Optimization

Unit II continued

Query Optimization

- The query optimization problem in large-scale distributed databases is **NP-hard** in nature and difficult to solve.
- The complexity of the optimizer increases as the number of relations and number of joins in a query increases.
- A Distributed Database is a collection interrelated database distributed over network so as to improve the of logically a computer performance, reliability, availability and modularity of the distributed systems.
- Query processing is much more difficult in Distributed Environments than in Centralized Environments

Distributed Query Processing

- Since the data is geographically distributed onto multiple sites, the processing of query involves transmission of data among different sites.
- The retrieval of data from different sites is known as Distributed Query Processing (DQP).
- The query processor selects data from databases located at multiple sites in a network and performs processing over multiple CPUs to achieve a single query result set

Phases in Distributed Query Processing

- There are three phases involved in Distributed Query Processing

1. **Local Processing Phase:** In this phase, the initial Algebraic Query specified on global relations is transformed into fragments (Data Decomposition) and made available to the respective sites for processing (Data Localization) like local selections and projections.

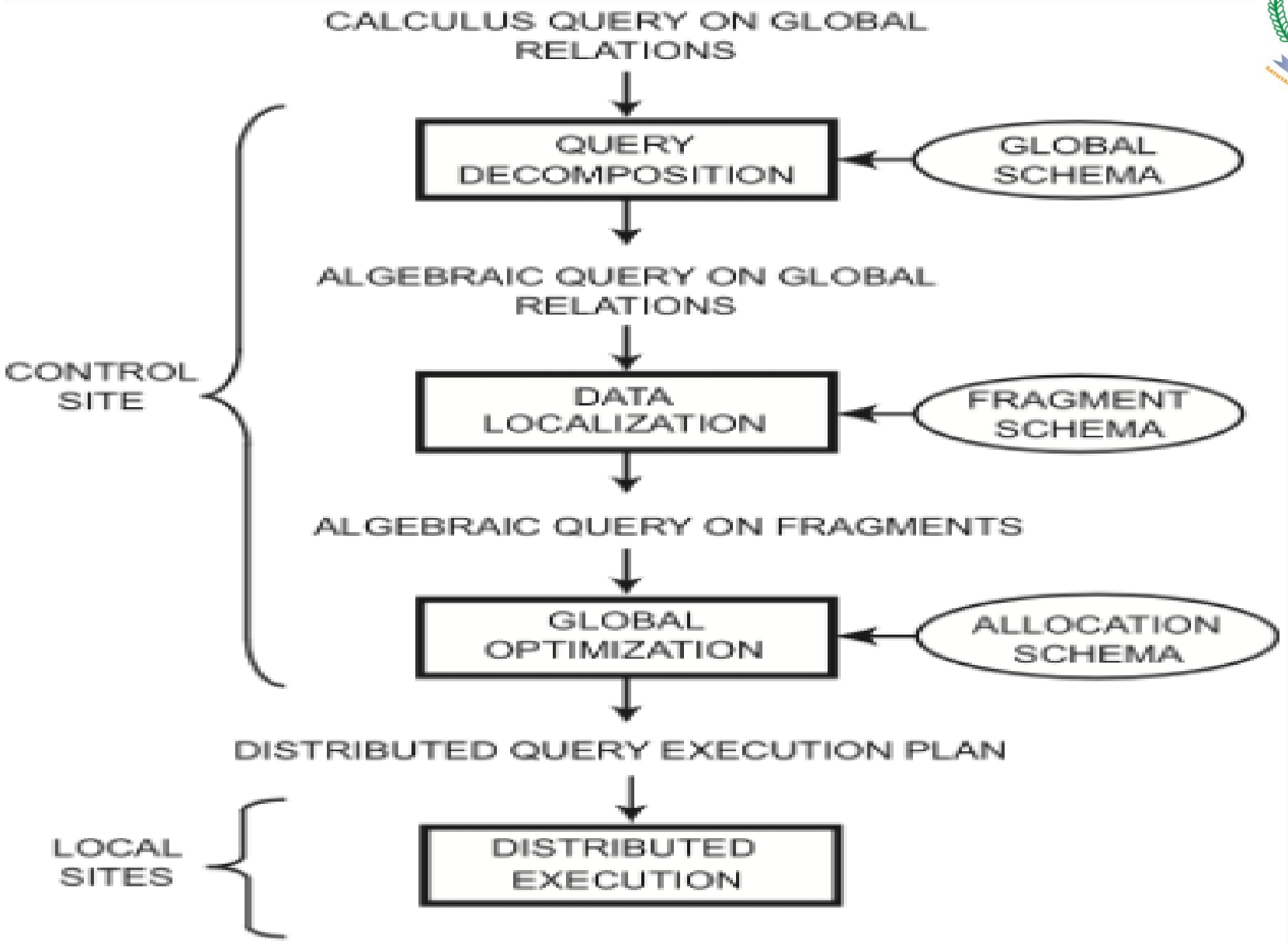
Phases in Distributed Query Processing

2. **Reduction Phase**: A sequence of Joins and Semi Joins (reducers) are used to minimize the amount of data i.e. the size of the relations that needs to be transmitted in order to accomplish a join operation in a cost effective manner.
3. **Final Processing or Assembly Phase**: In this phase all the processed files are transmitted to the assembly site for the generation of final output.

