**Nmk96 Advanced Database**

**Chapter 4: Query processing and Optimization**

## **Query Processing**

**Query Processor**: Group of components of a DBMS that turns user queries and data modification commands into a sequence of database operations and executes those operations. Query Processing Refers to the range of activities involved (parsing, validating, optimizing, and executing a query)in extracting data from a database. The aim of query processing is to find information in one or more databases and deliver it to the user quickly and efficiently. Traditional techniques work well for databases with standard, single-site relational structures, but databases containing more complex and diverse types of data demand new **query processing and optimization techniques**.

The aims of query processing are to transform a query written in a high-level language, typically SQL, into a correct and efficient execution strategy expressed in a low-level language (implementing the relational algebra), and to execute the strategy to retrieve the required data. A Query Processor is responsible to produce an execution plan that will guarantee an acceptable response time. The activities involved in.

Query Processing can be divided into the following main phases:

* 1. Query Decomposition
  2. Optimization
  3. Code generation and
  4. Execution

### **Query Decomposition**

* Query decomposition is the process of transforming a high-level query into a relational algebra query, and to check that the query is syntactically and semantically correct.
* Query decomposition consists of parsing and validation

Typical stages in query decomposition are:

1. **Analysis**: lexical and syntactical analysis of the query (correctness). Query tree will be built for the query containing leaf node for base relations, one or many non-leaf nodes for relations produced by relational algebra operations and root node for the result of the query. Sequence of operation is from the leaves to the root.
2. **Normalization**: convert the query into a normalized form. The predicate WHERE will be converted to Conjunctive (∨) or Disjunctive (∧) Normal form.
3. **Semantic Analysis**: to reject normalized queries that is not correctly formulated or contradictory. Incorrect if components do not contribute to generate result. Contradictory if the predicate cannot be satisfied by any tuple.
4. **Simplification**: to detect redundant qualifications, eliminate common sub-expressions, and transform the query to a semantically equivalent but more easily and effectively computed form.
5. **Query Restructuring** More than one translation is possible Use transformation rules

## **Query Optimization**

What is wrong with the ordinary query?

* Everyone wants the performance of their database to be optimal. In particular, there is often a requirement for a specific query or object that is query based, to run faster.
* Problem of query optimization is to find the sequence of steps that produces the answer to user request in the most efficient manner, given the database structure.
* The performance of a query is affected by the tables or queries that underlies the query and by the/ Choosing a query execution plan is called Query Optimization and it mainly means making decisions about data access methods. Query Optimization strongly relies on File Organization techniques.

Query optimizers are one of the main means by which modern database systems achieve their performance advantages. Given a request for data manipulation or retrieval, **an optimizer will choose an optimal plan for evaluating the request from among the manifold alternative strategies.** i.e. there are many ways (access paths) for accessing desired file/record. The optimizer tries to select the most efficient (cheapest) access path for accessing the data. DBMS is responsible to pick the best execution strategy based various considerations.

Query optimizers were already among the largest and most complex modules of database systems.The objective in query optimization is to select an efficient execution strategy. As there are many equivalent transformations from same high-level query, aim is to choose the one that minimizes resource usage. Generally, an efficient execution plan reduces the total execution time of a query; thereby reducing the response time of a query. The problem is computationally intractable with large number of relations, so the strategy adopted is reduced to finding a near optimum solution.

**Most efficient processing**: Least amount of I/O and CPU resources.

### **Selection of the best method:**

In a **non-procedural language**, the system does the optimization at the time of execution. On the other hand, in a **procedural language**, programmers have some flexibility in selecting the best method.

For optimizing the execution of a query the programmer must know:

* File organization
* Record access mechanism and primary or secondary key.
* Data location on disk.
* Data access limitations.

### **Approaches to Query Optimization**

There are two approaches to Query Optimization,

1. heuristic approach which improves and refine relational Algebra tree to create equivalent logical query plan. It orders operations in a query.
2. cost based estimation which use database statistics to estimate physical costs of logical operators. The second uses comparing different strategies based on relative cost, and selecting one that minimizes resource usage

* Heuristic (Logical Transformations)
  + Use Transformation Rules to convert one relational algebra expression into an equivalent form that is known to be more efficient
  + Heuristic Optimization Guidelines
* Cost Based (Physical Execution Costs)
  + Data Storage/Access Refresher
  + Catalog & Costs

#### **Heuristics Approach**

* The heuristic approach uses the knowledge of the characteristics of the relational algebra operations and the relationship between the operators to optimize the query.
* Thus, the heuristic approach of optimization will make use of:
  + **Properties of individual operators**
  + **Association between operators**
  + **Query Tree**: a graphical representation of the operators, relations, attributes and predicates and processing sequence during query processing.
    - Query tree is composed of three main parts:**A query tree** is a tree data structure that corresponds to a relational algebra expression.
    1. **The Leafs**: the base relations used for processing the query/ extracting the required information.
    2. **The Root**: the final result/relation as an output based on the operation on the relations used for query processing
    3. **Nodes**: intermediate results or relations before reaching the final result.
    - Sequence of execution of operation in a query tree will start from the leaves and continues to the intermediate nodes and ends at the root.

