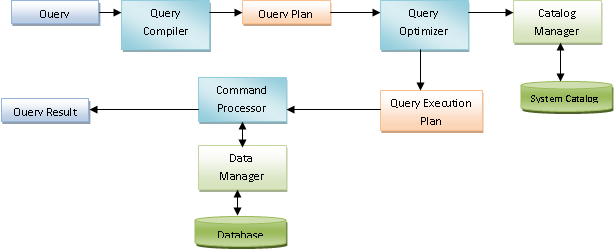
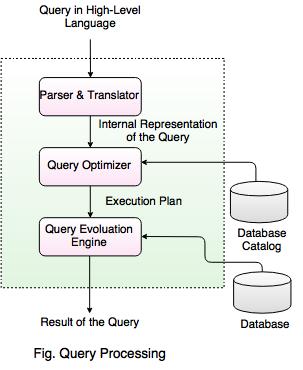
Introduction to Query Processing 27.08.2019

* Query Processing is a translation of high-level queries into low-level expression.
* It is the step by step process of breaking the high level language into low level language which machine can understand and perform the requested action for user. Query processor in the DBMS performs this task.
* It is a step wise process that can be used at the physical level of the file system, query optimization and actual execution of the query to get the result.
* It requires the basic concepts of relational algebra and file structure.
* It refers to the range of activities that are involved in extracting data from the database.
* It includes translation of queries in high-level database languages into expressions that can be implemented at the physical level of the file system.
* 

**In the above diagram,**

* When a query is submitted to the database, it is received by the query compiler.
* It then scans the query and divides it into individual tokens.
* Once the tokens are generated, they are verified for their correctness by the parser, Parser checks the syntax and verifies the relations and the attributes which are used in the query.
* Then the tokenized queries are transformed into different possible relational expressions, relational trees and relational graphs (Query Plans).
* Query optimizer then picks them to identify the best query plan to process
* It checks in the system catalog for the constraints and indexes and decides the best query plan.
* It generates different execution plans for the query plan.
* The query execution plan then decides the best and optimized execution plan for execution.
* The command processor then uses this execution plan to retrieve the data from the database and returns the result.

**There are four phases in a typical query processing.**

* Parsing and Translation
* Query Optimization
* Evaluation or query code generation
* Execution in DB’s runtime processor.

**Translating SQL Queries into Relational Algebra**

#### **Example**

SELECT Ename FROM Employee  
  WHERE Salary > 5000;

**Translated into Relational Algebra Expression**  
  
σ Salary > 5000 (π Ename (Employee))  
                                        OR  
π Ename (σ Salary > 5000 (Employee))

**Equivalence rule:**

Two relational algebra expressions are said to be equivalent if the two expressions generate the same set of tuples on every legal database instance

Relational Algebraic Equivalence Transformation

Rules

1. Conjunctive selection operations can be deconstructed into a sequence of individual selections; cascade of σ.

σθ1∧θ2(E) = σθ1(σθ2(E))

2. Selection operations are commutative:

σθ1(σθ2(E)) = σθ2(σθ1(E))

3. Only the final operations in a sequence of projection operations

is needed, the others can be omitted; cascade of Π

ΠL1(ΠL2(. . .(ΠLn(E)). . .)) = ΠL1(E)

4. Selections can be combined with Cartesian products and theta

joins:

σθ(E1 × E2) = E1 ⊲⊳θ E2

σθ1(E1 ⊲⊳σθ2E2) = E1 ⊲⊳θ1∧θ2 E2

5. Theta join operations are commutative:

E1 ⊲⊳θ E2 = E2 ⊲⊳θ E1

6. Natural-join operations are associative:

(E1 ⊲⊳ E2) ⊲⊳ E3 = E1 ⊲⊳ (E2 ⊲⊳ E3)

Theta joins are associative in the following manner

(E1 ⊲⊳θ1 E2) ⊲⊳θ2∧θ3 E3 = E1 ⊲⊳θ1∧θ3

(E2 ⊲⊳θ2 E3)

where θ2 involves attributes from E2 and E3 only.

7. The selection operation distributes over the theta join operation

under the following two conditions:

(a) It distributes when all the attributes in the selection condition

θ0 involve only the attributes of one of the expressions (E1)

being joined.

σθ0

(E1 ⊲⊳θ E2) = (σθ0

(E1)) ⊲⊳θ E2

(b) It distributes when the selection condition θ1 involves only

the attributes of E1 and θ2 involves only the attributes of E2

σθ1∧θ2

(E1 ⊲⊳θ E2) = (σθ1

(E1)) ⊲⊳θ (σθ2

(E2))

8. The projection operation distributes over the theta join.

(a) Let L1 and L2 be attributes of E1 and E2 respectively. Suppose that the join condition θ involves only attributes in L1∪

L2. Then

ΠL1∪L2

(E1 ⊲⊳θ E2) = (ΠL1

(E1)) ⊲⊳θ (ΠL2

(E2))

(b) Consider a join E1 ⊲⊳θ E2. Let L1 and L2 be sets of attributes

from E1 and E2 respectively. Let L3 be attributes of E1 that

are involved in the join condition θ, but are not in L1 ∪ L2,

and let L4 be attributes of E2 that are involved in the join

condition θ, but are not in L1 ∪ L2. Then

ΠL1∪L2

(E1 ⊲⊳θ E2) = ΠL1∪L2

((ΠL1∪L3

(E1)) ⊲⊳θ (ΠL2∪L4

(E2)))

9. The set operations union and intersection are commutative.

E1 ∪ E2 = E2 ∪ E1

E1 ∩ E2 = E2 ∩ E1

Set difference is not commutative.

