Joins

A join can be defined as a cross-product followed by selection and projection.

Joins can be implemented through the following methods.

1. Simple Nested Loop Join
2. Block Nested Loop Join.
3. Index Nested Loop Join.
4. Sort-Merge Join.

Simple Nested Loop Join :-

* The simplest join algorithm is a tuple-at-a-time nested loops evaluation.

Algorithm

```
for each tuple r ∈ R do
    for each tuple s ∈ S do
        if r_i == s_j then
            add <r, s> to result.
```

Block Nested loop Join :-

The Simple Nested loop Join does not effectively utilize buffer pages.

Suppose we have enough memory to hold the smaller relation, say R, with atleast two extra buffer pages left over.

We can read in the smaller relation and use one of the extra buffer pages to scan the larger relation S.

For each tuple $s \in S$, we check R and output a tuple $\langle r, s \rangle$ for qualifying tuples S.

The second extra buffer page is used an output buffer.

Algorithm

```
for each block of B-2 pages of R do
    for each page of S do
    {
        for all matching in-memory tuples
        r ∈ R-block and s ∈ S-page
            add <r,s> to result
    }
```

```
Tr = first tuple in R;
Ts = first tuple in S;
Gs = first tuple in S;

while Tr ≠ eof and Gs ≠ eof do
{
    while Tr_i < Gs_j do
        Tr = next tuple in R after Tr

    while Tr_i > Gs_j do
        Gs = next tuple in S after Gs

    Ts = Gs;
    while Tr_i = Gs_j do
    {
        Ts = Gs;
        while Ts_j == Tr_i do
        {
            add <Tr, Ts> to result;
            Ts = next tuple in S after Ts;
        }
        Tr = next tuple in R after Tr;
    }
    Gs = Ts;
}
```

Indexed Nested Loop Join

If there is an index on one of the relations on the join attribute(s), we can take advantage of the index by making the indexed relation be the inner relation. suppose we have a suitable index on S

Algorithm

for each tuple $r \in R$ do
    for each tuple $s \in S$ where $r_i = s_j$
        add $\langle r, s \rangle$ to result.

Sort-Merge Join

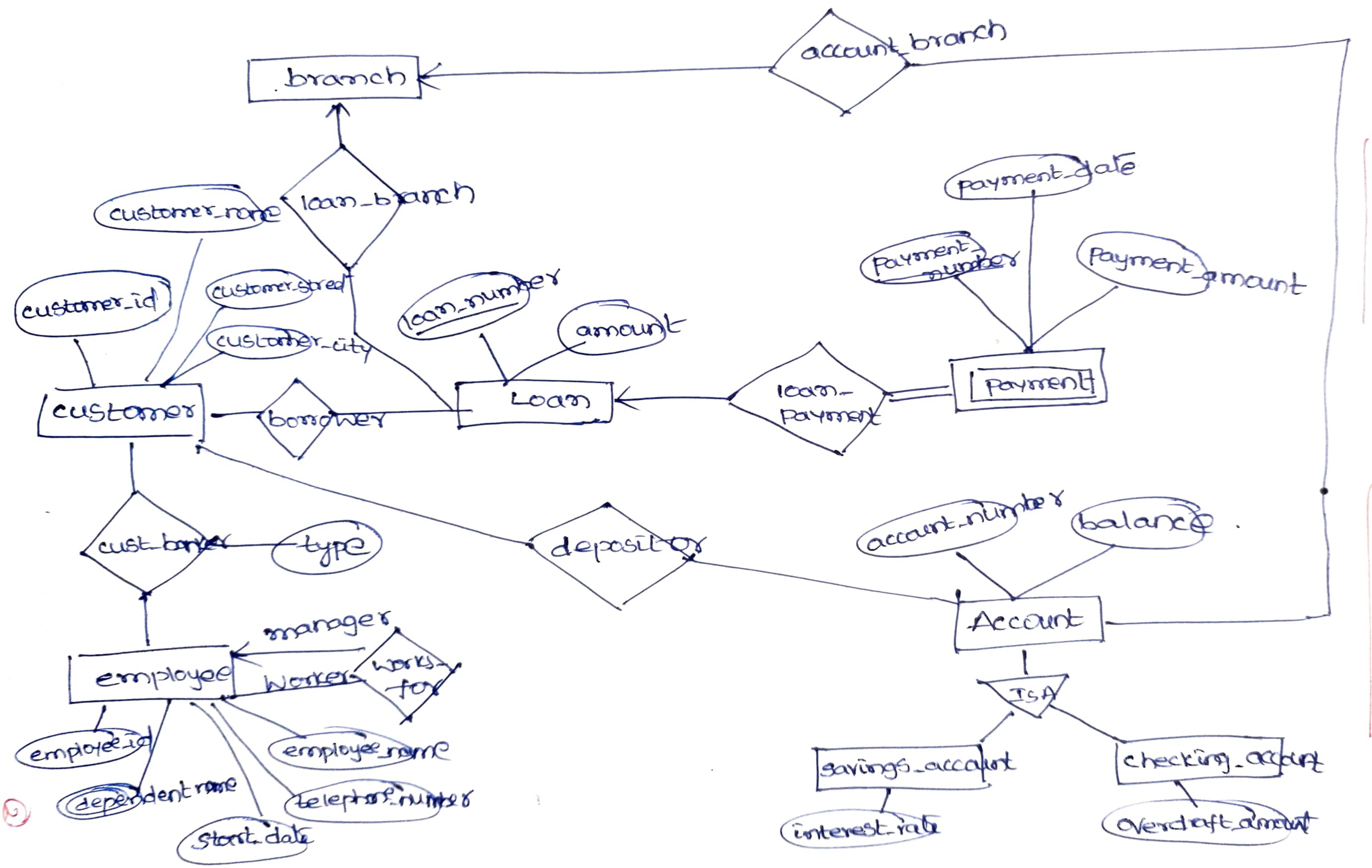
The basic idea behind the sort-merge join algorithm is to sort both relations on the join attribute and then look for qualifying tuples $r \in R$ and $s \in S$ by essentially mapping the two relations.

