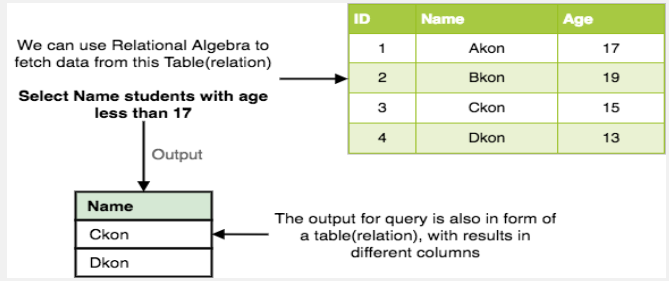
**Relational Algebra in DBMS**

Relational algebra is one of the two formal query languages associated with the relational model. Queries in algebra are composed using a collection of operators. A fundamental property is that every operator in the algebra accepts (one or two) relation instances as arguments and returns a relation instance as the result.  
This property makes it easy to compose operators to form a complex query a  
relational algebra expression is recursively defined to be a relation, a unary algebra operator applied to a single expression, or a binary algebra operator applied to two expressions. We describe the basic operators of the algebra (selection, projection, union, cross-product, and difference), as well as some additional operators that can be defined in terms of the basic operators but arise frequently enough to warrant special attention, in the following sections.

Each relational query describes a step-by-step procedure for computing the desired answer, based on the order in which operators are applied in the query. The procedural nature of the algebra allows us to think of an algebra expression as a recipe, or a plan, for evaluating a query, and relational systems in fact use algebra expressions to represent query evaluation plans.  
Every database management system must define a query language to allow users to access the data stored in the database. Relational Algebra is a procedural query language, which takes instances of relations as input and yields instances of relations as output from database tables to access data in different ways. It uses operators to perform queries. An operator can be  
Either unary or binary.  
In relational algebra, input is a relation (table from which data has to be accessed) and output is also a relation (a temporary table holding the data asked for by the user).



Relational Algebra works on the whole table at once, so we do not have to use loops etc to iterate over all the rows (tuples) of data one by one. All we have to do is specify the table name from which we need the data, and in a single line of command, relational algebra will traverse the entire given table to fetch data for you.  
Relational database systems are expected to be equipped with a query language that can assist its users to query the database instances. There are two kinds of query languages relational algebra and relational calculus.

**I. Unary Relational Operations**

**A. SELECT (symbol: σ (Sigma))**

This is used to fetch rows (tuples) from table (relation) which satisfies a given  
condition (predicate)?

Where, σ represents the Select Predicate, r is the name of relation (table name in which you want to look for data), and p is the prepositional logic, where we specify the conditions that must be satisfied by the data. In prepositional logic, one can use unary and binary operators

Like =, <, > etc, to specify the conditions.  
Let’s take an example of the Student table we specified above in the Introduction of relational algebra, and fetch data for students with age more than 17. **σage > 17 (Student)**

https://wachemo-elearning.net/wp-content/uploads/2022/07/image-197.png

This will fetch the tuples (rows) from table Student, for which age will be greater than 17.  
You can also use, and, or etc operators, to specify two conditions, for **example,** σage > 17 and gender = ‘Male’ (Student)

https://wachemo-elearning.net/wp-content/uploads/2022/07/image-198.png

This will return tuples(rows) from table Student with information of male students, of age more than 17.(Consider the Student table has an attribute Gender too.)  
example

Example :- σ topic = “Database” (COURSE)

https://wachemo-elearning.net/wp-content/uploads/2022/07/image-202.png

**Output –** Selects tuples from Tutorials where topic = ‘Database’.

**Example:-**σ topic = “Database” and author = “guru99″( COURSE )

https://wachemo-elearning.net/wp-content/uploads/2022/07/image-205.png

Output – Selects tuples from Tutorials where the topic is ‘Database’ and ‘author’ is guru99.  
**Example :-** σ sales > 50000 (Customers)

https://wachemo-elearning.net/wp-content/uploads/2022/07/image-207.png

**Output –** Selects tuples from Customers where sales is greater than 50000

**B. PROJECT (symbol: π (pi))**

Project operation is used to project only a certain set of attributes of a relation. In simple words, if you want to see only the names all of the students in the Student table, then you can use Project Operation.  
It will only project or show the columns or attributes asked for, and will also remove duplicate data from the columns. **Syntax:**∏A1, A2…(r)

Where A1, A2 etc are attribute names (column names).  
**For example:** ∏Name, Age(Student)  
Above statement will show us only the Name and Age columns for all the rows of data in Student table.  
It eliminates all attributes of the input relation but those mentioned in the projection list.  
The projection method defines a relation that contains a vertical subset of Relation.

