Distributed Database – SCS1613

UNIT - II

QUERIES AND OPTIMAZATION

Global Queries to Fragment Queries-Equivalence Transformations for Queries-Distributed Grouping and Aggregate Function Evaluation-Parametric Queries-Optimization of Access Strategies-Framework for Query Optimization-Join Queries- General Queries-Introduction to Distributed Transactions.

Global Queries to Fragment Queries

When a query is placed, it is at first scanned, parsed and validated. An internal representation of the query is then created such as a query tree or a query graph. Then alternative execution strategies are devised for retrieving results from the database tables. The process of choosing the most appropriate execution strategy for query processing is called query optimization.

Query Optimization Issues in DDBMS

In DDBMS, query optimization is a crucial task. The complexity is high since number of alternative strategies may increase exponentially due to the following factors −

- The presence of a number of fragments.
- Distribution of the fragments or tables across various sites.
- The speed of communication links.
- Disparity in local processing capabilities.

Hence, in a distributed system, the target is often to find a good execution strategy for query processing rather than the best one. The time to execute a query is the sum of the following

- Time to communicate queries to databases.
- Time to execute local query fragments.

- Time to assemble data from different sites.
- Time to display results to the application.

Query Processing

Query processing is a set of all activities starting from query placement to displaying the results of the query. The steps are as shown in the following diagram −

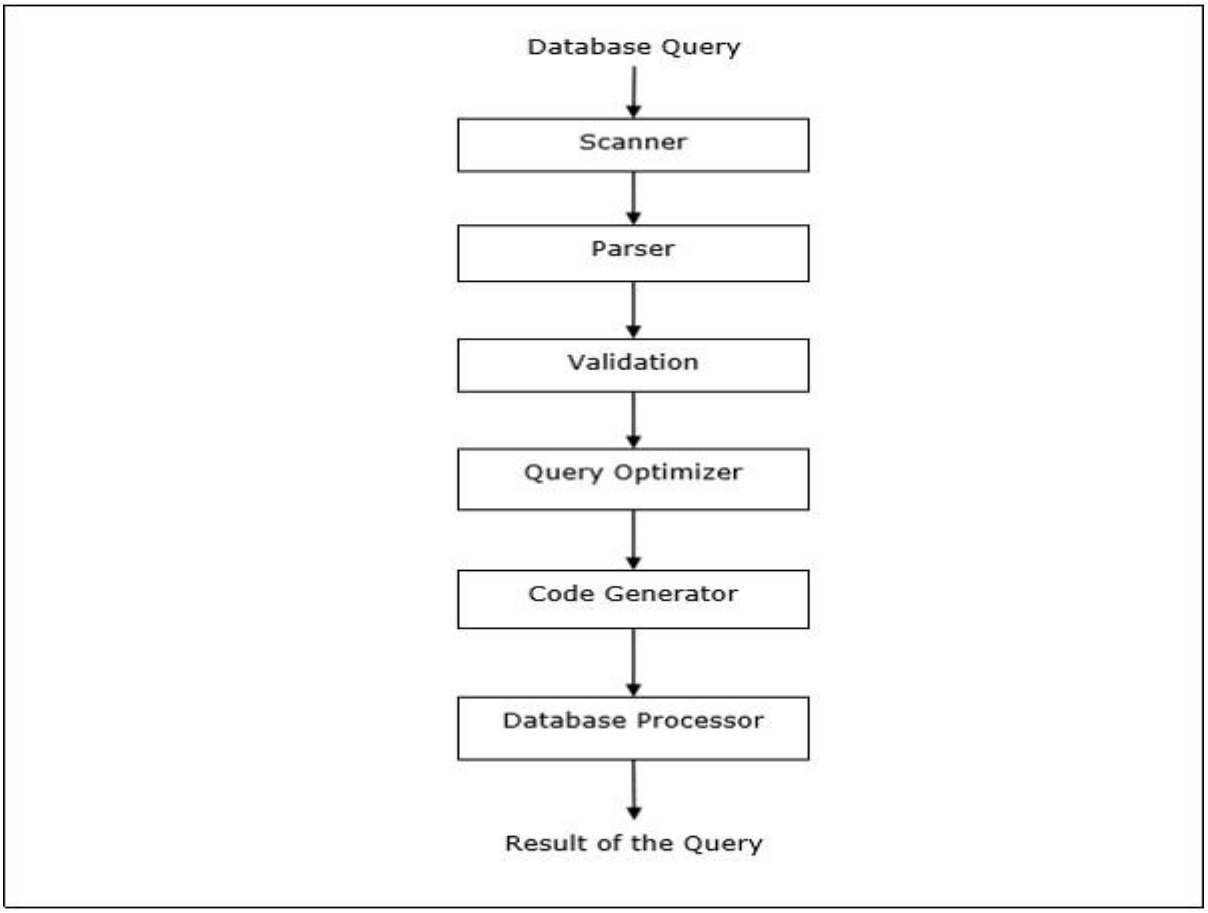

Figure 1 steps in query processing

Global Query Optimization

Input: Fragment query

- Find the *best* (not necessarily optimal) global schedule

  ➡ Minimize a cost function

➡ Distributed join processing

✦ Bushy vs. linear trees

✦ Which relation to ship where?

✦ Ship-whole vs ship-as-needed

➡ Decide on the use of semijoins

✦ Semijoin saves on communication at the expense of more local processing.

➡ Join methods

✦ nested loop vs ordered joins (merge join or hash join)

Cost-Based Optimization

- Solution space

  ➡ The set of equivalent algebra expressions (query trees).

- Cost function (in terms of time)

  ➡ I/O cost + CPU cost + communication cost

  ➡ These might have different weights in different distributed environments (LAN vs WAN).