Algorithm :

sort_merge_join(R, S)

If R is not sorted on attribute i, sort it;
If S is not sorted on attribute j, sort it

E-R Diagram for banking enterprise.



Query optimization

Query optimization is the overall process of choosing the most efficient means of executing a SQL statement.

The components involved in query optimization are:-

1. Query Parser
2. Query optimizer
3. Query plan Evaluator.

1. **Query parser:-** It parses the given query.

2. **Query optimizer:-** It comprises of two components.
    1. Query plan Generator
    2. Query plan cost ~~Gene~~ Estimator.

Query plan Generator:-

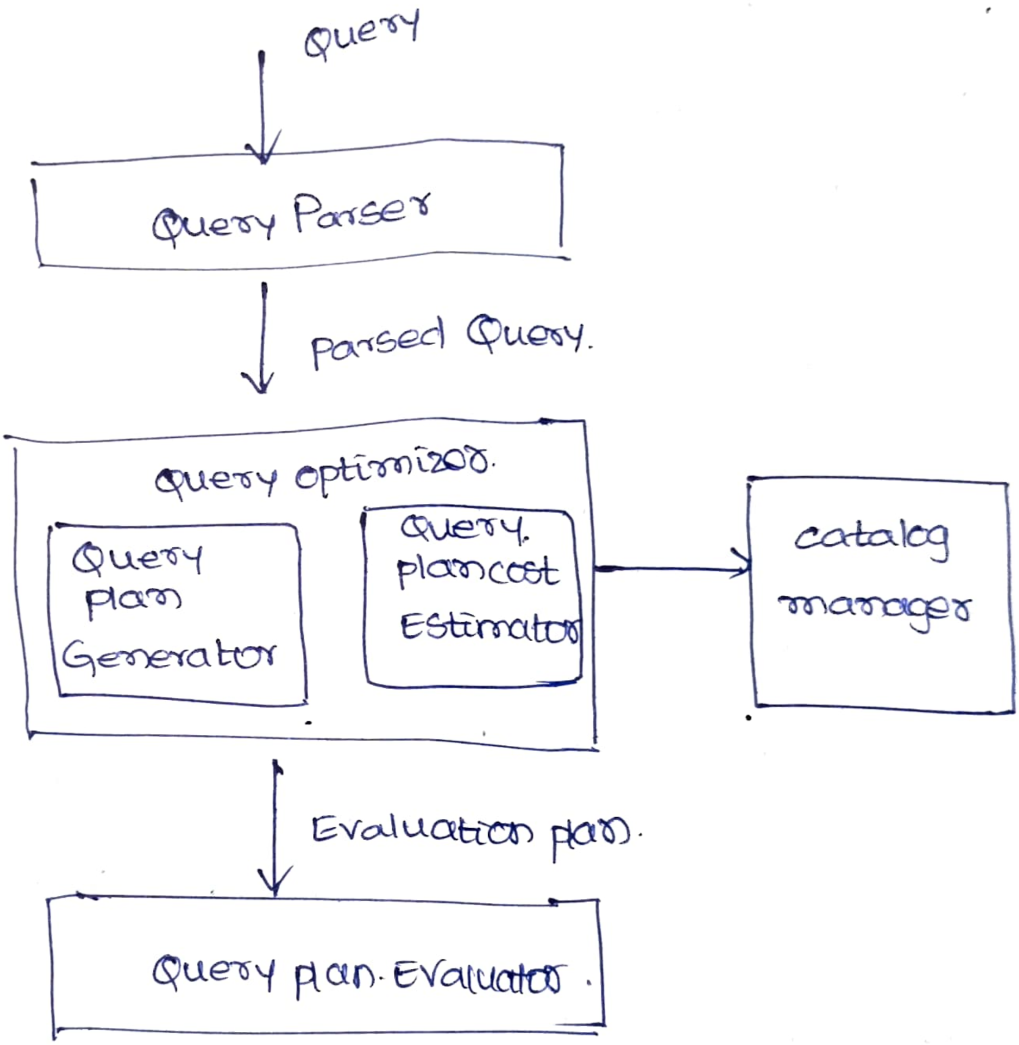
It generates different plans for execution of the query.