**II.Relational Algebra Operations from Set Theory**

**A. UNION (υ):** This operation is used to fetch data from two relations (tables) or temporary relation (result of another operation).  
For this operation to work, the relations (tables) specified should have same number of attributes (columns) and same attribute domain. Also the duplicate tuples are automatically  
**Eliminated from the result**

**Syntax: A ∪ B**  
Where A and B are relations.

For example, if we have two tables RegularClass and ExtraClass, both  
have a column student to save name of student, then,  
**∏Student (RegularClass) ∪ ∏Student (ExtraClass)**

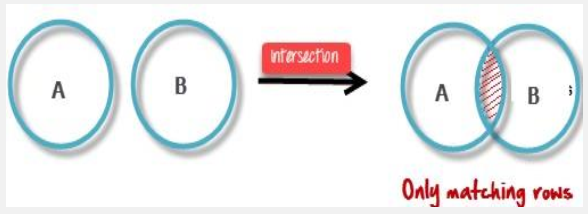
Above operation will give us name of Students who are attending both regular classes and extra classes, eliminating repetition.

UNION is symbolized by ∪ symbol. It includes all tuples that are in tables A or in B.  
It also eliminates duplicate tuples.

For a union operation to be valid, the following conditions must hold –  
• R and S must be the same number of attributes.  
• Attribute domains need to be compatible.  
• Duplicate tuples should be automatically removed

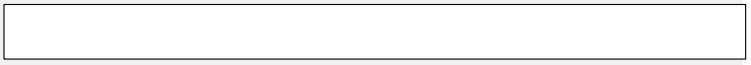
**B. INTERSECTION (∩)**

Defines a relation consisting of a set of all tuple that are in both A and B. However, A and B must be union-compatible



**C. DIFFERENCE (-)**

This operation is used to find data present in one relation and not present in the second relation. This operation is also applicable on two relations, just like  
Union operation.  
**Syntax:** A – B where A and B are relations.  
For example, if we want to find name of students who attend the regular class but not the extra class, then, we can use the below operation:  
**example :-** ∏Student(RegularClass) – ∏Student(ExtraClass)



The result of A – B, is a relation which includes all tuples that are in A but not in B.  
• The attribute name of A has to match with the attribute name in B.  
• The two-operand relations A and B should be either compatible or Union compatible.  
• It should be defined relation consisting of the tuples that are in relation A, but not in B.

**D. CARTESIAN P R O D U C T ( x)**

This i s u s e d t o c o m b i n e d a t a f r o m t w o d i f f e r e n t relations (tables) into one and fetch data from the combined relation.

**Syntax: A X B**  
For example, if we want to find the information for Regular Class and Extra Class which are conducted during morning, then, we can use the following operation: σtime = ‘morning’ (RegularClass X ExtraClass)

https://wachemo-elearning.net/wp-content/uploads/2022/07/image-218.png

For the above query to work, both RegularClass and ExtraClass should have the attribute time.  
This type of operation is helpful to merge columns from two relations. Generally, a Cartesian product is never a meaningful operation when it performs alone. However, it becomes meaningful when it is followed by other operations.

**III. Binary Relational Operations**

**A. JOIN**

Join operation is essentially a cartesian product followed by a selection criterion. Join operation denoted by ⋈.  
JOIN operation also allows joining variously related tuples from different  
relations

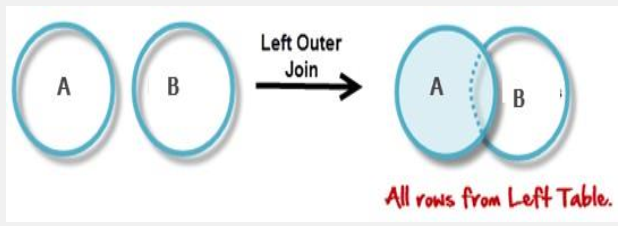
**Types of Joint**

Various forms of join operation are:

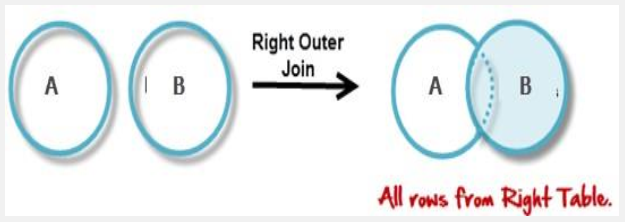
**Inner Joins**

In an**inner join**, only those tuples that satisfy the matching criteria are included, while the rest are excluded. Let’s study various types of Inner Joins:  
Theta join: The general case of JOIN operation is called a Theta join. It is denoted by**symbol θ**

1. **EQUI join**: When a theta join uses only equivalence condition, it becomes a equi join.  
   It is the most difficult operations to implement efficiently in an RDBMS and one reason why RDBMS have essential performance problems.  
   **Natural join (⋈)**: Natural join can only be performed if there is a common attribute (column) between the relations. The name and type of the attribute must be same.
2. **Outer join:** In an outer join, along with tuples that satisfy the matching criteria, we also include some or all tuples that do not match the criteria.
3. **Left Outer Join (A B):**In the left outer join, operation allows keeping all tuple in the left relation. However, if there is no matching tuple is found in right relation, then the attributes of right relation in the join result are filled with null values.

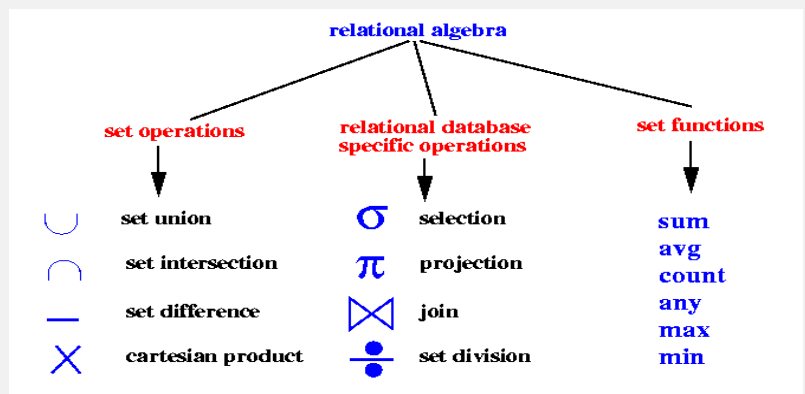


**5.Right Outer Join (A B):** In the right outer join, operation allows keeping all tuple in the right relation. However, if there is no matching tuple is found in the left relation, then the attributes of the left relation in the join result are filled with null values.



**6. Full Outer Join:** In a full outer join, all tuples from both relations are  
included in the result, irrespective of the matching condition

Database performance tuning often requires a deeper understanding of  
how queries are processed and optimized within the database management  
system. In this note we provide a general overview of how query processing



(rule based and cost-based query optimizers operate) and then provide some specific examples of query optimization in commercial DBMS.  
In this session we discuss the techniques used by DBMS to process, optimize and execute high- level queries. A query expressed in high level query language such as SQL must first be scanned, parsed and validated.  
The scanner identifies the language tokens such as Keywods, attribute names, and relation names whereas parser checks the query syntax to determine whether it is formulated according to the syntax rules (rules of grammar) of the query language.  
The query must be validated, by checking that all attribute and relation names are valid and semantically meaningful names in the schema of particular database being queried. An internal representation of the query is then created as a tree data structure called query tree.  
It is also possible to represent the query using graph data structure called query graph.  
The DBMS must then device execution strategy for retrieving the result of query from the database. A query typically has many possible execution strategies and process of choosing a suitable one for processing a query is known as query optimization.

1.**3. Query-processing**

The system executes the query using the optimal strategy generated. In query-optimization, a SQL query is first translated into an equivalent relational algebra expression using a query tree data structure before to be optimized.  
We will therefore set out below how to pass from a SQL query to an expression in Relational Algebra. Query is processed in two phases: the query-optimization phase and the query-processing phase. In order to facilitate the understanding, we will add the query compilation phase before the two previous phases because queries are viewed by user as  
**Data Manipulation Language (DML) scripts.**

1.**3.1. Query-compilation:** DML processor translates DML statements into low-level instructions (compiled query) that the query optimizer can understand.