Layers of Query Processing

Four main layers are involved in distributed query processing.

- The first three layers map the input query into an optimized distributed query execution plan. They perform the functions of *query decomposition*, *data localization*, and *global query optimization*. Query decomposition and data localization correspond to query rewriting. The first three layers are performed by a central control site and use schema information stored in the global directory.
- The fourth layer performs *distributed query execution* by executing the plan and returns the answer to the query. It is done by the local sites and the control site.


CALCULUS QUERY ON GLOBAL RELATIONS
QUERY DECOMPOSITION
GLOBAL SCHEMA
ALGEBRAIC QUERY ON GLOBAL RELATIONS
CONTROL SITE
DATA LOCALIZATION
FRAGMENT SCHEMA
ALGEBRAIC QUERY ON FRAGMENTS
GLOBAL OPTIMIZATION
ALLOCATION SCHEMA
DISTRIBUTED QUERY EXECUTION PLAN
LOCAL SITES
DISTRIBUTED 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.
- However, the information about data distribution is not used here but in the next layer. Thus the techniques used by this layer are those of a centralized DBMS.

Steps in Query decomposition

- First, the calculus query is rewritten in a *normalized form* that is suitable for subsequent manipulation. Normalization is done by applying logical operator priority.
- Second, the normalized query is analyzed semantically so that incorrect queries are detected and rejected as early as possible.
- Third, the correct query (still expressed in *relational calculus*) is simplified. One way to simplify a query is to eliminate redundant predicates. Note that redundant queries are likely to arise when a query is the result of system transformations applied to the user query. Such transformations are used for performing semantic data control (views, protection, and semantic integrity control).

Steps in Query decomposition

- Fourth, the calculus query is *restructured* as an *algebraic query*. That several algebraic queries can be derived from the same calculus query, and that some algebraic queries are "better" than others. The quality of an algebraic query is defined in terms of expected performance.
- The traditional way to do this transformation toward a "better" algebraic specification is to start with an initial algebraic query and transform it in order to find a "good" one.
- The initial algebraic query is derived immediately from the calculus query by translating the predicates and the target statement into relational operators as they appear in the query.
- This directly translated algebra query is then restructured through transformation rules.
- The algebraic query generated by this layer is good in the sense that the worse executions are typically avoided.

Data Localization

- The input to the second layer is an algebraic query on global relations.
- The main role of the second layer is to localize the query's data using data distribution information in the fragment schema
- This layer determines which fragments are involved in the query and transforms the distributed query into a query on fragments.

Steps in Data Localization

- Generating a query on fragments is done in two steps.
  - ✓ First, the query is mapped into a *fragment query* by substituting each relation by its reconstruction program (also called *materialization program*).
  - ✓ Second, the fragment query is simplified and restructured to produce another "good" query. Simplification and restructuring may be done according to the same rules used in the decomposition layer.
- As in the decomposition layer, the final fragment query is generally far from optimal because information regarding fragments is not utilized.

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.
- The previous layers have already optimized the query, for example, by eliminating redundant expressions.
- This optimization is independent of fragment characteristics such as fragment allocation and cardinalities.

Elements considered for Global Query Optimization

- *Query optimization* consists of finding the "best" ordering of operators in the query, including communication operators that minimize a cost function.
- The cost function, often defined in terms of time units, refers to computing resources such as disk space, disk I/Os, buffer space, CPU cost, communication cost, and so on.
- Generally, it is a weighted combination of I/O, CPU, and communication costs.
- Nevertheless, a typical simplification made by the early distributed DBMSs, as we mentioned before, was to consider communication cost as the most significant factor.
- This used to be valid for wide area networks, where the limited bandwidth made communication much more costly than local processing. This is not true anymore today and communication cost can be lower than I/O cost.

Elements considered for Global Query Optimization

- To select the ordering of operators it is necessary to predict execution costs of alternative candidate orderings.
- Determining execution costs before query execution (i.e., static optimization) is based on fragment statistics and the formulas for estimating the cardinalities of results of relational operators.
- Thus the optimization decisions depend on the allocation of fragments and available statistics on fragments which are recorder in the allocation schema.