The properties of each operations and the association between operators is analyzed using set of rules called transformation rules. Use of the transformation rules will transform the query to relatively good execution strategy. In using heuristics during query optimization, the following steps must be followed:

* Translate queries into query trees or query graphs
* Apply transformation rules for relational algebra expression.
* Perform heuristic optimization

**Example**

Example Join Query over R(A,B,C) and S(C,D,E): **Select** B, D**From** R, S**Where** R.A = “c” AND S.E = 2 AND R.C = S.C

We need transformation rule to address such query

##### **Transformation Rules for Relational Algebra**

1. Cascade of Selection: conjunctive Selection Operations can cascade into individual Selection Operations and Vice Versa

***σ(c1∧c2∧c3) (R)= σc1(σc2(σc3(R))*** where ***ci*** is a predicate

1. Commutativity of Selection operations

***σc1(σc2(R))= σc2(σc1(R))*** where ***ci*** is a predicate

1. Cascade of Projection: in the sequence of Projection Operations, only the last in the sequence is required

***L1L2L3L4(R)=L1(R)***

1. Commutativity of Selection with Projection and Vise Versa
   1. If the predicate ***c1*** involves only the attributes in the projection list (***L1)***, then the selection and projection operations commute

***L1(σc1(R))= σc1 (L1(R))***

1. Commutativity of Theta Join/Cartesian Product

R X S is equivalent to S X R

Also holds for Equi-Join and Natural-Join

***(R******c1S)= (S******c1R)***

1. Commutativity of Selection with Theta Join
   1. If the predicate ***c1*** involves only attributes of one of the relations (R) being joined, then the Selection and Join operations commute

***σc1 (R******c S)= (σc1 (R))******c S)***

* 1. If the predicate is in the form ***c1***∧***c2*** and ***c1*** involves only attributes of R and ***c2*** involves only attributes of S, then the Selection and Theta Join operations commute

***σc1∧c2 (R******c S)= (σc1 (R))******c (σc2 S))***

1. Commutativity of Projection and Theta Join

δIf the projection list is of the form ***L1, L2***, where ***L1*** involves only attributes of R and ***L2*** involves only attributes of S being joined and the predicate ***c*** involves only attributes in the projection list, then the Selection and Join operations commute

***L1,L2 (R******c S)= (L1,L2(R))******c (L1,L2 (S))***

1. Commutativity of the Set Operations: Union and Intersection but not Se t difference

***R∩S=S∩R*** and ***R∪S=S∪R***

1. Associativity of the Theta JOIN,Cartesian Product, Union and Intersection.

***(Rθ S)θ T=Rθ (Sθ T)*** where***θ***is one of the operations

1. Commuting Selection with set operations

***σc (Rθ S)= (σc(R) θσc(S))*** where***θ***is one of the operations

1. Commuting projection with union

***L1 (S∪R) = L1 (S) ∪L1 (R)***

Heuristic Approach will be implemented by using the above transformation rules in the following sequence or steps.

***Sequence for Applying Transformation Rules***

1. Use

Rule-1🡺 Cascade Selection

1. Use

Rule-2: Commutativity of Selection

Rule-4: Commuting selection with projection

Rule-6: Commuting selection with Join and Cartesian

Rule-10: commuting selection with setoperations

1. Use

Rule-9: Associativity of Binary Operations (Join,Cartesian, Union and Intersection). Rearrange nodes by making the most restrictive operations to be performed first (moving it as far down the tree as possible)

1. Perform Cartesian Operations with the subsequent Selection Operation
2. Use

Rule-3: Cascade of projection

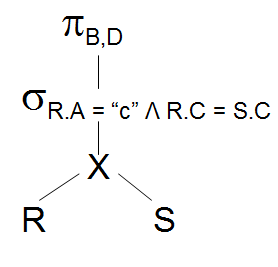
Rule-4: Commuting projection with selection