**1.3.1.1. Translating SQL queries into Relational Algebra**  
Typically, queries are decomposed into query blocks which form the basic unit that can be translated into the algebraic operators and optimized. A query block contains a single SELECT- FROM- WHERE expression as well as GROUP BY and HAVING clause if these are part of the block. It can also include aggregate operators such as MAX, COUNT, MIN and SUM. These operators must also include in the extended algebra.  
• Query optimization is a function of many relational database management systems. The query optimizer attempts to determine the most efficient way to execute a given query by considering the possible query plans. Generally, the query optimizer cannot be accessed directly by users: once queries are submitted to database server, and parsed by the parser, they are then passed to the query optimizer where optimization occurs. It aims to choose an efficient execution strategy for query execution.  
• The optimizer automatically generates a set of reasonable strategies for processing a given query, and selects one optimal on the basis of the expected cost of each of the strategies generated.

Query Processing would mean the entire process or activity which involves query translation into low level instructions, query optimization to save resources, cost estimation or evaluation of query, and extraction of data from the database.  
Goal: To find an efficient Query Execution Plan for a given SQL query which would minimize the cost considerably, especially time.

**Cost Factors:** Disk accesses [which typically consumes time], read/write operations [which typically needs resources such as memory/RAM].  
The programmer write code to perform the queries with higher level database query languages such as SQL and a special component of the DBMS called the Query Processor takes care of arranging the underlying access routines to satisfy a given query.  
A query is processed in the following four general steps:

1. **Scanning and Parsing**  
   When a query is first submitted (via an applications program), it must be scanned and parsed to determine if the query consists of appropriate syntax. Scanning is the process of converting the query text into a tokenized representation. The tokenized representation is more compact and is suitable for processing by the parser.  
   This representation may be in a tree form. The Parser checks the tokenized representation for correct syntax. In this stage, checks are made to determine if columns and tables identified in the query exist in the database and if the query has been formed correctly with the appropriate keywords and structure. If the query passes the parsing checks, then it is passed on to the Query Optimizer.
2. **Query Optimization or planning the execution strategy**  
   For any given query, there may be a number of different ways to execute it. Each operation in the query (SELECT, JOIN, etc.) can be implemented using one or more different Access Routines.  
   For example, an access routine that employs an index to retrieve some rows would be more efficient that an access routine that performs a full table scan.  
   The goal of the query optimizer to find a reasonably efficient strategy for executing the query (not quite what the name implies) using the access routines. Optimization typically takes one of two forms:  
   Heuristic Optimization or Cost Based Optimization  
   In Heuristic Optimization, the query execution is refined based on heuristic rules for reordering the individual operations. With Cost Based Optimization, the overall cost of executing the query is systematically reduced by estimating the costs of executing several different execution plans.
3. **Query Code Generator (interpreted or compiled)**  
   Once the query optimizer has determined the execution plan (the specific ordering of access routines), the code generator writes out the actual access routines to be executed.

With an interactive session, the query code is interpreted and passed directly to the runtime database processor for execution. It is also possible to compile the access routines and store them for later execution.

1. **Execution in the runtime database processor**  
   At this point, the query has been scanned, parsed, planned and (possibly) compiled. The runtime database processor then executes the access routines against the database. The results are returned to the application that made the query in the first place. Any runtime errors are also returned.  
   The major steps involved in query processing are depicted in the figure below;