Query Optimization using *join*

- An important aspect of query optimization is *join ordering*, since permutations of the joins within the query may lead to improvements of orders of magnitude.
- One basic technique for optimizing a sequence of distributed join operators is through the *semijoin* operator.
- The main value of the semijoin in a distributed system is to **reduce the size** of the join operands and then the communication cost.
- However, techniques which consider local processing costs as well as communication costs may not use semijoins because they might increase local processing costs.
- The output of the query optimization layer is a optimized algebraic query with communication operators included on fragments.
- It is typically represented and saved (for future executions) as a *distributed query execution plan*.

Distributed Query Execution

- The last layer is performed by all the sites having fragments involved in the query.
- Each subquery executing at one site, called a *local query*, is then optimized using the local schema of the site and executed.
- At this time, the algorithms to perform the relational operators may be chosen.
- Local optimization uses the algorithms of centralized systems.

Summary of Distributed Query Processing

- The goal of distributed query processing may be summarized as follows:
  - Given a calculus query on a distributed database, find a corresponding execution strategy that minimizes a system cost function that includes I/O, CPU, and communication costs.
  - An execution strategy is specified in terms of relational algebra operators and communication primitives (send/receive) applied to the local databases (i.e., the relation fragments).
  - Therefore, the complexity of relational operators that affect the performance of query execution is of major importance in the design of a query processor.

Summary of Distributed Query Processing

- The performance of a distributed query is critically dependent upon the ability of the query optimizer to derive efficient query processing strategies
- Query Optimization is one of the most important and expensive stages in executing distributed queries.
- The complexity of the optimization process is determined by the number of relations referenced types of initial query access methods and the set of rules involved for generating possible query trees or query graphs.

Role of Query Optimizer

- The role of Query Optimizer is to produce Query Execution Plans (QEP) which represents an execution strategy of the query with minimum cost.
- An optimizer is a software module that performs optimization of queries on the basis of three important components of a query i.e.
    1. Search Space,
    2. Search Strategies
    3. Cost Models.

Search Space

- It refers to the generation of sets of alternative and equivalent QEPs of an input query by applying **Transformational Rules** such that they differ in the execution order of the operators .
- The QEPs are commonly referred to as Operator Trees or Join Trees whose operators are various types of Joins or Cartesian Products.
- This can be represented as a query graph (annotated tree) denoted as G = (N,A) where N is the set of nodes(vertices) in the Query Graph and A is the set of arcs (edges).

Search Space

- **Each node** represents a set of **Base File** (BF) in the join specification of the query.
- Two nodes are connected by an arc if the query joins the two corresponding files. Each node in the query graph has an associated site set.
- These **leaf nodes** represent file materializations resulting from local processing and it is a **reduced file**.
- The **root node** represents the final step where the **query result** is generated. Each parent node uses the result files of its children as its inputs.

Example

- File BF0 is stored at sites S1 and S2;
- File BF1 at site S3;
- File BF2 at site S4; and
- File BF3 at sites S3 and S4.
- Consider a query that joins a sequence of four files, referenced BF0 through BF3.
- Join attributes are designated A0 through A2 (i.e. four files are joined using three join operations).

Query Graph

- In SQL this query is specified as:

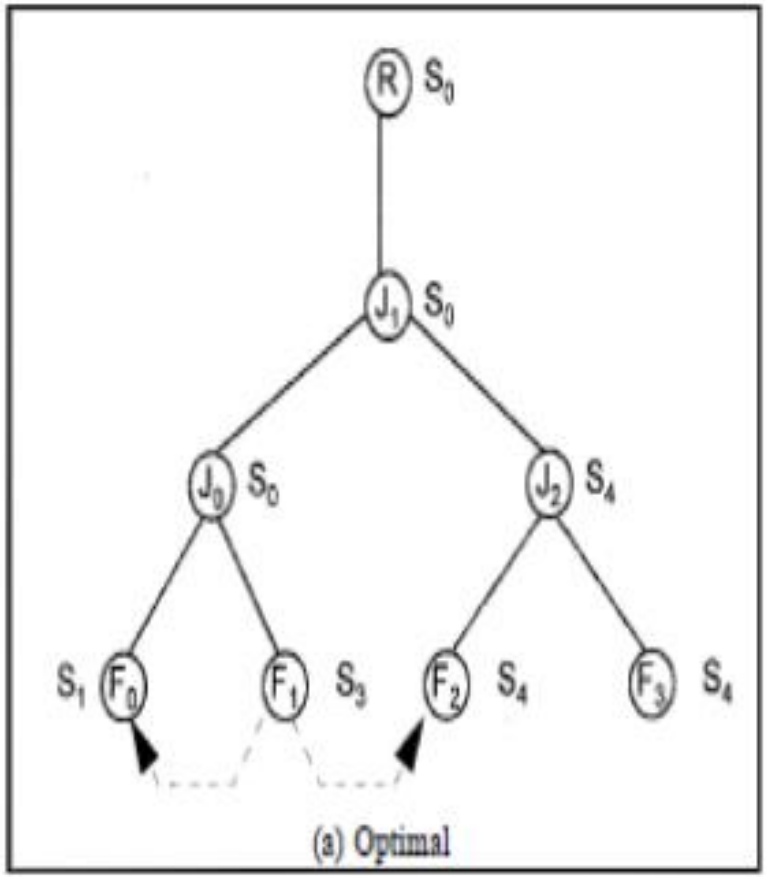
  select A1, from BF0, BF1, BF2, BF3 where BF0.A0 = BF1.A0 and BF1.A1 = BF2.A1 and BF2.A2 = BF3.A2