Rule-7: Commuting projection with Join and Cartesian

Rule-11: commuting projection with Union

**Q**uery compilation is divided into three steps: -

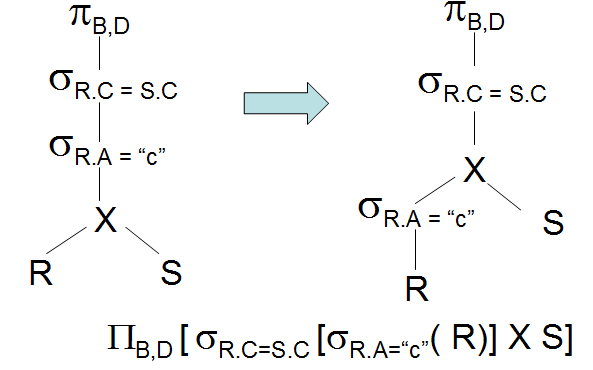
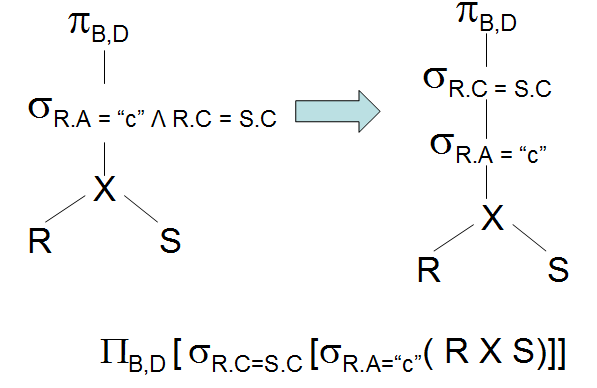
1. **Parsing**: Parse SQL query into parser tree. Parse tree, representing the query and its structure is constructed
2. **Logical query plan**: Transforms parse tree into expression tree of relational algebra. Parse tree – converted to an initial query plan –an algebraic representation of the query
3. **Physical query plan**: Transforms logical query plan into physical query plan. Abstract/initial/logical plan – turned into physical plan by selecting algorithms to implement each of the operations of the logical plan which includes:
   * Operation performed
   * Order of operation
   * Algorithm used
   * The way in which stored data is obtained and passed from one operation to another

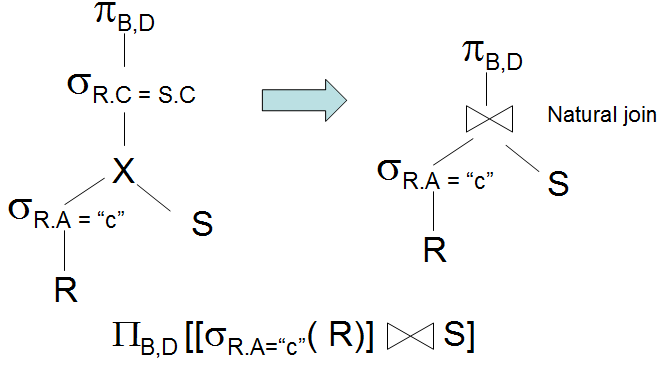
**Example**:

Select B,D From R,S Where R.A = “c” ∧ R.C=S.C

***Relational Algebra:***

ΠB,D [ sR.A=“c”∧ R.C = S.C (RXS)]



******

***Main Heuristic***

The main heuristic is to first apply operations that reduce the size (the cardinality and/or the degree) of the intermediate relation. That is:

1. Perform Selection as early as possible: that will reduce the cardinality (number of tuples) of the relation.
2. Perform Projection as early as possible: that will reduce the degree (number of attributes) of the relation.

* Both a and b will be accomplished by placing the select and project operations as far down the tree as possible.

1. Select and join operations with most restrictive conditions resulting with smallest absolute size should be executed before other similar operations. This is achieved by reordering the nodes with join

**Example**: consider the following schemas and the query, where the employee and the project relations are related by the works\_on relation.

Employee (EEmpID, FName, LName, Salary, Dept, Sex, DoB)

Project (PProjID, PName, PLocation, PFund, PManagerID)

Works\_on (WEmpID, WProjID)

WEmpID (refers to employee identification) and PProjID (refers to project identification) are foreign keys to WORKS\_ON relation from Employee and Project relations respectively.

**Query**: The manager of the company working on road construction would like to view employees name born before January 1 1965 who are working on the project named Ring Road.

Relational Algebra representation of the query will be:

***π<FName, LName> (<****DoB<Jan1 1965****∧WEmpID=EEmpID∧PProjID=WProjID∧PName=’Ring*** *Road’****>(***EMPLOYEE ***X*** Works\_on ***X*** PROJECT)***)***

The SQL equivalence for the above query will be:

**Select**FName, LName**FROM** EmPLOYEE, Works\_on, Project**Where***DoB<Jan 1 1965 ∧EEmpID=WEmpID∧WProjID=PProjID∧PName=”Ring Road”*

The initial query tree will be:

***(****DoB<Jan1 1965)* ***∧(WEmpID=EEmpID)∧ (PProjID=WProjID)∧(PName=’Ring*** *Road’)*

***<FName, LName>***

X

X

By applying the first step (cascading the selection) we will come up with the following structure.

