Relational Algebra in DBMS

Relational Algebra is a **procedural query language** that uses mathematical operations to manipulate relations (tables). It forms the foundation of SQL queries.

1. Selection (σ)

- **Purpose**: Selects rows (tuples) from a relation based on a condition.
- **Notation**: σ_{condition}(R)
- **Example**:
- $\sigma_{\text{City = 'Chennai'}}$ (Customer)

→ Retrieves all customers who live in Chennai.

2. Projection (π)

- **Purpose**: Selects specific columns (attributes).
- **Notation**: π_{attributes}(R)
- **Example**:
- $\pi_{\text{CustomerName, City}}$ (Customer)

→ Retrieves only names and cities of customers.

3. Union (∪)

- **Purpose**: Combines tuples from two relations (removes duplicates).
- **Requirement**: Both relations must be **union-compatible** (same attributes and domains).
- **Example**:
- $\pi_{\text{CustomerName}}$ (Customer_India) ∪ $\pi_{\text{CustomerName}}$ (Customer_USA)

→ Retrieves customers from either India or USA.

4. Set Difference (−)

- **Purpose**: Returns tuples present in one relation but not in the other.
- **Example**:
- π CustomerName ($_{\text{Customer\_India)}}$ – π CustomerName ($_{\text{Customer\_USA)}}$

→ Retrieves customers who are only from India but not from USA.

5. Intersection (∩)

- **Purpose**: Returns tuples present in both relations.
- **Example**:
- π CustomerName (Customer_India) ∩ π CustomerName (Customer_USA)

→ Retrieves customers who are in both India and USA tables.

6. Cartesian Product (×)

- **Purpose**: Combines all tuples of two relations.
- **Example**:
- Customer × Orders

→ Every customer is paired with every order (used as intermediate step for JOIN).

7. Rename (ρ)

- **Purpose**: Renames relation or attributes.
- **Notation**: ρ_{new-name}(R)
- **Example**:
- ρ C(Customer)

→ Renames Customer relation as C.

8. Join Operations

Joins combine related tuples from two relations.

a) Theta Join (⋈_{condition})

- Example:
- Customer $\bowtie_{\text{Customer.CustID = Order.CustID}}$ Order

→ Matches customers with their orders.

b) Equi Join

- A special case of Theta Join with = operator.

c) Natural Join (⋈)

- Joins on common attributes automatically.
- Example:
- Customer ⋈ Order

d) Outer Joins (Left, Right, Full)

- Keep non-matching tuples also.

9. Division (÷)

- **Purpose**: Finds tuples in one relation associated with **all tuples** in another.
- **Example**:
- R ÷ S

If R(Student, Course) and S(Course), this returns students who have taken **all courses in S**.

10. Assignment (←)

- **Purpose**: Stores result of a relational algebra expression into a variable.
- **Example**:
- Temp ← σ City = 'Chennai' (Customer)

Example with a Database Schema

Customer(CustomerID, Name, City)
Orders(OrderID, CustomerID, Amount)

Queries:

1. Get names of customers from Chennai:

π Name (σ City='Chennai' (Customer))

2. Find customers who have placed orders:

π Name (Customer $\bowtie_{\text{Customer.CustomerID = Orders.CustomerID}}$ Order)

3. Find customers who did not place any orders:

π Name (Customer) − π Name (Customer $\bowtie$ Orders)

4. Find customers who ordered all products in Product table:

(Customer × Product) ÷ Orders

Selection with More Conditions (σ)

The **Selection operator (σ)** retrieves rows (tuples) from a relation that satisfy a given **predicate** (condition).
You can combine multiple conditions using **logical operators**:

- **AND (∧)**
- **OR (∨)**
- **NOT (¬)**

1. Selection with AND (Conjunction)

Example:
Get customers who live in **Chennai AND** have CustomerID > 5.

σ (City = 'Chennai' ∧ CustomerID > 5) (Customer)

2. Selection with OR (Disjunction)

Example:
Get customers who live in **Chennai OR Delhi**.

σ (City = 'Chennai' ∨ City = 'Delhi') (Customer)

3. Selection with NOT (Negation)

Example:
Get customers who **do not live** in Chennai.

σ (¬(City = 'Chennai')) (Customer)

(or equivalently)

σ (City ≠ 'Chennai') (Customer)

4. Combined Complex Conditions

You can mix **AND, OR, NOT** together.

Example:
Get customers who live in **Chennai OR Bangalore**, but **NOT Delhi**, and have CustomerID < 50.