- The query graph for the distributed query, augmented with the site set for each file, is shown above.

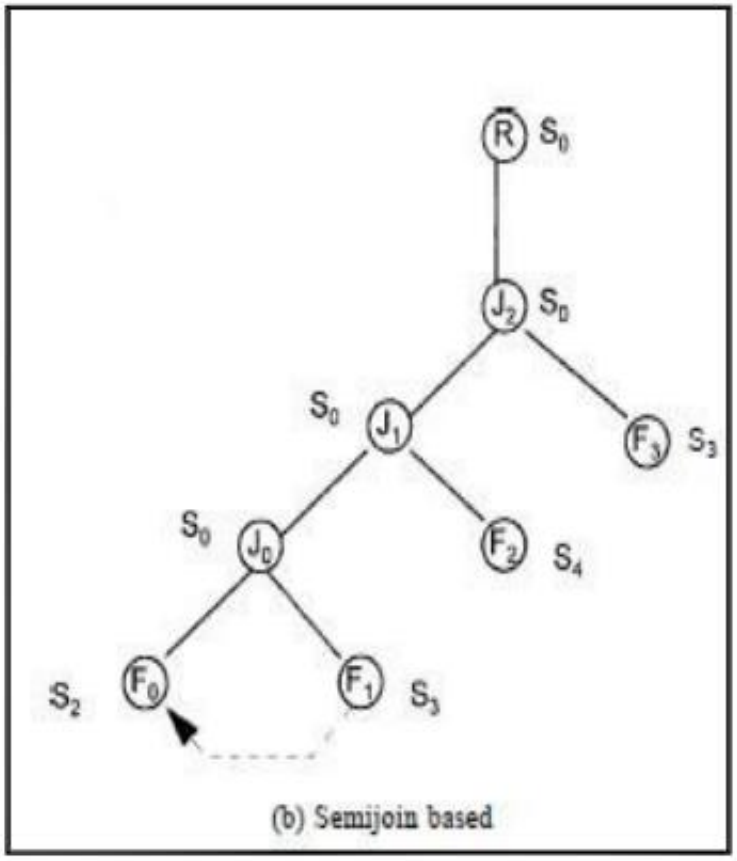
Query Graph for Distributed Query

Query Tree

- Since a query graph is a complicated query tree modeling various join operations, the QEP for the given query can be represented as follows:



(a) Optimal

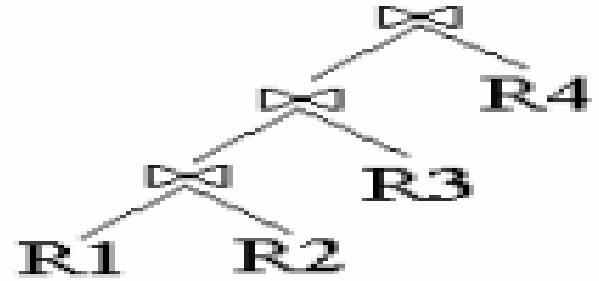


(b) Semijoin based

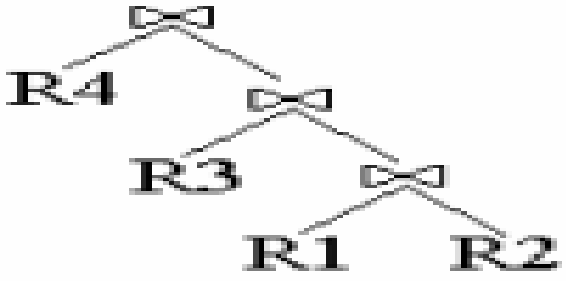
Task of Query Optimizer

- Determine the Order of Execution of Join Operation.
- Determine the Join Operation Access Methods.
- Determine the shape of the Join Query Graph in the given search space by implementing the appropriate search strategy such that the performance measure of the resulting Query Execution Plan is optimized.
- It can be Linear Trees (Left Deep Trees), Right Deep trees, Bushy Join Trees, Binary Trees or Zig-Zag
- Determine the order of data movements between the sites (Join Sites) so as to reduce the amount of data and cost on the communication network.