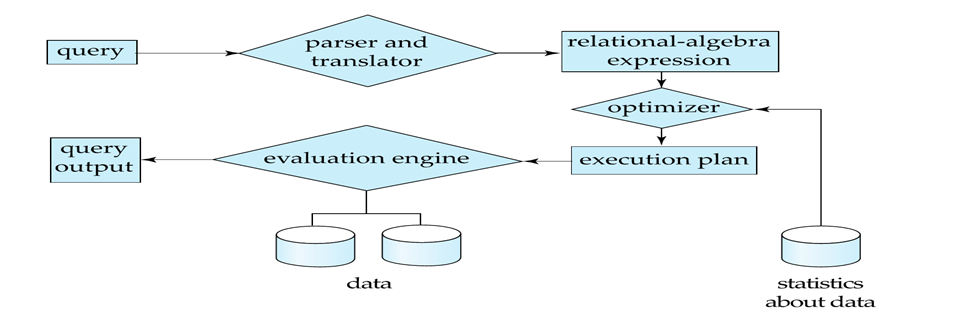


Figure 1. 2 Steps in Database Query Processing

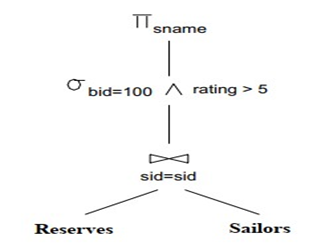
Let us discuss the whole process with an example. Let us consider the following two relations as the example tables for our discussion;  
Employee (Eno, Ename, Phone) Proj\_Assigned(Eno, Proj\_No, Role, DOP)  
Where,  
Eno is Employee number, Ename is Employee name,  
Proj\_No is Project Number in which an employee is assigned; Role is the role of an employee in a project,  
DOP is duration of the project in months.  
With this information, let us write a query to find the list of all employees who are working in a project which is more than 10 months old.  
SELECT Ename FROM Employee, Proj\_Assigned WHERE Employee. Eno = Proj\_Assigned. Eno AND DOP > 10;  
Input:  
A query written in SQL is given as input to the query processor. For our case, let us consider the SQL query written above.

**Step 1: Parsing**  
In this step, the parser of the query processor module checks the syntax of the query, the user’s privileges to execute the query, the table names and attribute names, etc. The correct table names attribute names and the privilege of the users can be taken from the system catalog (data dictionary).  
**Step 2: Translation**  
If we have written a valid query, then it is converted from high level language SQL to low level instruction in Relational Algebra.  
For example, our SQL query can be converted into a Relational Algebra equivalent as follows;  
πEname(σDOP>10 Λ Employee.Eno=Proj\_Assigned.Eno(Employee X Prof\_Assigned))

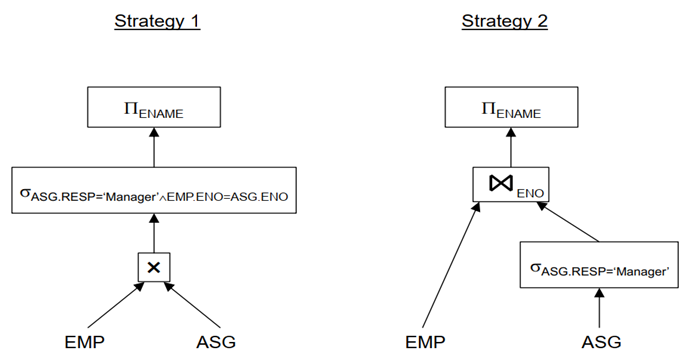
**Step 3: Optimizer**  
Optimizer uses the statistical data stored as part of data dictionary. The statistical data are information about the size of the table, the length of records, the indexes created on the table, etc. Optimizer also checks for the conditions and conditional attributes which are parts of the query.  
Step 4: Execution Plan  
A query can be expressed in many ways. The query processor module, at this stage, using the information collected in step 3 to find different relational algebra expressions that are equivalent and return the result of the one which we have written already.  
For our example, the query written in Relational algebra can also be written as the one given below;  
πEname (Employee ⋈Eno (σDOP>10 (Prof\_Assigned)))  
So far, we have got two execution plans. Only condition is that both plans should give the same  
Result

**Step 5: Evaluation**  
Though we got many execution plans constructed through statistical data, though they return same result (obvious), they differ in terms of Time consumption to execute the query,

