विचार संगम

भारतीय दर्शन, विज्ञान, साहित्य-कला और तकनीकी के विशेष संदर्भ में

सम्पादक
डॉ. दिनेश चन्द्र शर्मा

विचार संगम

भारतीय दर्शन, विज्ञान, साहित्य-कला और तकनीकी के विशेष संदर्भ में

सम्पादक :

डॉ. दिनेश चन्द्र शर्मा

Ph.D., D.Lit. (Sub.)

निदेशक, कॉम्पिटिशन प्लस ग्रुप ऑफ इन्स्टीट्यूशन,

अजमेर

Books n Books

51/1509, आचमन, वेदविहार,

लोहाखान, अजमेर-305001

प्रकाशक :

पुष्पा शर्मा

Books n Books

51/1509, आचमन, वेदविहार,

लोहाखान, अजमेर-305001

ई-मेल : bnbajmer@gmail.com

संस्करण

प्रथम 2025

ISBN : 978-81-993854-0-5

मूल्य - 400/-

रचना - विचार संगम-भारतीय दर्शन, विज्ञान, साहित्य-कला और तकनीकी के विशेष संदर्भ में

सम्पादक - डॉ. दिनेश चन्द्र शर्मा

मुद्रक - श्रीनिवास प्रिन्टोग्राफिक्स, अजमेर

11

RELATIONAL ALGEBRA

Chitrawali Panwar
Asst. Professor (Department of Science and Technology)
Hukumchand Noble Institute of Science and Technology, Ajmer

Relational Algebra is a formal language used to query and manipulate relational databases, consisting of a set of operations like selection, projection, union, and join. It provides a mathematical framework for querying databases, ensuring efficient data retrieval and manipulation. Relational algebra serves as the mathematical foundation for query SQL

Relational algebra simplifies the process of querying databases and makes it easier to understand and optimize query execution for better performance. It is essential for learning SQL because SQL queries are based on relational algebra operations, enabling users to retrieve data effectively.

Key Concepts in Relational Algebra

Relational Algebra is like the mathematics behind databases. It helps us understand how to fetch, combine, and filter data from tables using logical rules. Before learning its operations, we must know some basic building blocks: relations, tuples, attributes, and domains.

1. Relations : (Think of it as a Table)

* In relational algebra, a relation is a table that consists of rows and columns, representing data in a structured format. Each relation has a unique name and is made up of tuples.

* A relation is simply a table that holds data in rows and columns.

* Each relation has a unique name, like Students, Courses, or Teachers.

* All the data in a relation follows the same structure.

Example:

Roll No	Name	Marks
1	Riya	92
2	Arjun	85
3	Neha	78

Here, Students is the relation - a table representing student data.

2. Tuples : (Think of it as a Row)

* A tuple is a single row in a relation, which contains a set of values for each attribute. It represents a single data entry or record in a relational table.

* A tuple is one single row in a table.

* It represents one record or one entry of data.

Example:

Roll No	Name	Marks
1	Riya	92

This single row is a tuple, showing Riya's information.

3. Attributes : (Think of it as a Column)

* Attributes are the columns in a relation, each representing a specific characteristic or property of the data. For example, in a "Students" relation, attributes could be "Name", "Age", and "Grade".

* Attributes are the headings of each column in the table.

* Each attribute describes a type of data, like Name, Marks, or Roll No.

Example:

In Students(RollNo, Name, Marks) - the attributes are Roll No, Name, and Marks.

These define what kind of information the table stores.

4. Domains : (Think of it as the Type of Values Allowed)

* A domain is the set of possible values that an attribute can have. It defines the type of data that can be stored in each column of a relation, such as integers, strings, or dates.

* A domain is the set of possible values each attribute can have.

For example:

* The domain of RollNo ? integers (1, 2, 3…)

* The domain of Name ? text strings (Riya, Arjun, Neha…)

* The domain of Marks ? numbers between 0 and 100

This ensures data consistency - you can't put a name where a

mark should be!