Query plan cost Estimator:-

It estimates the cost of each execution plan of the query.

3. Query Plan Evaluator:-

It selects the best plan for evaluating the query.



Query parsing, optimization and Execution.

Query Evaluation Plan:-

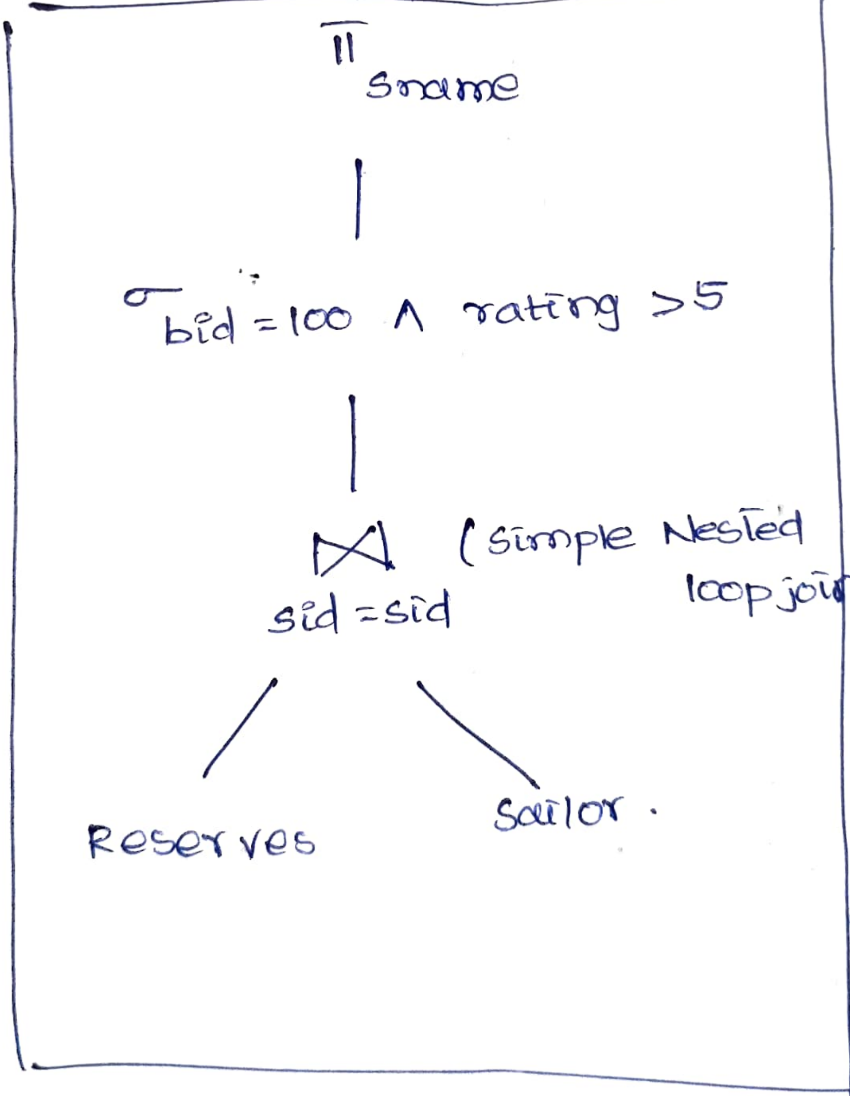
A query evaluation plan consists of an extended relational algebra tree with additional annotations at each node indicating the access methods to use for each table and the implementation method to use for each relational operator.