or the Space required executing the query. Hence, it is mandatory to choose one plan which obviously consumes less cost.  
At this stage, we choose one execution plan of the several we have developed. This Execution plan accesses data from the database to give the final result.  
In our example, the second plan may be good. In the first plan, we join two relations (costly operation) then apply the condition (conditions are considered as filters) on the joined relation. This consumes more time as well as space.  
In the second plan, we filter one of the tables (Proj\_Assigned) and the result is joined with the Employee table. This join may need to compare less number of records. Hence, the second plan is the best (with the information known, not always).  
Example: See the following schema  
Sailors (sid: integer, sname: string, rating: integer, age: real) Reserves (sid: integer, bid: integer, day: dates, rname: string)  
Reserves:  
Each tuple is 40 bytes long  
100 tuples per page 1 00 pages. Sailors:  
Each tuple is 50 bytes long  
80 tuples per page  
500 pages.  
Consider the following SQL query:  
SELECT S.sname FROM Reserves R, Sailors S WHERE R.sid = S.sid AND R.bid = 100 AND S.rating > 5  
This query can be expressed in relational algebra (RA) as follows:  
πsname (σbid=100∧rating>5 (Reserves sid=sid Sailors))  
The algebraic expression partially specifies how to evaluate the query:  
✓ Compute the natural join of Reserves and Sailors  
✓ Perform the selections  
✓ Project the sname field  
The RA Tree

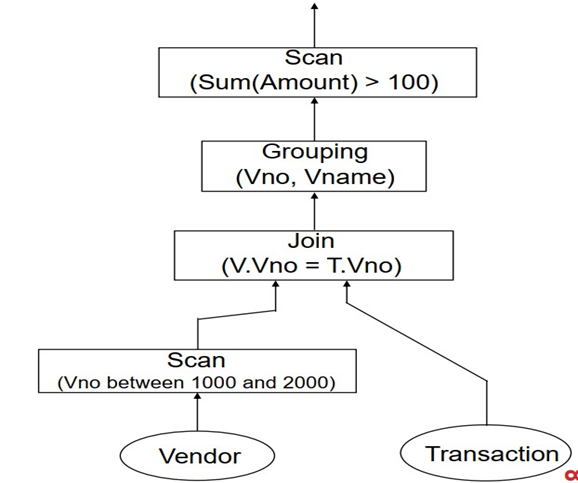


Query Expressed as a Relational Algebra Tree



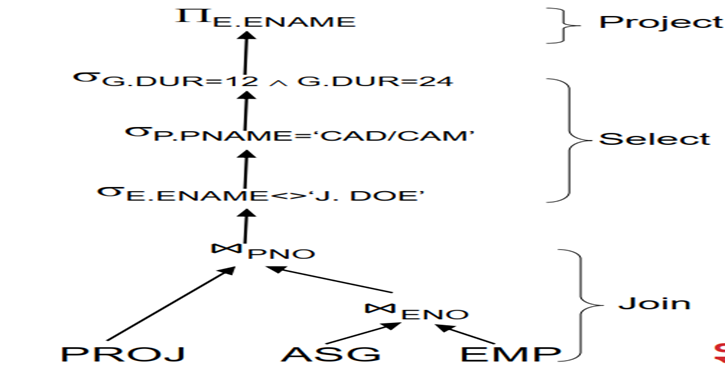
**EXAMPLE**  
SELECT V,Vno, Vname, count(\*), sum(Amount) FROM Vendor V, Transaction T WHERE V,Vno=T,Vno AND V,Vno berween 1000 AND 2000 GROUP BY V,Vno, Vname HAVING sum(Amount)>100  
➢ Scan the Vendor table, select all tuples where Vno = [1000, 2000], eliminate attributes other than Vno and Vname, and place the result in a temporary relation R1  
➢ Join the tables R1 and Transaction, eliminate attributes other than Vno, Vname, and Amount,  
and place the result in a temporary relation R2. This may involve:  
✓ sorting R1 on Vno

✓ sorting Transaction on Vno  
✓ merging the two sorted relations to produce R2  
➢ perform grouping on R2, and place the result in a temporary relation R3. This may involve:  
✓ sorting R2 on Vno and Vname  
✓ grouping tuples with identical values of Vno and Vname  
✓ counting the number of tuples in each group, and adding their Amounts  
➢ Scan R3, select all tuples with sum(Amount) > 100 to produce the result.