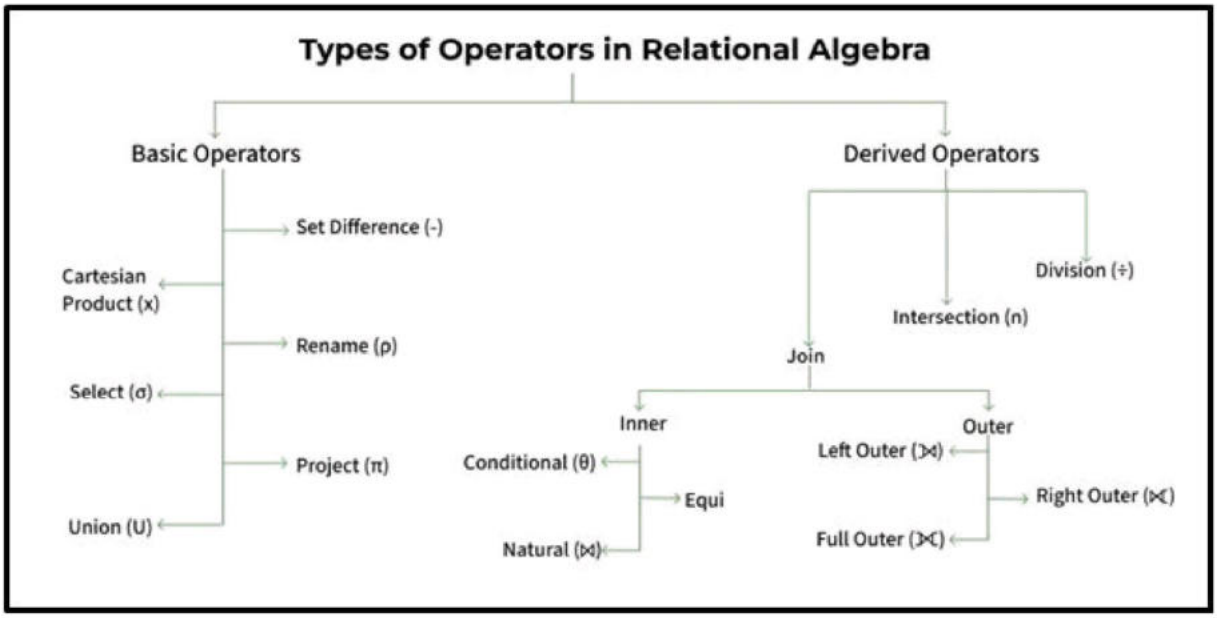
Easy Way to Remember (R-T-A-D):

Concept	Think of…	Example
Relation	Table	Students
Tuple	Row	Riya, 20, 92
Attribute	Column	Name, Age, Marks
Domain	Value Types	Integers, Strings, Numbers

Analogy: If your database is a school:

* **Relation** = Class register
* **Tuple** = Each student
* **Attribute** = What you record (Name, RollNo, Marks)
* **Domain** = Type of values allowed (Text, Numbers)

Relational algebra consists of various basic operators that help us to fetch and manipulate data from relational tables in the database to perform certain operations on relational data. Basic operators are fundamental operations that include selection (σ), projection (∏), union (U), set difference (–), Cartesian product (×), and rename (ρ).



Operators in Relational Algebra

1. Selection (σ) :

The Selection Operation is basically used to filter out rows from a given table based on certain given condition. It basically allows us to retrieve only those rows that match the condition as per condition passed during SQL Query.

Example: If we have a relation R with attributes A, B, and C, and we want to select tuples where C > 3, we write:

A	B	C
1	2	4
2	2	3
3	2	3
4	3	4

σ **(c > 3) (R)** will select the tuples which have c more than 3.

Output:

A	B	C
1	2	4
4	3	4

Explanation: The selection operation only filters rows but does not display or change their order. The projection operator is used for displaying specific columns.

