Distributed Database System (ECS-242/602)

Mr. Abdul Rehman email id: abdul@northern.edu.pk Whatsapp# 03087792217

(Week 5) Lecture 9-10 (Chapter 21 of Book)

Objectives: Learning objectives of this lecture are

- **Query Processing**
- **Query Decomposition**
  - **Analysis**
  - **Normalization**
  - **Semantic Analysis**
  - **Simplification**

Query Processing:

The activities involved in parsing, validating, optimizing, and executing a query.

Query Optimization:

The activity of choosing an efficient execution strategy for processing a query.

Details:

Query processing can be divided into four main phases:

1- Decomposition (consisting of parsing and validation)
2- Optimization
3- Code generation
4- Execution

Distributed Database System (ECS-242/602)

Mr. Abdul Rehman **email id: abdul@northern.edu.pk Whatsapp# 03087792217**

Important Terms:

1- **System Catalog**

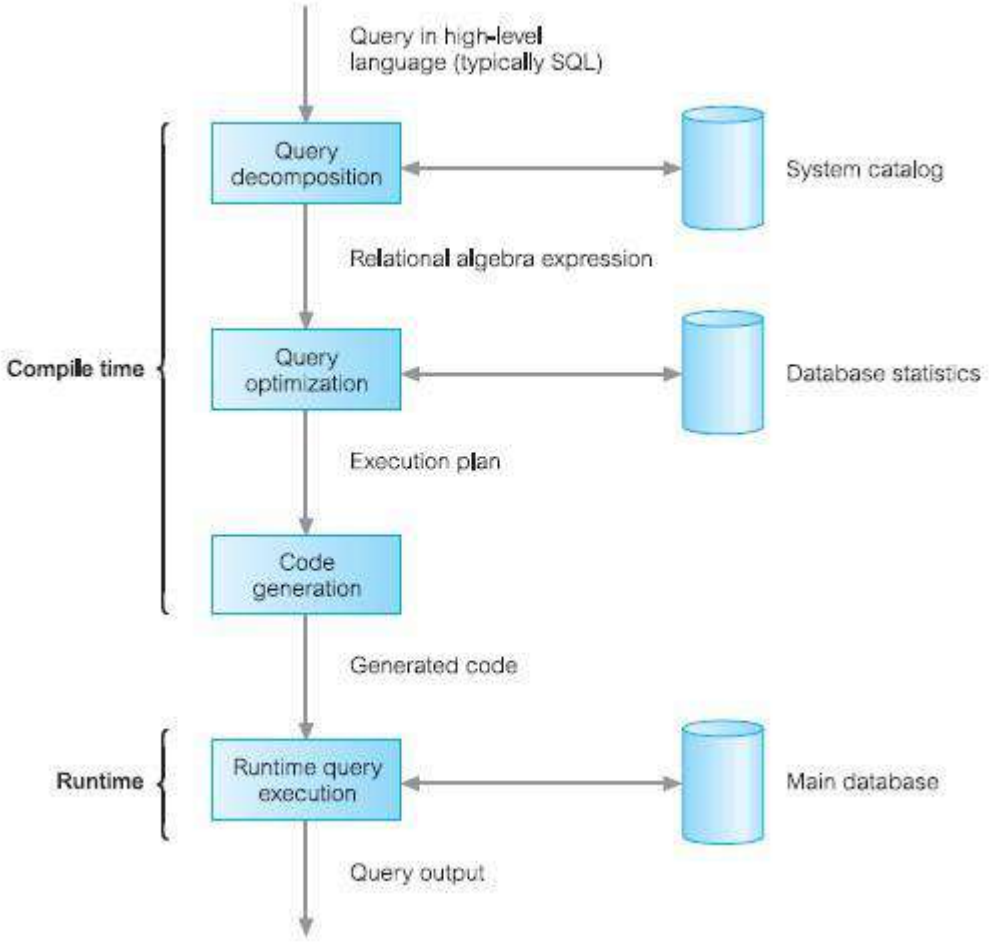
It contains all the meta data of database. Everything about data is stored in system catalog. For example; names of databases, tables in databases, columns in tables, data type of columns, primary/foreign columns etc.

2- **Database Statistics**

Information about data, for example; how many male and female students. Number of students from Islamabad. Number of students who got c grade in DDS etc.

3- **Main Database**

Actual database where data residues.



Explanation of Diagram:

Mr. Abdul Rehman email id: abdul@northern.edu.pk Whatsapp# 03087792217

1- **Query Decomposition:**

Query is written in some high-level language which passes through first step known as "query decomposition". Query decomposition actually check whether query is syntactically correct or not, if syntax is correct then whether the credentials mentioned in query are correct or not. For example; if user enter incorrect table name. that is why it refers to system catalog to confirm table names and column names are correct or not.

2- **Query Optimization**

In this phase, shortest path to execute the query is selected. Certain rules are applied so that query can be executed in less time not effecting the final output of the query.