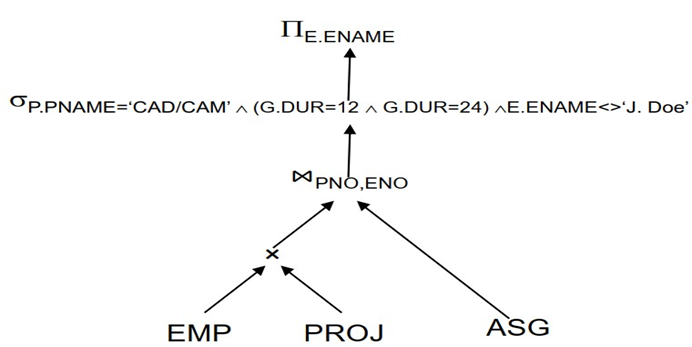


Find the names of employees other than J. Doe who worked on the CAD/CAM project for either one or two years  
SELECT ENAME FROM PROJ P, ASG G, EMP E WHERE G.ENO=E.ENO AND

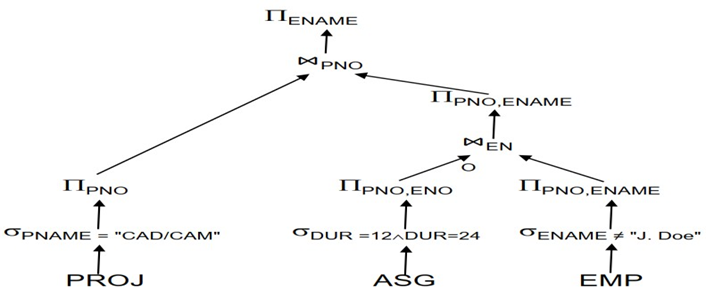
G.PNO=P.PNO AND E.ENAME<>’J.Doe’ AND P>PNAME=’CAD/CAM’ AND (G>DUR=12  
OR G.DUR=24)



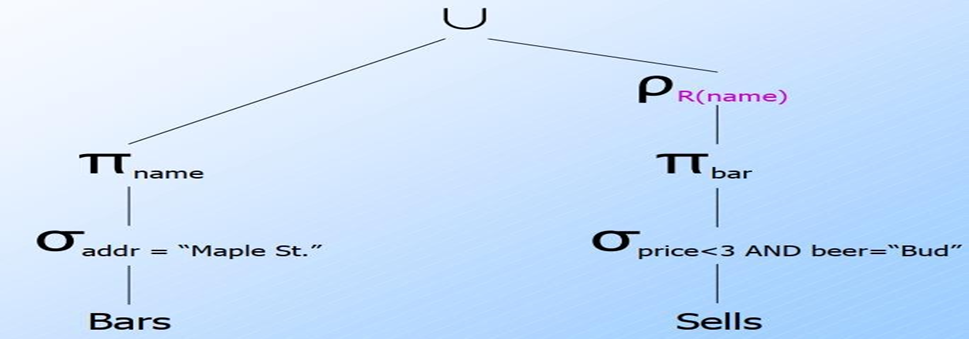
EQUIVALENT QUERY



ANOTHER EQUIVALENT QUERY



Example: Tree for a Query Using the relations Bars(name, addr) and Sells(bar, beer, price), find the names of all the bars that are either on Maple St. or sell Bud for less than $3

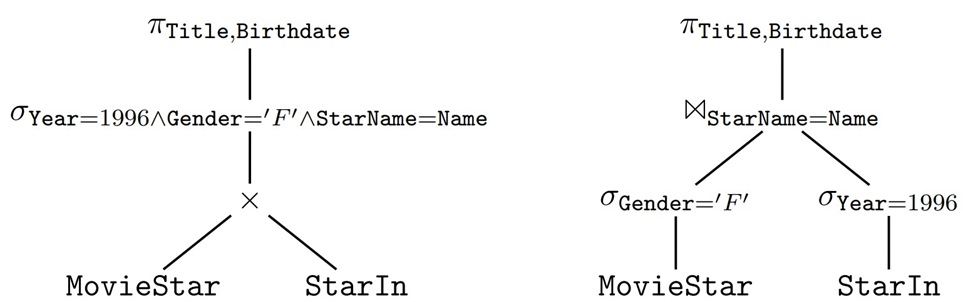


**Expression Trees: Example.**

MovieStar(Name,Address,Gender,Birthdate) StarIn(Title,Year,StarName)

Query: “*Find the birthdate and the movie title for those female stars who appeared in movies in* 1996.

**SELECT**Title, Birthdate **FROM**MovieStar, StarIn **WHERE**Year=1996 **AND**Gender=’F’ **ND**Name=StarName;



Example. StarIn (Title, Year, StarName).  
SELECT StarName, MIN(Year) as minYear FROM StarIn GROUP BY StarName HAVING COUNT(Title)>=3;

