Relational Algebra in DBMS

Relational Algebra is a procedural query language. Relational algebra mainly provides a theoretical foundation for relational databases and SQL. The main purpose of using Relational Algebra is to define operators that transform one or more input relations into an output relation. Given that these operators accept relations as input and produce relations as output, they can be combined and used to express potentially complex queries that transform potentially many input relations (whose data are stored in the database) into a single output relation (the query results). As it is pure mathematics, there is no use of English Keywords in Relational Algebra and operators are represented using symbols.

Fundamental Operators

These are the basic/fundamental operators used in Relational Algebra.

1. Selection(σ)
2. Projection(π)
3. Union(U)
4. Set Difference(-)
5. Set Intersection(∩)
6. Rename(ρ)
7. Cartesian Product(X)

1. Selection (σ)

It is used to select required tuples of the relations. The selection operation works on a single relation R and defines a relation that contains only those tuples of R that satisfy the specified condition (Predicate).

Syntax of Selection Operation

The syntax of selection operation is: **σ** Predicate (R). Here R refers to relation and predicate refers to
condition.

Example:

Name	Department	Salary
Peter	IT	4000
Jane	Finance	3000
Mary	HR	3000
John	Accounts	4000

Question: List the Name, Department, and Salary of those employees who are having a salary of more than 3000.

Answer: Query expressed in relational algebra as: **σ(Salary>3000)R**

Name	Department	Salary
Peter	IT	4000
John	Accounts	4000

Note: The selection operator only selects the required tuples but does not display them. For display, the data projection operator is used.

2. Projection(π)

It is used to project required column data from a relation. The projection operation works on a single relation R and defines a relation that contains a vertical subject of R, extracting the values of specified attributes and elimination duplicates. The projection operation can be considered as column wise filtering.

Syntax of Projection Operation

The syntax of projection operation is given by**: π** a**1**, a**2** ...a**n** (R). Where a**1**, a**2** . . . a**n** are attributes and R
stands for relation.

Example: Consider the relation Staff, with the attributes Staff_No, Name,
Gender, DOB, and Salary.

Question: Produce the list of salaries for all staff showing only the Name and salary detail.

Answer: Relational algebra expression: **π**Name, Salary (Staff)

Note: By Default, projection removes duplicate data.

3. Union(U)

Union operation in relational algebra is the same as union operation in set theory. The union of two relations R and S defines a relation that contains all the tuples of R or S or both R and S, duplicate tuples being eliminated.

Relational Algebra Expression

The union of two relations R and S are denoted by RUS

Example:

FRENCH

Student_Name	Roll_Number
Ram	01
Mohan	02
Vivek	13
Geeta	17

GERMAN

Student_Name	Roll_Number
Vivek	13
Geeta	17
Shyam	21
Rohan	25

Consider the following table of Students having different optional subjects in their course.

π(Student_Name)FRENCH U π(Student_Name)GERMAN

Student_Name
Ram
Mohan
Vivek
Geeta
Shyam
Rohan

Note: The only constraint in the union of two relations is that both relations must have the same set of Attributes.

4. Set Difference(-)

Set Difference in relational algebra is the same set difference operation as in set theory. The set difference operation defines a relation consisting of the tuples that are in relation R but not in S.

Relational Algebra Expression

The difference between two relations R and S is denoted by R – S. It results in finding tuples that are in
one relation but are not in another.

Example: From the above table of FRENCH and GERMAN, Set Difference is used as follows

π(Student_Name)FRENCH - π(Student_Name)GERMAN

Student_Name
Ram
Mohan

Note: The only constraint in the Set Difference between two relations is that both relations must have the same set of Attributes.

5. Set Intersection(∩)

Set Intersection in relational algebra is the same set intersection operation in set theory. The intersection operation defines a relation consisting of the set of all tuples that are in both R and S.

Relational Algebra Expression

The intersection of two relations R and S is denoted by R ∩ S.

Example: From the above table of FRENCH and GERMAN, the Set Intersection is used as follows

π(Student_Name)FRENCH $\cap$ π(Student_Name)GERMAN

Student_Name
Vivek
Geeta

Note: The only constraint in the Set Difference between two relations is that both relations must have the same set of Attributes.

6. Rename(ρ)

Rename is a unary operation used for renaming attributes of a relation.

ρ(a/b)R will rename the attribute 'b' of the relation by 'a'.

7. Cross Product(X)

Cross-product between two relations. Let's say A and B, so the cross product between A X B will result in all the attributes of A followed by each attribute of B. Each record of A will pair with every record of B.

Example:

Relation **A**

Name	Age	Sex
Ram	14	M
Sona	15	F
Kim	20	M

Relation **B**

ID	Course
1	DS
2	DBMS

A X B

Name	Age	Sex	ID	Course
Ram	14	M	1	DS
Ram	14	M	2	DBMS
Sona	15	F	1	DS
Sona	15	F	2	DBMS

Name	Age	Sex	ID	Course
Kim	20	M	1	DS
Kim	20	M	2	DBMS

Note: If A has ‘n’ tuples and B has ‘m’ tuples then A X B will have ‘ n*m ‘ tuples.