3- **Code Generation**

In this phase, executable code is generated so that it could be executed to fetch results.

4- **Run-Time query execution**

In this phase, query is executed that is why main database is accessed where data exists. So that data can be fetched and displayed to the user.

Query Decomposition

Query decomposition is the first phase of query processing. The aims of query decomposition are to transform a high-level query into a relational algebra query, and to check that the query is syntactically and semantically correct. The typical stages of query decomposition are analysis, normalization, semantic analysis, simplification, and query restructuring.

(1) Analysis

In this stage, the query is lexically and syntactically analyzed using the techniques of programming language compilers (see, for example, Aho and Ullman, 1977). In addition, this stage verifies that the relations and attributes specified in the query are defined in the system catalog. It also verifies that any operations applied to database objects are appropriate for the object type. For example, consider the following query:

SELECT staffNumber FROM Staff WHERE position > 10; This query would be rejected on two grounds:

i- In the select list, the attribute staffNumber is not defined for the Staff relation (should be staffNo) ii- In the WHERE clause, the comparison '>10' is incompatible with the data type

position, which is a variable character string.

On completion of this stage, the high-level query has been transformed into some internal representation that is more suitable for processing. The internal form that is typically chosen is some kind of query tree, which is constructed as follows:

Relational Tree Construction:

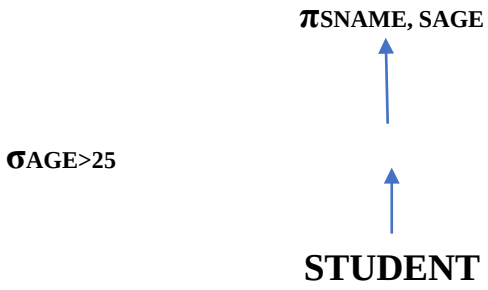
- A leaf node is created for each base relation in the query.
- A non-leaf node is created for each intermediate relation produced by a relational algebra operation.
- The root of the tree represents the result of the query.
- The sequence of operations is directed from the leaves to the root.

Example 1:

We have a relational Query

$$(\pi_{SNAME,\ SAGE}\ (\sigma_{AGE>25}\ (STUDENT)))$$

Relational Tree of above query will be

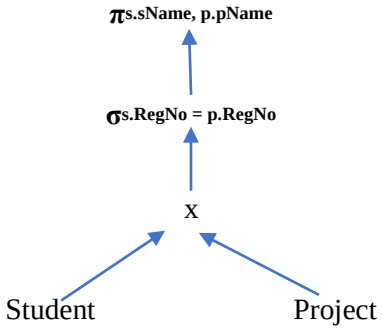


Above relational tree is representing the order of execution of query which is more visible in Query tree as compared to relational algebraic query.

Example 2:

$$(\pi_{s.sName,\ p.pName}\ (\sigma_{s.RegNo\ =\ p.RegNo}\ (Student \times Project)))$$ Relational

Tree of above query will be



In above query, first operation will be cartesian product of student and project, then sigma will be applied and at last pi will be applied.

(2) Normalization

The normalization stage of query processing converts the query into a normalized form that can be more easily manipulated.

- **Conjunctive normal form (AND operator between operations)**

  Conjunction means 'AND' in databases. Our moto is to transform user's query in such a form that all predicates (conditions) must have and operator between each other. In result, if a query has 10 conditions and one of them is false, we need not to apply others because in AND operator if one is false, overall answer of query is false.

  Example:

  (position = 'Manager' ∨ salary > 20000) ∧ branchNo = 'B003'

  Here we can see there are two main parts of query which have 'AND' between them,

  Left Part: (position = 'Manager' ∨ salary > 20000)

  Right Part: branchNo = 'B003'

  If answer of left part is 'FALSE', then we do not need to check right part because there is 'AND' operator between them.

- **Disjunctive normal form (OR operator between operations)**

  Disjunction means 'OR' in databases. This is opposite of Conjunction normal form. Our moto is to transform our query in such a format that all predicates(condition) must have 'OR' operator between them. In result, if one of the predicate is 'TRUE' then we do not need to check others because overall answer will be true.

Example:

(position = 'Manager' ∧ salary > 20000) ∨ branchNo = 'B003'

Here we can see there are two main parts of query which have 'OR' between them,

Left Part: (position = 'Manager' ∧ salary > 20000)

Right Part: branchNo = 'B003'

If answer of left part is 'TRUE', then we do not need to check right part because there is 'OR' operator between them.

(3) Semantic analysis

Semantic analysis checks whether the query is logically true or not. There may be queries which are syntactically correct but they are not logically true (which do not satisfy any rows in data).

For Example:

SELECT * FROM STUDENT WHERE AGE >25 AND AGE < 20

Above query is syntactically correct, but it cannot be true for any values. Its answer will always be empty set. So rather than checking the database, semantic analysis just checks the correctness of query and without checking database records, it sends a message to DBMS that query is logically false and no result will be shown. This process will ultimately save time for query execution.

