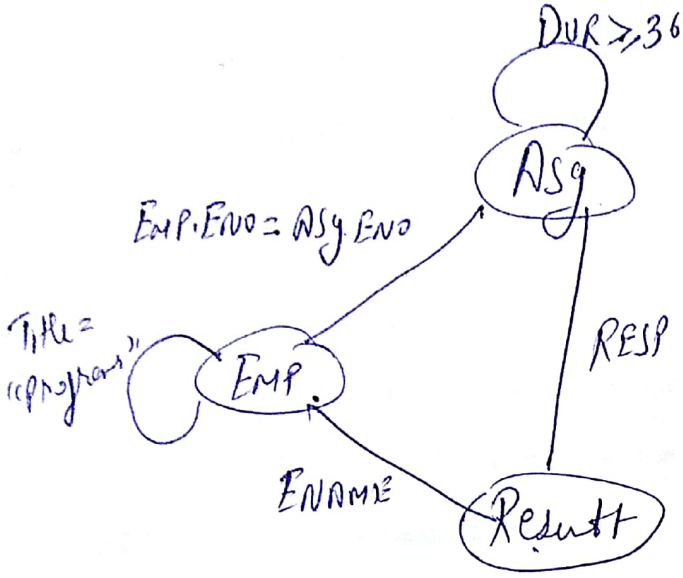
P1 → Proj
Proj etc.
J. Due P3

Example:

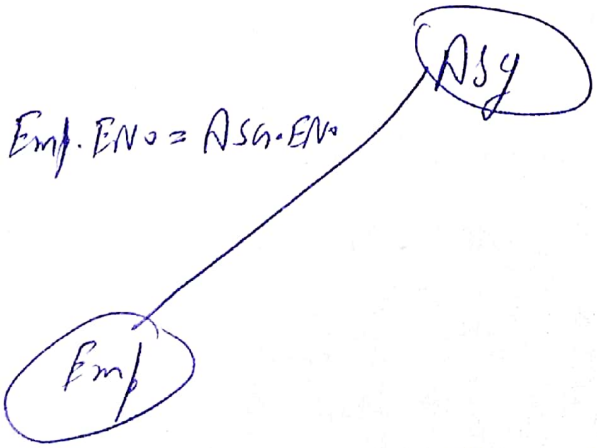
```
SELECT  ENAME, RESP
FROM    EMP, ASG, PROJ
WHERE   EMP.ENO = ASG.ENO
AND     PNAME   = "(A)/(AM"
AND     DUR ≥ 36
AND     TITLE = "Programm."
```

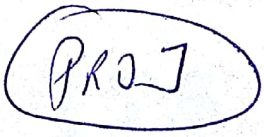
( )

$(\neg(P1) \wedge (P1 \vee P2) \wedge \neg(P2)) \vee P3$



Disconnected Query Graph
means query semantically Incorrect





Join Graph

Elimination of Redundancy

- Simplify the Query by eliminating Redundancy

By using following idempotency rules:-

1. $P \wedge P \Leftrightarrow P$
2. $P \vee P \Leftrightarrow P$
3. $P \wedge True \Leftrightarrow P$
4. $P \wedge False \Leftrightarrow false$
5. $P \vee true \Leftrightarrow true.$
6. $P \vee False \Leftrightarrow P$
7. $P \wedge \neg P \Leftrightarrow False$
8. $P \vee \neg P \Leftrightarrow true$
9. $P_1 \wedge (P_1 \vee P_2) \Rightarrow P_1$
10. $P_1 \vee (P_1 \wedge P_2) \Rightarrow P_1$

Eg

```
SELECT  TITLE
FROM    EMP
WHERE   (NOT (TITLE = "Programmer")
AND          (TITLE = "Elect. Engg. Programmer"
OR           TITLE = "ERECT Engineer")
AND     NOT ( TITLE = "Elect Engineer"))
OR      ENAME = "J. Doe"
```

Can be simplified

```
SELECT  TITLE
FROM    EMP
WHERE   ENAME = "J. Doe"
```

Simplification:

Let:

$P_1 \Leftrightarrow$ Title = "Programmer"

$P_2 \Leftrightarrow$ Title = "Elect. Eng"

$P_3 \Leftrightarrow$ Ename = "J. Doe"

$\Rightarrow (\neg P_1 \wedge (P_1 \vee P_2) \wedge \neg P_2) \vee P_3$

Disjunctive Normal Form for this qualification is obtained by applying rule 5 of Section 1 (Normalization)

Step 1 $(\neg P_1 \wedge ((P_1 \wedge \neg P_2) \vee (P_2 \wedge \neg P_2))) \vee P_3$

Applying rule 3 of Section 1

Step 2 $(\neg P_1 \wedge P_1 \wedge \neg P_2) \vee (\neg P_1 \wedge P_2 \wedge \neg P_2) \vee P_3$

By applying rule 7 of this Section:

Step 3 $(false \wedge \neg P_2) \vee (\neg P_1 \wedge false) \vee P_3$

By applying same rule, we get

Step 4 $false \vee false \vee P_3$

which is equivalent to P_3 by rule 4

Step 5 P_3

Eg → $(\neg P_1 \wedge (P_1 \vee P_2) \wedge \neg P_2) \vee P_3$

Skpl → Rule 5

$$P_1 \wedge (P_2 \vee P_3) \Leftrightarrow (P_1 \wedge P_2) \vee (P_1 \wedge P_3)$$

Solve $(\neg P_1 \wedge (P_1 \vee P_2) \wedge \neg P_2) \vee P_3$

$P_2 \quad P_3 \quad P_1$

$\Downarrow$

$\neg P_2 \wedge (P_1 \vee P_2)$

$\Downarrow$

$(\neg P_2 \wedge P_1) \vee (\neg P_2 \wedge P_2)$

$\Downarrow$

$(P_1 \wedge \neg P_2) \vee (P_2 \wedge \neg P_2)$

Soln $(\neg P_1 \wedge ((P_1 \wedge \neg P_2) \vee (P_2 \wedge \neg P_2))) \vee P_3$

Step 2

Rule 3

$P_1 \wedge (P_2 \wedge P_3) \Leftrightarrow (P_1 \wedge P_2) \wedge P_3$

$(\neg P_1 \wedge (P_1 \wedge \neg P_2) \vee \neg P_1 \wedge (P_2 \wedge \neg P_2)) \vee P_3$

Soln $((\neg P_1 \wedge P_1 \wedge \neg P_2) \vee (\neg P_1 \wedge P_2 \wedge \neg P_2)) \vee P_3$

4 Rewriting

The last step of Query Decomposition is rewrite the query in relational algebra.

For the sake of clarity, represent the relational algebra query graphically by operator tree.

Root ⟺ (Answer of query)

Non leaf nodes ⟺ conditions (where clause)

Leaf nodes ⟺ Relations (Tables) in database

Operator Tree

Example

```
SELECT ENAME
FROM PROJ, ASG, EMP
WHERE ASG.ENO = EMP.ENO
AND ASG.PNO = PROJ.PNO
AND ENAME != "J. Doe"
AND PROJ.PNAME = "CAD/CAM"
AND (DUR = 12 OR DUR = 24)
```