SQL Query

```
SELECT   S.sname
FROM     Reserves R, Sailor S
WHERE    R.sid = S.sid and
         R.bid = 100 and S.rating > 5
```

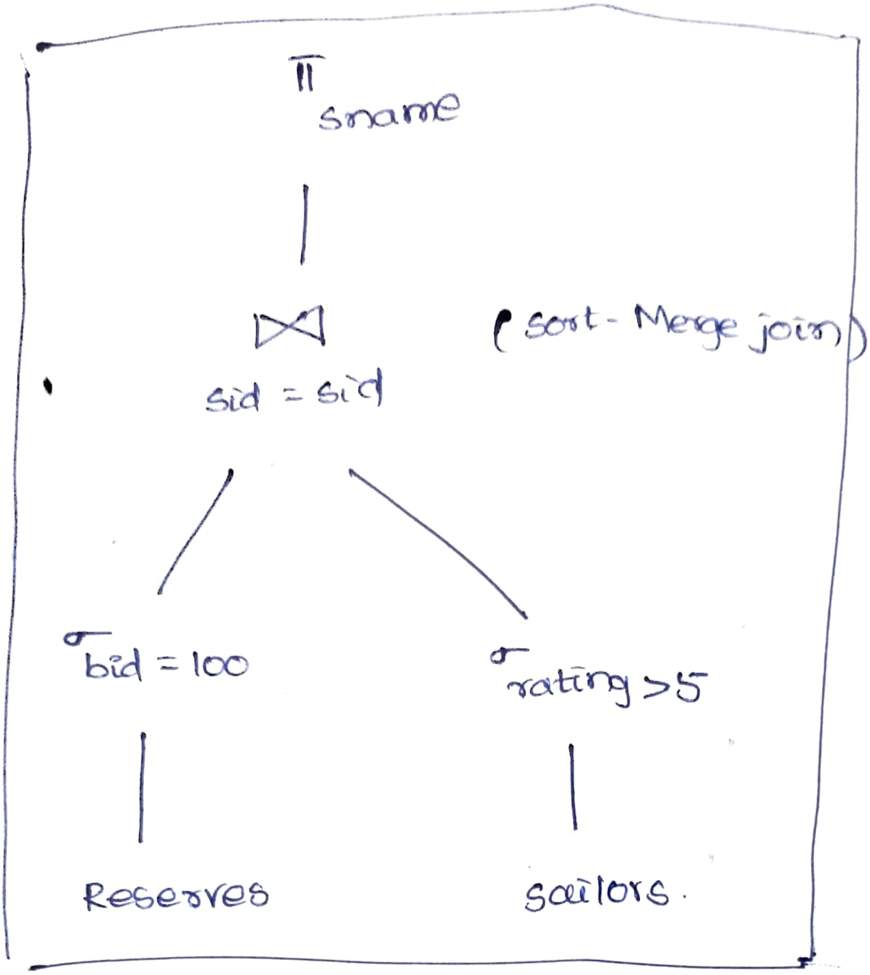
$$\pi_{sname}\left(\sigma_{bid=100 \wedge rating>5}\left(Reserves \bowtie_{sid=sid} Sailor\right)\right)$$

Relational algebra expression.