Student

RegNo	sName	sAge	
101	NAUMAN	31	✘
102	SHAHID	32	✘
103	IMRAN	22	✘
104	AYESHA	29	✘

- If we apply above query on Student table, we can see that all rows are returning false on the given query.
- There cannot exist any data that will return true any row.

Normalize Attribute Connection Graph:

Above graph checks a query and tells if query is logically true or not. Attribute connection graph must be applied if query satisfies following conditions.

Mr. Abdul Rehman email id: abdul@northern.edu.pk Whatsapp# 03087792217

1- No or operator in query

- Graph constructed for only those predicates which have ○ '>' OR '<' operator in it ○ Integer Data
  ○ Predicates with string/float datatypes will be ignored **How to**

Create Attribute Connection Graph:

1- Create node for each distinct column in where clause of query.

2- Create node for a constant value and name it '0'.

3- Draw a directed edge from column node to constant node if predicate has '<' symbol and write value in predicate over the edge.

4- Draw a directed edge from constant node to column node if predicate has '>' symbol and write value in predicate with a negative sign over the edge.

5- Check for cycle in graph, if graph has cycle then

   a. Add values in cycle, if sum of values is <=1 then query is logically 'FALSe' else 'TRUE'

Block Diagram:

If (cycle)

{

Sum = add values in cycle If

(Sum <= 1)

Query is 'FALSE'

Else

Query is 'TRUE'

}

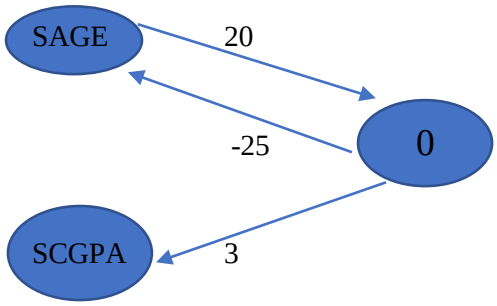
Else

Query is 'TRUE'

Example 1:

SELECT * FROM STUDENT WHERE SAGE > 25 AND SAGE < 20 AND SCGPA > 3 Graph:

Mr. Abdul Rehman email id: abdul@northern.edu.pk Whatsapp# 03087792217



Now, as we see there is a cycle between SAGE and Constant node, so we will sum these values, SUM = 20 + (-25) = -5

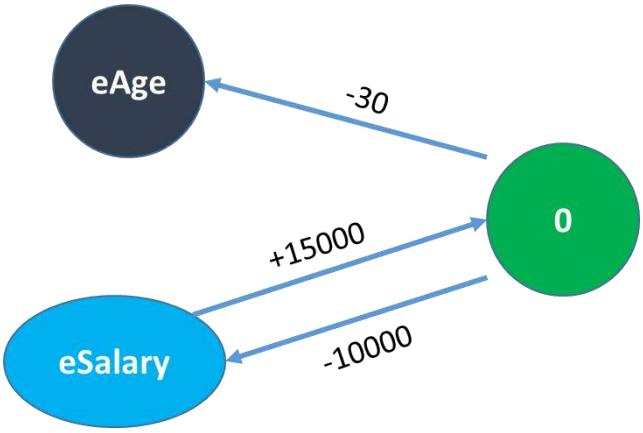
As SUM <= 1, so Query is Logically 'FALSE'.

NOTE: if multiple nodes has cycle between them, then each cycle is dealt individually. If any of cycle's sum is <=1 then whole query is 'FALSE'.

Example 2:

Select name, Age from Employee WHERE eName='Ahmad' AND eAge > 30 AND eSalary < 15000 AND eSalary > 10000

- As eName column has varchar datatype, so it will not be executed in the graph. - Graph will look like



- As there is cycle between eSalary and constant node.
- Sum = 15000 – 10000
- Sum = 5000
- As Sum is NOT <=1,

- QUERY is LOGICALLY TRUE

(4) Simplification

The objectives of the simplification stage are to detect redundant qualifications, eliminate common subexpressions, and transform the query to a semantically equivalent but more easily and efficiently computed form. Typically, access restrictions, view definitions, and integrity constraints are considered at this stage, some of which may also introduce redundancy. If the user does not have the appropriate access to all the components of the query, the query must be rejected. Assuming that the user has the appropriate access privileges, an initial optimization is to apply the well-known idempotency rules of boolean algebra, such as:

$p \wedge (p) \equiv p$	$p \vee (p) \equiv p$
$p \wedge false \equiv false$	$p \vee false \equiv p$
$p \wedge true \equiv p$	$p \vee true \equiv true$
$p \wedge (\sim p) \equiv false$	$p \vee (\sim p) \equiv true$
$p \wedge (p \vee q) \equiv p$	$p \vee (p \wedge q) \equiv p$

Distributed Database System (ECS-242/602)

Mr. Abdul Rehman **email id: abdul@northern.edu.pk Whatsapp# 03087792217**