Derived Operators

These are some of the derived operators, which are derived from the fundamental operators.

1. Natural Join(⋈)
2. Conditional Join

1. Natural Join(⋈)

Natural join is a binary operator. Natural join between two or more relations will result in a set of all combinations of tuples where they have an equal common attribute. The natural join performs an equi join of the two relations R and S over all common attributes. One occurrence of each common attribute is eliminated from the result. In other words a natural join will remove duplicate attribute. In most systems a natural join will require that the attributes have the same name to identity the attributes to be used in the join. This may require a renaming mechanism. Even if the attributes do not have same name, we can perform the natural join provided that the attributes should be of same domain.

Example:

Relation EMPLOYEE

Name	ID	Dept_Name
A	120	IT
B	125	HR
C	110	Sales
D	111	IT

Relation DEPT

Dept_Name	Manager
Sales	Y
Production	Z
IT	A

Natural join between EMP and DEPT with condition:

EMP.Dept_Name = DEPT.Dept_Name

EMP ⋈ DEPT

Name	ID	Dept_Name	Manager
A	120	IT	A

Name	ID	Dept_Name	Manager
C	110	Sales	Y
D	111	IT	A

2. **Conditional Join**

Conditional join works similarly to natural join. In natural join, by default condition is equal between common attributes while in conditional join we can specify any condition such as greater than, less than, or not equal.

Example:

Relation R

ID	Sex	Marks
1	F	45
2	F	55
3	F	60

Relation S

ID	Sex	Marks
10	M	20
11	M	22
12	M	59

Join between R and S with condition **R.marks >= S.marks**

R.ID	R.Sex	R.Marks	S.ID	S.Sex	S.Marks
1	F	45	10	M	20
1	F	45	11	M	22
2	F	55	10	M	20
2	F	55	11	M	22
3	F	60	10	M	20
3	F	60	11	M	22
3	F	60	12	M	59

3. **Equi Join**

A special case of condition joins where the condition C contains only equality. There is duplication of the attributes common in both relations.

4. **Theta Join**

A conditional join in which we impose condition other than equality condition. If equality condition is imposed then theta join become equi join. The symbol **θ** stands for the comparison operator which could be **>, <, >=, <=.**

Expression of Theta Join:

σθ(R × S**)**

It combines the attributes of both relations.

5. **Outer Join**

In outer join, matched pairs are retained unmatched values in other tables are left null.

Types of Outer Join

i. **Left Outer Join.** Left outer joins is a join in which tuples from R that do not have matching values in the common column of S are also included in the result relation.

ii. **Right Outer Join**. Right outer join is a join in which tuples from S that do not have matching values in the common column of R are also included in the result relation.

iii. **Full Outer Join.** Full outer join is a join in which tuples from R that do not have matching values in the common columns of S still appear and tuples in S that do not have matching values in the common columns of R still appear in the resulting relation.

6. **Semi-Join**

The semi-join of a relation R, defined over the set of attributes A, by relation S, defined over the set of attributes B, is the subset of the tuples of R that participate in the join of R with S. The advantage of semi-join is that it decreases the number of tuples that need to be handled to form the join. In centralized database system, this is important because it usually results in a decreased number of secondary storage accesses by making better use of the memory. It is even more important in distributed databases, since it usually reduces the amount of data that needs to be transmitted between sites in order to evaluate a query.

Expression for Semi-Join

R**α F** S =**Π** A(R **αF** S) where F is the predicate.

Example of Semi-Join

In order to understand semi-join consider two relations EMPLOYEE and PAY. The semi-join of EMPLOYEE with the PAY is denoted by:

EMPLOYEE **α** EMPLOYE.DESIGNATION=PAY.DESIGNATION PAY.

Relational Calculus in DBMS

As Relational Algebra is a procedural query language, Relational Calculus is a non-procedural query language. It basically deals with the end results. It always tells me what to do but never tells me how to do it.
The major difference between relational calculus and relational algebra is summarized as:

- A relational calculus query specifies ***what*** information is retrieved
- A relational algebra query specifies ***how*** information is retrieved

There are two types of Relational Calculus

1. Tuple Relational Calculus(TRC)
2. Domain Relational Calculus(DRC)

1. Tuple Relational Calculus (TRC) in DBMS

Tuple Relational Calculus (TRC) is a non-procedural query language used in relational database management systems (RDBMS) to retrieve data from tables. TRC is based on the concept of tuples, which are ordered sets of attribute values that represent a single row or record in a database table.
TRC is a declarative language, meaning that it specifies what data is required from the database, rather than how to retrieve it. TRC queries are expressed as logical formulas that describe the desired tuples.
Syntax: The basic syntax of TRC is as follows:

```
{ t | P(t) }
```

where t is a **tuple variable** and P(t) is a **logical formula** that describes the conditions that the tuples in the result must satisfy. The **curly braces {}** are used to indicate that the expression is a set of tuples.
For example, let's say we have a table called "Employees" with the following attributes:

Employee ID
Name
Salary
Department ID

To retrieve the names of all employees who earn more than $50,000 per year, we can use the following TRC query:

{ t | Employees(t) ∧ t.Salary > 50000 }

In this query, the "Employees(t)" expression specifies that the tuple variable t represents a row in the "Employees" table. The "∧" symbol is the logical AND operator, which is used to combine the condition "t.Salary > 50000" with the table selection.