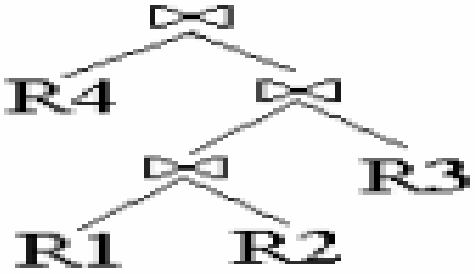
Tree Structure



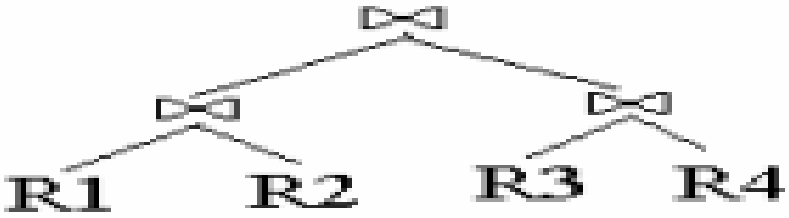
(a) Left-Deep



(b) Right-Deep



(c) Zig-Zag



(d) Bushy

Search Strategy

- It refers to the algorithms applied to explore the search space and determine the best Query Execution Plan (QEP) based on Join Selectivity and Join Sites so as to reduce the cost of query optimization.
- There are basically two classes of strategies that solve the problem of Join Scheduling for Query Optimization.

    1. Deterministic Strategy
    2. Randomized Strategies

Deterministic Strategy

- The first approach is Deterministic Strategy that proceeds by
  1. building plans
  2. starting from base relations,
  3. joining one or more relations at each step till complete plans are obtained.
  4. To reduce the optimization cost, the plans that do not lead to optimal solutions are pruned. The Dynamic Programming builds all such plans using Breadthfirst search while Greedy Algorithms uses depthfirst search.

Randomized Strategies

- Search the optimal solution around some particular points.
- These strategies do not guarantee optimal plan but they avoid high cost of optimization in terms of memory and time consumption.
- Iterative Improvement and Simulated Annealing are common solution algorithms under these strategies.

Cost Model

- The Objective of Query Optimization in Distributed Database Environment is to **minimize the total cost** of computer resources.
- An optimizer cost model includes cost functions to predict the **cost of operators and formulas** to evaluate the sizes of the results.
- The cost function can be expressed with respect to either total time or response time.
- The total time is inclusive of

  1.Local Processing Cost (CPU Time + I/O Cost),

  2.Communication Cost (Fixed time to initiate a message + Time to transmit a data).

Cost Model

- Minimizing the total time implies that the utilization of resources increases thus increasing the system throughput.
- The Response Time is evaluated as the time elapsed between initiation and completion of a query including parallelism.
- In parallel transferring, the response time is minimized by increasing the degree of parallel execution.

Cost Model

- Primarily, the cost of a query depends on the **size of intermediate relation** that are produced during execution and which must be transmitted over a network for a query operation at a different site.
- The emphasis is on the estimation of the size of the intermediate relations based on Join Orders and Join Methods so as to reduce the amount of data transfers and hence decrease the total cost and total time of the distributed query execution.

Optimization of Access Strategies

- Query Optimization process involves finding a near optimal query execution plan which represents the overall execution strategy for the query.
- The efficiency of distributed database system is significantly dependent on the extent of optimality of this execution plan.

Optimization of Access Strategies

- A good query execution strategy involves three phases.
  - First is to find a search space which is a set of alternative execution plans for query.
  - Second is to build a cost model which can compare costs of different execution plans.
  - Finally in third step we explore a search strategy to find the best possible execution plan using cost model.
- Before putting any queries to a Distributed Database, one needs to design it according to the needs of an organization.

- **Query Optimizer is responsible for this tough job of execution, plan creation, and selection**
- First it will be parsed and syntax will be verified and if passed, the "**Binding**" phase will verify the existence of base objects (tables, views, and functions).
- If everything is good during the first two phases, then the Query Optimizer's job starts.
- Query Optimizer (based on server and query configurations and table statistics - metadata of user data) will create multiple execution plans and will select an optimum performing plan based on execution cost.
- Execution cost is the total of CPU cost and IO cost.

Why optimum and why not best

Because Query Optimizer is bound to create and select a plan out of hundreds or thousands of plans, but within a fixed half second (500 milliseconds), the Optimizer can't take a whole day to test all expected plans; that's why the Query Optimizer optimum plans within half milliseconds and sends it for execution.