***(****DoB<Jan1 1965)****( (WEmpID=EEmpID)( (PProjID=WProjID)( (PName=’Ring*** *Road’)* (EMPLOYEE ***X*** Works\_on ***X*** PROJECT)***) ) )***

By applying the second step it can be seen that some conditions have attribute that belong to a single relation ( DoB belongs to EMPLOYEE and PName belongs to PROJECT) thus the selection operation can be commuted with Cartesian Operation. Then, since the condition ***WEmpID=EEmpID***base the employee andworks\_on relation the selection with this condition can be cascaded.

***((PProjID=WProjID) ((PName=’Ring*** *Road’)* PROJECT***) X ((WEmpID=EEmpID) (***Works\_on ***X (****DoB<Jan1 1965)* EMPLOYEE***))))***

The query tree after this modification will be:

***(PProjID=WProjID)***

***<FName, LName>***

X

X

***(****DoB<Jan1 1965)*

***(WEmpID=EEmpID)***

***(PName=’Ring*** *Road’)*

Using the third step, perform most restrictive operations first.

From the query given we can see that selection on Project is most restrictive than selection on Employee. Thus, it is better to perform selection on project befor on employee. rearrange the nodes to achieve this.

***(WEmpID=EEmpID)***

***<FName, LName>***

X

X

***(PName=’Ring*** *Road’)*

***(PProjID=WProjID)***

***(****DoB<Jan1 1965)*

Using the forth step, Perform Cartesian Operations with the subsequent Selection Operation.

***<FName, LName>***

***(PProjID=WProjID)***

***(WEmpID=EEmpID)***

***(PName=’Ring*** *Road’)*

***(****DoB<Jan1 1965)*

Using the fifth step, Perform the projection as early as possible.

***(PName=’Ring*** *Road’)*

***<FName, LName>***

***(PProjID=WProjID)***

***(WEmpID=EEmpID)***

***(****DoB<Jan1 1965)*

***<PProjID>***

***<FName,LName,EEmpID>***

***<WEmpID>***

For every project located in ‘Stafford’, list the project number, the controlling department number and the department manager’s last name, address, and birth date.

#### **II. Cost Estimation Approach to Query Optimization**

The main idea is to minimize the cost of processing a query. The cost function is comprised of:

* I/O cost + CPU processing cost + communication cost + Storage cost

These components might have different weights in different processing environments

The DBMs will use information stored in the system catalogue for the purpose of estimating cost. The main target of of query optimization is to minimize the size of the intermediate relation.

The size will have effect in the cost of:

* Disk Access
* Data Transpiration
* Storage space in the Primary Memory
* Writing on Disk

The statistics in the system catalogue used for cost estimation purpose are:

* Cardinality of a relation: the number of tuples contained in a relation currently (r)
* Degree of a relation: number of attributes of a relation
* Number of tuples on a relation that can be stored in one block of memory
* Total number of blocks used by a relation
* Number of distinct values of an attribute (d)
* Selection Cardinality of an attribute (S): that is average number of records that will satisfy an equality condition S=r/d

By sing the above information one could calculate the cost of executing a query and selecting the best strategy, which is with the minimum cost of processing.

##### **Cost Components for Query Optimization**

The costs of query execution can be calculated for the following major process we have during processing.

1. ***Access Cost of Secondary Storage***

Data is going to be accessed from secondary storage, as an query will be needing some part of the data stored in the database. The disk access cost can again be analyzed in terms of:

* Searching
* Reading, and
* Writing, data blocks used to store some portion of a relation.

The disk access cost will vary depending on the

* File organization used and the access method implemented for the file organization.
* The data allocation scheme, whether the data is stored contiguously or in scattered manner, will affect the disk access cost.

1. ***Storage Cost*** While processing a query, as any query would be composed of many database operations, there could be one or more intermediate results before reaching the final output. These intermediate results should be stored in primary memory for further processing. The bigger the intermediate relation, the larger the memory requirement, which will have impact on the limited available space. This will be considered as a cost of storage.
2. ***Computation Cost*** Query is composed of many operations. The operations could be database operations like reading and writing to a disk, or mathematical and other operations like:
   * Searching
   * Sorting
   * Merging
   * Computation on field values
3. ***Communication Cost*** In most database systems the database resides in on station and various queries originate from different terminals. This will have impact on the performance of the system adding cost for query processing. Thus, the cost of transporting data between the database site and the terminal from where the query originate should be analyzed.