The result of this query will be a set of tuples, where each tuple contains the Name attribute of an employee who earns more than $50,000 per year.

TRC can also be used to perform more complex queries, such as joins and nested queries, by using additional logical operators and expressions.

While TRC is a powerful query language, it can be more difficult to write and understand than other SQL-based query languages, such as Structured Query Language (SQL). However, it is useful in certain applications, such as in the formal verification of database schemas and in academic research.
Tuple Relational Calculus is a **non-procedural query language,** unlike relational algebra. Tuple Calculus provides only the description of the query but it does not provide the methods to solve it. Thus, it explains what to do but not how to do it.

Tuple Relational Query

In Tuple Calculus, a query is expressed as

$$\{t| P(t)\}$$

where t = resulting tuples, P(t) = known as Predicate and these are the conditions that are used to fetch t. Thus, it generates a set of all tuples t, such that Predicate P(t) is true for t.

P(t) may have various conditions logically combined with OR (∨), AND (∧), NOT(¬).
It also uses quantifiers:
∃ t ∈ r (Q(t)) = ”there exists” a tuple in t in relation r such that predicate Q(t) is true.
∀ t ∈ r (Q(t)) = Q(t) is true “for all” tuples in relation r.

Quantifiers are words that refer to quantities such as “some” or “all” and tell for how many elements a given predicate is true. A predicate is a sentence that contains a finite number of variables and becomes a statement when specific values are substituted for the variables.
Quantifiers can be broadly classified into two types: (1) Universal Quantifier and (2) Existential Quantifier.

i. Universal Quantifier

The universal quantifier (∀) indicates that a predicate is true for all elements in a given domain.

Example: ∀x∈N, P(x)

Translation: “For all natural numbers xxx, xxx is even.”

ii. Existential Quantifier

The existential quantifier (∃) indicates that there exists at least one element in a given domain for which the predicate is true.

Example: ∃x∈N, P(x)

Translation: “There exists a natural number x such that x is even.”

2. Domain Relational Calculus in DBMS

Domain Relational Calculus is a non-procedural query language equivalent in power to Tuple Relational Calculus. Domain Relational Calculus provides only the description of the query but it does not provide the methods to solve it. In Domain Relational Calculus, a query is expressed as,

$$\{ < x1, x2, x3, ..., xn > | P (x1, x2, x3, ..., xn) \}$$

where, < x1, x2, x3, …, xn> represents resulting domains variables and P (x1, x2, x3, …, xn) represents the condition or formula equivalent to the Predicate calculus.

Predicate Calculus Formula:

1. Set of all comparison operators
2. Set of connectives like and, or, not
3. Set of quantifiers

Example:

Table-1: Customer

Cust_Name	Street	City
Deb	K	Nkr
Saya	U	Eld
Sou	N	Ksm
Ritu	J	Nrb

Table-2: Loan

Loan number	Branch name	Amount
L01	Main	200
L03	Main	150
L10	Sub	90
L08	Main	60

Table-3: Borrower

Cust_Name	Loan number
Ritu	L01
Deb	L08
Sou	L03

Query-1: Find the loan number, branch, amount of loans of greater than or equal to 100 amount.

$\{\prec l, b, a\succ \mid \prec l, b, a\succ \in loan \wedge (a \geq 100)\}$

Resulting relation:

Loan number	Branch name	Amount
L01	Main	200
L03	Main	150

Query-2: Find the loan number for each loan of an amount greater or equal to 150.

$\{\prec l\succ \mid \exists\ b, a\ (\prec l, b, a\succ \in loan \wedge (a \geq 150))\}$

Resulting relation:

Loan number
L01
L03

Query-3: Find the names of all customers having a loan at the “Main” branch and find the loan amount .

$\{\prec c, a\succ \mid \exists\ l\ (\prec c, l\succ \in borrower \wedge \exists\ b\ (\prec l, b, a\succ \in loan \wedge (b = \text{“Main”})))\}$

Resulting relation:

Cust_Name	Amount
Ritu	200
Deb	60
Sou	150

Note:
The domain variables those will be in resulting relation must appear before | within ≺ and ≻ and all the domain variables must appear in which order they are in original relation or table.