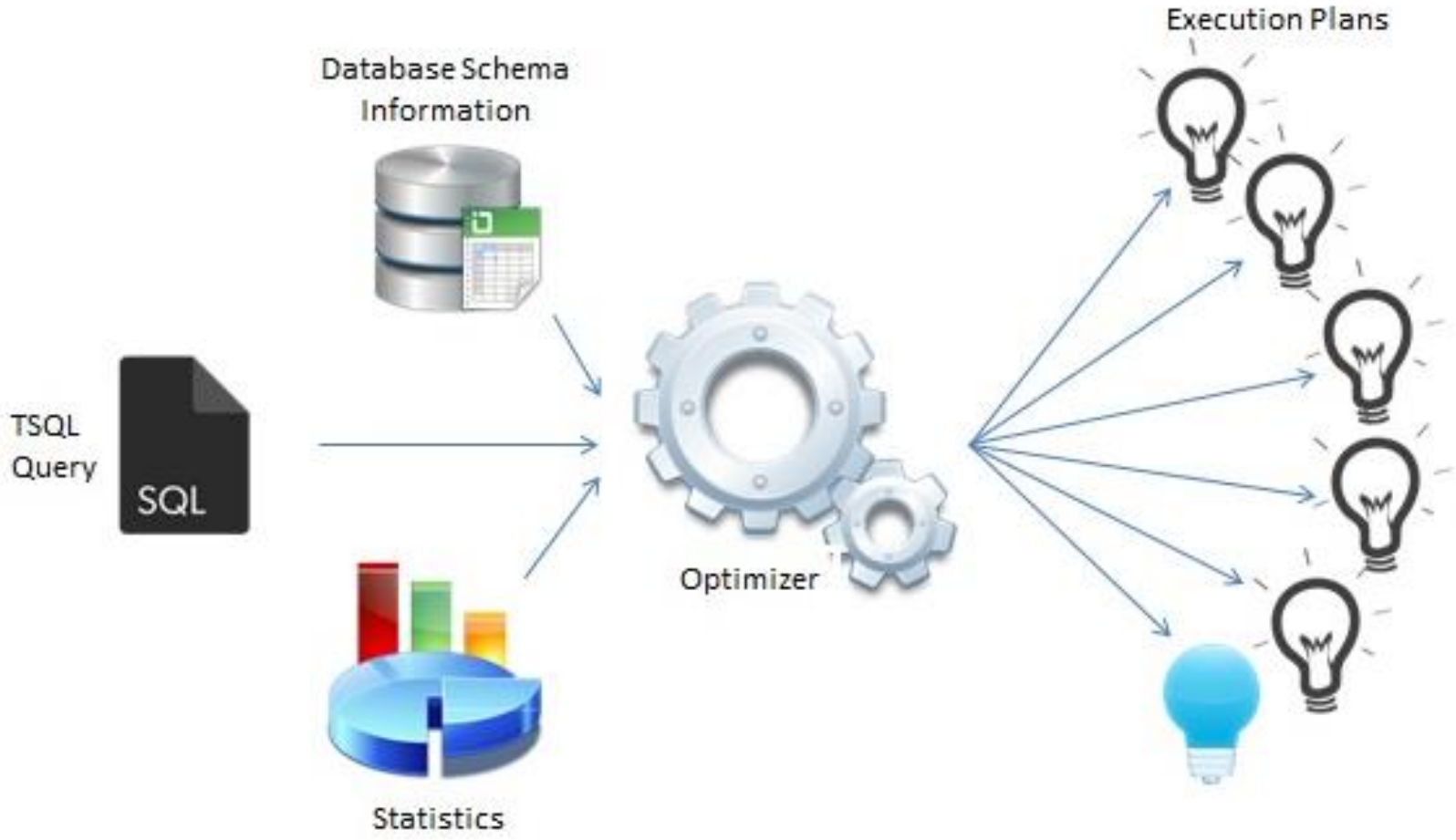
Serial Plan Execution or Parallel Plan Execution

- A query can be executed using a single CPU core, or query work can be divided and resolved through multiple CPU cores on a box.
- It's the Query Optimizer (and sometimes execution engine) that will decide whether a query will be fast on a single CPU or on multiple CPUs, and if it is good to divide on multiple CPUs, then how many CPUs will be involved.
- A query can be slow if it’s not written according to Query Optimizer, and Optimizer will be unable to divide work on multiple CPUs and will try to resolve it through only one CPU.

Query Optimizer

- Query Optimizer is designed in a way to keep a balance between the quality of a query and its execution time.
- Query Optimizer can’t take hours to evaluate all candidate plans in order to select the best one, but rather it is designed to try within a limited time and select optimum plan.