σ ((City = 'Chennai' ∨ City = 'Bangalore') ∧ City ≠ 'Delhi' ∧ CustomerID < 50) (Customer)

Practical Example

Customer(CustomerID, Name, City, Age)

CustomerID	Name	City	Age
1	Arun	Chennai	25
2	Deepa	Delhi	30
3	Kiran	Bangalore	28
4	Priya	Chennai	35
5	Mohan	Mumbai	22

Query 1: Customers in Chennai AND Age > 30

σ (City='Chennai' $\wedge$ Age>30) (Customer)

Output → Priya

Query 2: Customers in Chennai OR Bangalore

σ (City='Chennai' $\vee$ City='Bangalore') (Customer)

Output → Arun, Kiran, Priya

Query 3: Customers NOT in Delhi

σ (City $\neq$ 'Delhi') (Customer)

Output → Arun, Kiran, Priya, Mohan

Relational Algebra Aggregate Functions

Aggregate functions in **Relational Algebra** are used to perform calculations on a set of tuples and return a single value (or grouped values). They are similar to SQL's SUM, AVG, MIN, MAX, COUNT.

Notation

- **G** operator is used for grouping and aggregation.
- Syntax:

  G <grouping_attributes> g <aggregate_functions>(Relation)

- Example of aggregate functions:
  - COUNT(A) → number of tuples
  - SUM(A) → total of attribute A
  - AVG(A) → average of attribute A
  - MIN(A) → smallest value of attribute A
  - MAX(A) → largest value of attribute A

Example Schema

- **Employee(EmpID, Name, Dept, Salary)**

Examples

1. Find the average salary of all employees

- **Relational Algebra**:

G AVG(Salary)(Employee)

- **Explanation**: Groups the entire relation (no grouping attribute), computes the average of the Salary column.

2. Find the total salary paid in each department

- **Relational Algebra**:

  G Dept g SUM(Salary)(Employee)

- **Explanation**: Groups tuples by Dept, and for each group calculates the sum of Salary.

3. Find the number of employees in each department

- **Relational Algebra**:

  G Dept g COUNT(EmpID)(Employee)

- **Explanation**: Groups employees by Dept and counts the number of employees in each group.

4. Find the maximum salary in each department

- **Relational Algebra**:

  G Dept g MAX(Salary)(Employee)

- **Explanation**: Groups employees by department and finds the highest salary in each group.

5. Find the department with minimum average salary

- **Relational Algebra**:
- G Dept g AVG(Salary)(Employee)

then select the tuple with **MIN(AVG(Salary))**.

✅ **Key Point**:

- Without GROUP BY (no grouping attributes), aggregate is applied to the whole relation.
- With grouping attributes, aggregate is applied **per group**.

Schema Example

- **Customer(CustID, Name, City)**
- **Orders(OrderID, CustID, Product, Amount)**

1. Simple Selection

SQL:

SELECT Name, City

FROM Customer

WHERE City = 'Chennai';

Relational Algebra:

π Name, City (σ City='Chennai' (Customer))

Explanation: First select (σ) rows where City = 'Chennai', then project (π) Name and City.

2. Join

SQL:

SELECT Name, Product

FROM Customer C

JOIN Orders O ON C.CustID = O.CustID;

Relational Algebra:

π Name, Product (Customer ⋈ Customer.CustID = Orders.CustID Orders)

Explanation: Perform natural join (⋈) on matching CustID, then project Name and Product.

3. Customers who did not place orders (Set Difference)

SQL:

SELECT Name

FROM Customer

WHERE CustID NOT IN (SELECT CustID FROM Orders);

Relational Algebra:

π Name (Customer) − π Name (Customer ⋈ Orders)

Explanation: Get all customer names, subtract those who appear in Orders.

4. Aggregation

SQL:

```
SELECT CustID, SUM(Amount) AS TotalSpent
FROM Orders
GROUP BY CustID;
```

Relational Algebra:

G CustID g SUM(Amount)(Orders)

Explanation: Group by CustID and sum Amount.

5. Nested Query (Division Example)

SQL:

```
-- Find customers who ordered ALL products
SELECT CustID
FROM Orders O
WHERE NOT EXISTS (
  SELECT P.Product
  FROM Product P
  WHERE NOT EXISTS (
    SELECT 1
```

```
    FROM Orders O2
    WHERE O2.CustID = O.CustID AND O2.Product = P.Product
  )
);
```

Relational Algebra (using Division):

(π CustID, Product (Orders)) ÷ (π Product (Product))

Explanation: Division operator finds customers who ordered every product in the Product table.