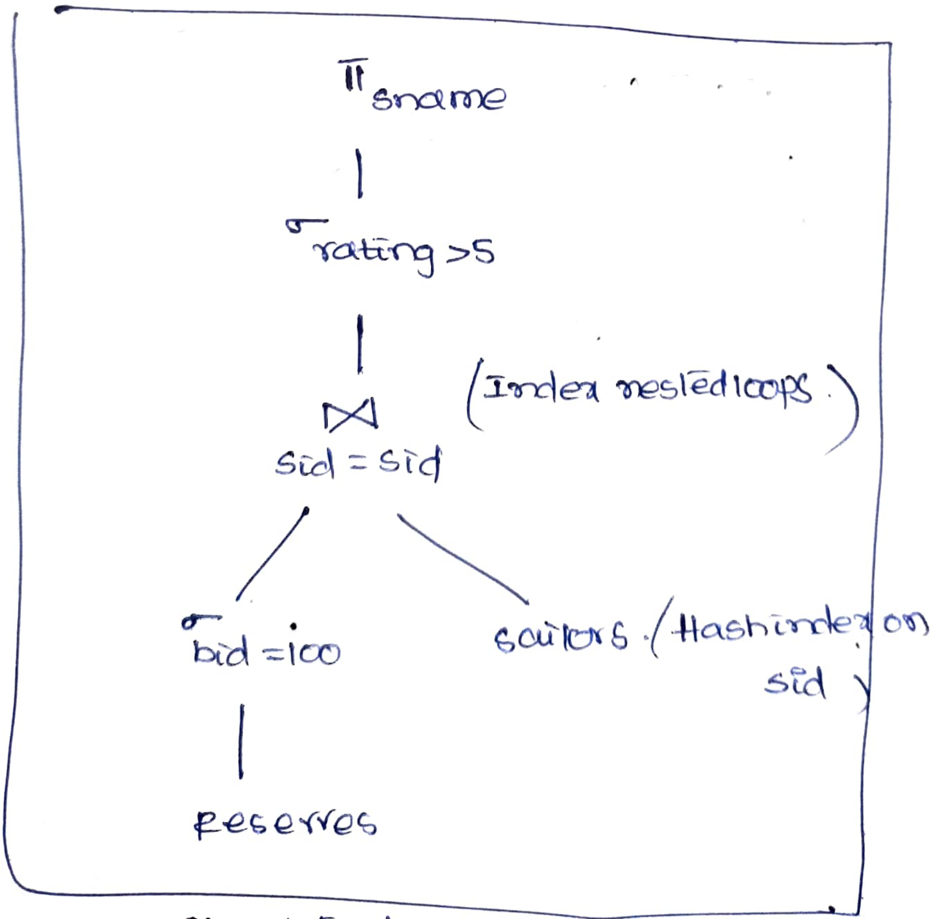


Relational algebra tree

Query Evaluation plan-1



Query Evaluation plan-2



Query Evaluation Plan-3.

Cost of a plan

The cost of a plan is the sum of costs for the operators it contains.

The cost of a plan can be broken down into three parts:

1. Reading the Input tables
2. Writing Intermediate tables.
3. Sorting the final result.

Equivalence Rules

An equivalence rule says that expressions of two forms are equivalent.

1. $\sigma_{\theta_1 \wedge \theta_2}(E) = \sigma_{\theta_1}(\sigma_{\theta_2}(E))$

2. $\sigma_{\theta_1}(\sigma_{\theta_2}(E)) = \sigma_{\theta_2}(\sigma_{\theta_1}(E))$

3. $\Pi_{L1}(\Pi_{L2}(\ldots \Pi_{Ln}(E))\ldots)) = \Pi_{L1}(E)$

4. $\sigma_\theta(E_1 \times E_2) = E_1 \bowtie_\theta E_2$

5. $\sigma_{\theta_1}(E_1 \bowtie_{\theta_2} E_2) = E_1 \bowtie_{\theta_1 \wedge \theta_2} E_2$

6. $E_1 \bowtie_\theta E_2 = E_2 \bowtie_\theta E_1$

7. $(E_1 \bowtie E_2) \bowtie E_3 = E_1 \bowtie (E_2 \bowtie E_3)$

8. $E_1 \cup E_2 = E_2 \cup E_1$

9. $E_1 \cap E_2 = E_2 \cap E_1$

10. $(E_1 \cup E_2) \cup E_3 = E_1 \cup (E_2 \cup E_3)$

11. $E_1 \cap (E_2 \cap E_3) = E_1 \cap (E_2 \cap E_3)$

12. $\sigma_\theta(E_1 \cup E_2) = \sigma_\theta(E_1) \cup \sigma_\theta(E_2)$

13. $\Pi_L(E_1 \cup E_2) = \Pi_L(E_1) \cup \Pi_L(E_2)$

Estimating Statistics of Expression Results

The cost of an operation depends on the size and other statistics of its inputs.

DBMS catalog

The DBMS catalog stores the following statistical information about database relations.

n_r — The number of tuples in the relation r

b_r — The number of blocks containing tuples of relation r

l_r — The size of a tuple of relation r in bytes

f_r — The number of tuples of relation r that fit into one block.