Query Optimizer



Query Optimizer

- Query Optimizer is designed in a way to keep a balance between the quality of a query and its execution time.
- Query Optimizer can’t take hours to evaluate all candidate plans in order to select the best one, but rather it is designed to try within a limited time and select optimum plan.

Query Optimizer

- To select an optimum performing plan, the most important thing for Query Optimizer is “**Join Order**."
- At a single time, only two tables can be joined together, and then a third can be joined with the output from the first two tables, or it is possible to join the two tables separately and then join the result sets.
- Ordering depends on the nature of join. But which table should be accessed first and which will be joined later -- all these possible patterns are evaluated to get best one.

Query Optimizer

- The number of expected plans (candidate plans) is directly proportional to the number of tables joined together in a query.
- Query Optimizer will always select **best** possible plan, if expected plans are very few and Query Optimizer is able to evaluate all possible plan within its limited time.
- On the other hand, if the database is over-normalized and more tables need to be joined, then there are hundreds or thousands of candidate plans and due to limited time Query Optimizer can evaluate a fewer number of plans, and there are possibilities that all evaluated plans were badly performing and Optimizer has selected the best way out of all the bad plans.

JOINS

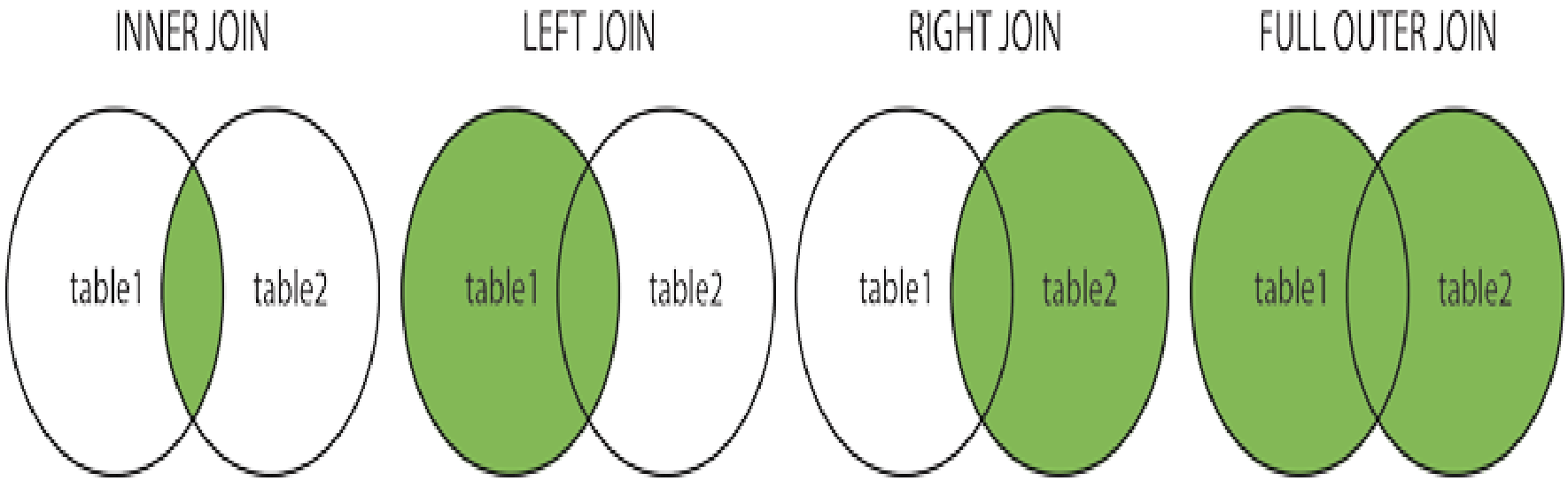
- A JOIN clause is used to combine rows from two or more tables, based on a related column between them.

Types of SQL JOINs

Here are the different types of the JOINs in SQL:

- **(INNER) JOIN**: Returns records that have matching values in both tables
- **LEFT (OUTER) JOIN**: Returns all records from the left table, and the matched records from the right table
- **RIGHT (OUTER) JOIN**: Returns all records from the right table, and the matched records from the left table
- **FULL (OUTER) JOIN**: Returns all records when there is a match in either left or right table

Types of SQL JOINs



Join Query

- Order table

OrderID	CustomerID	OrderDate

- Customer table

CustomerID	CustomerName	ContactName	Country

- SELECT Orders.OrderID, Customers.CustomerName, Orders.OrderDate
  FROM Orders
  INNER JOIN Customers ON Orders.CustomerID=Customers.CustomerID;