10. Set union and intersection are associative.

(E1 ∪ E2) ∪ E3 = E1 ∪ (E2 ∪ E3)

(E1 ∩ E2) ∩ E3 = E1 ∩ (E2 ∩ E3)

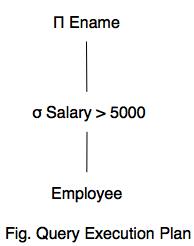
11. The selection operation distributes over the union, intersection,

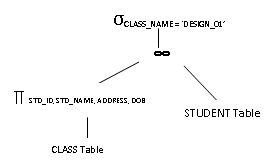
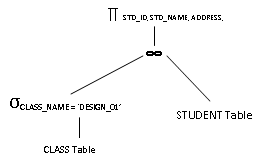
and set-difference operations.

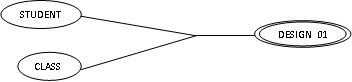
σP (E1 − E2) = σP (E1) − E2 = σP (E1) − σP (E2)

12. The projection operation distributes over the union operation.

ΠL(E1 ∪ E2) = (ΠL(E1)) ∪ (ΠL(E2))



A query tree is a tree data structure representing a relational algebra expression. The tables of the query are represented as leaf nodes. The relational algebra operations are represented as the internal nodes. The root represents the query .  



* A sequence of primitive operations that can be used to evaluate a query is a Query Execution Plan or Query Evaluation Plan.
* The above diagram indicates that the query execution engine takes a query execution plan and returns the answers to the query.
* Query Execution Plan minimizes the cost of query evaluation.

Introduction to Query Optimization

* Query optimization is a difficult part of the query processing.
* It determines the efficient way to execute a query with different possible query plans.
* It cannot be accessed directly by users once the queries are submitted to the database server or parsed by the parser.
* A query is passed to the query optimizer where optimization occurs.
* Main aim of Query Optimization is to minimize the cost function,  
  I/O Cost + CPU Cost + Communication Cost
* It defines how an RDBMS can improve the performance of the query by re-ordering the operations.
* It is the process of selecting the most efficient query evaluation plan from among various strategies if the query is complex.
* It computes the same result as per the given expression, but it is a least costly way of generating result.

Importance of Query Optimization

* Query optimization provides faster query processing.
* It requires less cost per query.
* It gives less stress to the database.
* It provides high performance of the system.
* It consumes less memory.

**DETAILED DESCRIPTION**

### **Parsing and Translation**

This is the first step of any query processing. The user typically writes his requests in SQL language. In order to process and execute this request, DBMS has to convert it into low level – machine understandable language. Any query issued to the database is first picked by query processor. It scans and parses the query into individual tokens and examines for the correctness of query. It checks for the validity of tables / views used and the syntax of the query. Once it is passed, then it converts each tokens into relational expressions, trees and graphs. These are easily processed by the other parsers in the DBMS.

SELECT STD\_ID, STD\_NAME, ADDRESS, DOB

FROM STUDENT s, CLASS c

WHERE s.CLASS\_ID = c.CLASS\_ID

AND c.CLASS**\_**NAME = ‘DESIGN\_01’;

When he issues this query, the DBMS reads and converts it into the form which DBMS can use to further process and synthesis it. This phase of query processing is known as parsing and translation phase. The query processor scans the SQL query submitted and divides into individual meaningful tokens. In our example, ’SELECT \* FROM’, ‘STUDENT s’, ‘CLASS c’, ‘WHERE’, ‘s.CLASS\_ID = c.CLASS\_ID’, ‘AND’ and ‘c.CLASS\_NAME = ‘DESIGN\_01’’ are the different tokens. These tokenized forms of query are easily used by the processor to further processing. It fires query on the data dictionary tables to verify if the tables and columns in these tokens exists or not. If they are not present in the data dictionary, then the submitted query will be failed at this stage itself. Else it proceeds to find if the syntax used in the query are correct. Please note that it does not validate if DESIGN\_01 exists in the table or not, it verifies if ’SELECT \* FROM’, ‘WHERE’, ‘s.CLASS\_ID = c.CLASS\_ID’, ‘AND’ etc have SQL defined syntaxes. Once it validates the syntaxes, it converts them into a relational algebra, relational tree and graph representations. These are easily understood and handled by the optimizer for further processing. Above query can be converted into any of the two forms of relation algebra as below. First query identifies the students in DESIGN\_01 class first and then selects only the requested columns from it. Another query first selects requested columns from the STUDENT table and then filters it for DESIGN\_01.  Both of them results in same result.

∏ STD\_ID, STD\_NAME, ADDRESS, DOB (σ CLASS\_NAME = ‘DESIGN\_01’ (STUDENT ∞CLASS))

 or

σ CLASS\_NAME = ‘DESIGN\_01’ (∏ STD\_ID, STD\_NAME, ADDRESS, DOB (STUDENT ∞CLASS))

Query processor then applies the rules and algorithms on these relational structures to represent more efficient and powerful structures which are used only by the DBMS. These structures are based on the mappings between the tables, joins used, cost of execution algorithm of these queries. It determines which structure – selecting and then projecting or projecting and then selecting – is the efficient way of processing, when to apply filters etc. In the third step of query processing, the best structure and plan selected by the optimizer is selected and executed. It digs into the database memory to retrieve the records based on the plan. Sometimes it process and compiles the query and keeps it in DB to use it in the runtime DB processor. The result is then returned to the user. This is the overall step processed by the DBMS when a simple to complex query is fired. The time taken by all these process will be in fraction of seconds. But ideal optimization and selection of execution path will make the query even faster