  ➡ Can also maximize throughput

- Search algorithm

  ➡ How do we move inside the solution space?

➡ Exhaustive search, heuristic algorithms (iterative improvement, simulated annealing, genetic,…)

Query Optimization Process

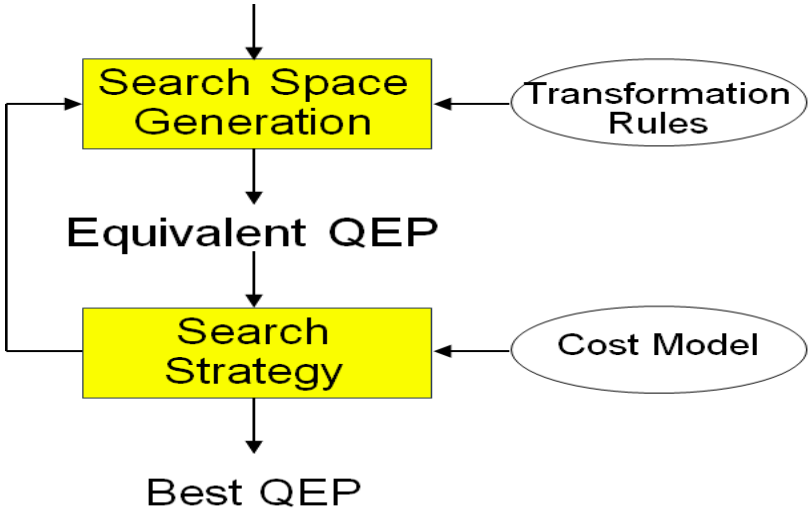

Figure 2 Query Optimization Process

Search Space

- Search space characterized by alternative execution
- Focus on join trees
- For *N* relations, there are O(*N*!) equivalent join trees that can be obtained by applying commutativity and associativity rules

SELECT ENAME,RESP

FROM EMP, ASG,PROJ

WHERE EMP.ENO=ASG.ENO

AND ASG.PNO=PROJ.PNO

Cost Functions

- Total Time (or Total Cost)

➡ Reduce each cost (in terms of time) component individually

➡ Do as little of each cost component as possible

➡ Optimizes the utilization of the resources

Increases system throughput

- Response Time

  ➡ Do as many things as possible in parallel

  ➡ May increase total time because of increased total activity

- Summation of all cost factors
- Total cost = CPU cost + I/O cost + communication cost
- CPU cost = unit instruction cost * no.of instructions
- I/O cost = unit disk I/O cost * no. of disk I/Os
- communication cost = message initiation + transmission

2-Step – Problem Definition

- Given

  ➡ A set of sites $S = \{s_1, s_2, \ldots, s_n\}$ with the load of each site

  ➡ A query $Q = \{q_1, q_2, q_3, q_4\}$ such that each subquery q_i is the maximum processing unit that accesses one relation and communicates with its neighboring queries

  ➡ For each q_i in Q, a feasible allocation set of sites $S_q = \{s_1, s_2, \ldots, s_k\}$ where each site stores a copy of the relation in q_i

- The objective is to find an optimal allocation of Q to S such that

➡ the load unbalance of S is minimized

➡ The total communication cost is minimized

- For each q in Q compute load (S_q)
- While Q not empty do

➡ Select subquery$a$ with least allocation flexibility

➡ Select best site b for$a$ (with least load and best benefit)

➡ Remove a from Q and recompute loads if needed

2-Step Algorithm Example

- Let $Q = \{q_1, q_2, q_3, q_4\}$ where q_1 is associated with R_1, q_2 is associated with R_2 joined with the result of q_1, etc.
- Iteration 1: select q_4, allocate to s_1, set load(s_1)=2
- Iteration 2: select q_2, allocate to s_2, set load(s_2)=3
- Iteration 3: select q_3, allocate to s_1, set load(s_1) =3
- Iteration 4: select q_1, allocate to s_3 or s_4

Relational Algebra :

- The Relational Algebra is used to define the ways in which relations (tables) can be operated to manipulate their data.
- This Algebra is composed of Unary operations (involving a single table) and Binary operations (involving multiple tables).
- Join, Semi-join these are Binary operations in Relational Algebra.

Join

- Join is a binary operation in Relational Algebra.
- It combines records from two or more tables in a database.

- A join is a means for combining fields from two tables by using values common to each.

Semi-Join

•A Join where the result only contains the columns from one of the joined tables.

•Useful in distributed databases, so we don't have to send as much data over the network. •Can dramatically speed up certain classes of queries.

What is "Semi-Join" ?

Semi-join strategies are technique for query processing in distributed database systems. Used for reducing communication cost.

A semi-join between two tables returns rows from the first table where one or more matches are found in the second table.

The difference between a semi-join and a conventional join is that rows in the first table will be returned at most once. Even if the second table contains two matches for a row in the first table, only one copy of the row will be returned.

Semi-joins are written using EXISTS or IN.

A Simple Semi-Join Example "Give a list of departments with at least one employee." Query written with a conventional join:

SELECT D.deptno, D.dname FROM dept D, emp E WHERE E.deptno = D.deptno ORDER BY D.deptno;

◦ A department with N employees will appear in the list N times.

◦ We could use a DISTINCT keyword to get each department to appear only once.

A Simple Semi-Join Example "Give a list of departments with at least one employee." Query written with a semi-join:

SELECT D.deptno, D.dname FROM dept D WHERE EXISTS (SELECT 1 FROM emp E WHERE E.deptno = D.deptno) ORDER BY D.deptno;

◦ No department appears more than once.

◦ Oracle stops processing each department as soon as the first employee in that department is found.