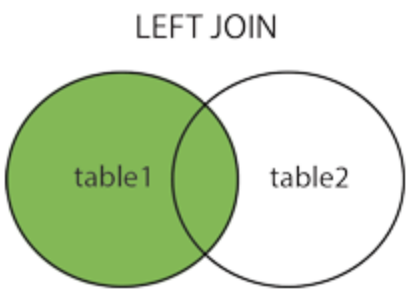
OrderID	CustomerName	OrderDate
10248	Wilman Kala	1996-07-04
10249	Tradição Hipermercados	1996-07-05
10250	Hanari Carnes	1996-07-08

SQL LEFT JOIN Keyword

The `LEFT JOIN` keyword returns all records from the left table (table1), and the matching records from the right table (table2). The result is 0 records from the right side, if there is no match.

LEFT JOIN Syntax

```
SELECT column_name(s)
FROM table1
LEFT JOIN table2
ON table1.column_name = table2.column_name;
```



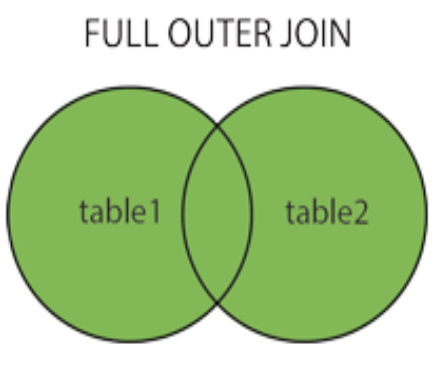
SQL FULL OUTER JOIN Keyword

The `FULL OUTER JOIN` keyword returns all records when there is a match in left (table1) or right (table2) table records.

Tip: `FULL OUTER JOIN` and `FULL JOIN` are the same.

FULL OUTER JOIN Syntax

```
SELECT column_name(s)
FROM table1
FULL OUTER JOIN table2
ON table1.column_name = table2.column_name
WHERE condition;
```



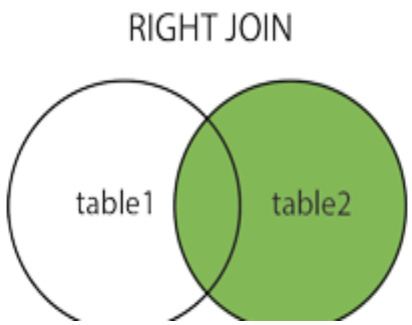
SQL RIGHT JOIN Keyword

The `RIGHT JOIN` keyword returns all records from the right table (table2), and the matching records from the left table (table1). The result is 0 records from the left side, if there is no match.

RIGHT JOIN Syntax

```
SELECT column_name(s)
FROM table1
RIGHT JOIN table2
ON table1.column_name = table2.column_name;
```

Note: In some databases `RIGHT JOIN` is called `RIGHT OUTER JOIN`.



Distributed Transaction

- A **distributed transaction** is a database transaction in which two or more network hosts are involved.
- Usually, hosts provide **transactional resources**, while the **transaction manager** is responsible for creating and managing a global transaction that encompasses all operations against such resources.
- Distributed transactions, as any other transactions, must have all four **ACID (atomicity, consistency, isolation, durability)** properties, where atomicity guarantees all-or-nothing outcomes for the unit of work (operations bundle).

Atomicity

- Atomicity guarantees that each transaction is treated as a single "unit", which either succeeds completely, or fails completely: if any of the statements constituting a transaction fails to complete, the entire transaction fails and the database is left unchanged.
- An atomic system must guarantee atomicity in each and every situation, including power failures, errors and crashes.
- A guarantee of atomicity prevents updates to the database occurring only partially, which can cause greater problems than rejecting the whole series outright.
- As a consequence, the transaction cannot be observed to be in progress by another database client. At one moment in time, it has not yet happened, and at the next it has already occurred in whole

Consistency

- Consistency ensures that a transaction can only bring the database from one valid state to another, maintaining database invariants: any data written to the database must be valid according to all defined rules, including constraints, cascades, triggers, and any combination thereof.
- This prevents database corruption by an illegal transaction, but does not guarantee that a transaction is *correct*. Referential integrity guarantees the primary key – foreign key relationship.

Isolation

- Transactions are often executed concurrently (e.g., multiple transactions reading and writing to a table at the same time). Isolation ensures that concurrent execution of transactions leaves the database in the same state that would have been obtained if the transactions were executed sequentially.
- Isolation is the main goal of concurrency control; depending on the method used, the effects of an incomplete transaction might not even be visible to other transactions.

Durability

- Durability guarantees that once a transaction has been committed, it will remain committed even in the case of a system failure (e.g., power outage or crash).
- This usually means that completed transactions (or their effects) are recorded in non-volatile memory.