$V(A,r)$ — The number of distinct values that appear in the relation r for attribute A.

Selection Size Estimation

① $\sigma_{A=a}(r)$ [Equality predicate]

If we assume uniform distribution of values, the selection result can be estimated to have

$$\frac{n_r}{V(A,r)} \text{ tupes.}$$

assuming that the value a appears in attribute A of some record of r.

② $\sigma_{A \le v}(r)$ [Single comparison predicate]

$min(A, r)$ — lowest value of A

$max(A, r)$ — highest value of A.

If $v < min(A, r)$ then the number of records that satisfy the condition is 0

If $v \ge max(A, r)$ then the number of records that satisfy the condition is n_r

Then the selection result is

$$n_r \cdot \frac{v - min(A, r)}{max(A, r) - min(A, r)}$$ in other cases.

③ Conjunctive Selection

A conjunctive selection is a selection of the form.

$$\sigma_{\theta_1 \wedge \theta_2 \wedge \cdots \theta_n}(r)$$

For each θ_i, the size of selection $\sigma_{\theta_i}(r)$ is s_i

The number of tuples in the full selection is estimated as

$$n_r * \frac{s_1 * s_2 * \cdots * s_n}{n_r^{\,n}}$$

4. <u>Disjunctive selection</u>:-

A disjunctive selection is a selection of the form

$$\sigma_{\theta_1 \vee \theta_2 \vee \cdots \vee \theta_n}(r)$$

The number of tuples in the full selection is estimated as

$$n_r \left(1 - \left(1 - \frac{s_1}{n_r}\right) * \left(1 - \frac{s_2}{n_r}\right) * \cdots * \left(1 - \frac{s_n}{n_r}\right)\right)$$

5. <u>Negation</u>

$$\sigma_{\neg\theta}(r) = n_r - [\sigma_\theta(r)]$$

<u>Join Size Estimation</u>

Let $r(R)$ and $s(S)$ be relations.

1. If $R \cap S = \phi$ then the join size estimation is same as cartesian product i.e. $n_r * n_s$
2. If $R \cap S$ is a key for R, then the number of tuples in $R \bowtie S$ is the number of tuples in S.

   If $R \cap S$ is a key for S, then the number of tuples in $R \bowtie S$ is the number of tuples in R.
3. If $R \cap S$ is neither a key for R and S then the number of tuples in $R \bowtie S$ is

$$\text{minimum} \left\{ \frac{n_r * n_s}{V(A, s)}, \frac{n_r * n_s}{V(A, r)} \right\}.$$

Projection :-

The size of a projection $\Pi_A(r)$ is $V(A, r)$

Set operations :-

The estimated size of $r \cup s$ is the sum of the sizes of r and s.

The estimated size of $r \cap s$ is the minimum of the sizes of r and s.

The estimated size of $r - s$ is equal to size of r.

Left outer join

$$\text{size}(r ⟕ s) = \text{size}(r \bowtie s) + \text{size}(r)$$

Right Outer Join

$$\text{size}(r ⟖ s) = \text{size}(r \bowtie s) + \text{size}(s)$$

Full Outer Join

$$\text{size}(r ⟗ s) = \text{size}(r \bowtie s) + \text{size}(r) + \text{size}(s)$$

Query optimization

Query optimization is the process of choosing the most efficient query evaluation plans for executing the query.

There are two methods of Query optimization.

1. Cost-based Query optimization. (CBO)
2. Rule-based Query optimization. (RBO)

Cost components of Query Execution:-

1. Access cost to secondary storage

This can be the cost of searching, reading or writing data blocks that are originally found on secondary storage especially on the disk

2. Memory Usage cost :-

The cost of memory usage can be calculated simply by using the number of memory buffers that are needed for the execution of the query.

3. Storage cost :-

It is the cost of storing any intermediate files of processing the input that are generated by the execution strategy of the query.

4. Computational cost

It is the cost of performing the memory operations that are available on the record within the data buffers.

5. communication cost :-

It is the cost associated with sending or communicating the query and its results from one place to another .

Cost-based optimizer :-

A cost-based optimizer generates a range of query-evaluation plans from the given query by using the equivalence rules .

Rule-based optimizer

It selects the efficient query evaluation plan based on rules.

Rule-1 : perform selection operation early.

Rule-2 : Perform projection operation early.

Rule-3 : Avoid cartesian product .