2. Projection($\prod$) :

While Selection operation works on rows, similarly projection operation of relational algebra works on columns. It basically allows us to pick specific columns from a given relational table based on the given condition and ignoring all the other remaining columns.

Example: Suppose we want columns B and C from Relation R.

$\prod$ **(B, C) (R)** will show following columns.

Output:

B	C
2	4
2	3
3	4

Explanation: By Default, projection operation removes duplicate values.

3. Union(U) :

The Union Operator is basically used to combine the results of two queries into a single result. The only condition is that both queries must return same number of columns with same data types. Union operation in relational algebra is the same as union operation in set theory.

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

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

If **FRENCH** and **GERMAN** relations represent student names in two subjects, we can combine their student names as follows:

∏ (Student_Name) (FRENCH) U ∏ (Student_Name) (GERMAN)

Output:

Student_Name
Ram
Mohan
Vivek
Geeta
Shyam
Rohan

Explanation: 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 basically provides the rows that are present in one table, but not in another tables. Set Difference in relational algebra is the same set difference operation as in set theory.

Example: To find students enrolled only in FRENCH but not in GERMAN, we write:

π(Student_Name)(FRENCH) - π(Student_Name)(GERMAN)

Student_Name
Ram
Mohan

Explanation: The only constraint in the Set Difference

between two relations is that both relations must have the same set of Attributes.

5. Rename(ρ) :

Rename operator basically allows you to give a temporary name to a specific relational table or to its columns. It is very useful when we want to avoid ambiguity, especially in complex Queries. Rename is a unary operation used for renaming attributes of a relation.

Example: We can rename an attribute **B** in relation **R** to **D**

A	B	C
1	2	4
2	2	3
3	2	3
4	3	4

ρ **(D/B)R** will rename the attribute 'B' of the relation by 'D".

Output Table:

A	D	C
1	2	4
2	2	3
3	2	3
4	3	4

6. Cartesian Product(X) :

The **Cartesian product** combines every row of one table with every row of another table, producing all the possible combination. It's mostly used as a precursor to more complex operation like joins. 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.

Relation A:

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

Relation B:

ID	Course
1	DS
2	DBMS

Output: If relation **A** has 3 rows and relation **B** has 2 rows, the Cartesian product **A × B** will result in 6 rows.

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
Kim	20	M	1	DS
Kim	20	M	2	DBMS

Explanation: If A has 'n' tuples and B has 'm' tuples then A X B will have 'n*m' tuples.

Derived Operators in Relational Algebra

Derived operators are built using basic operators and include operations like join, intersection, and division. These operators help perform more complex queries by combining basic operations to meet specific data retrieval needs.

1. Join Operators :

Join operations in relational algebra combine data from two or more relations based on a related attribute, allowing for more complex queries and data retrieval. Different types of joins include:

Inner Join :

An inner join combines rows from two relations based on a matching condition and only returns rows where there is a match in both relations. If a record in one relation doesn't have a corresponding match in the other, it is excluded from the result. This is the most common type of join.

* **Conditional Join:** A conditional join is an inner join where the matching condition can involve any comparison operator like equals (=), greater than (>), etc. **Example:** Joining Employees and Departments on DepartmentID where Salary > 50000 will return employees in departments with a salary greater than 50,000

* **Equi Join:** An equi join is a type of conditional join where the condition is specifically equality (=) between columns from both relations. **Example:** Joining Customers and Orders on CustomerID where both relations have this column, returning only matching records.

* **Natural Join:** A natural join automatically combines

relations based on columns with the same name and type, removing duplicate columns in the result. It's a more efficient way of joining. Example: Joining Students and Enrollments where StudentID is common in both, and the result contains only unique columns.

Outer Join

An outer join returns all rows from one relation, and the matching rows from the other relation. If there is no match, the result will still include all rows from the outer relation with NULL values in the columns from the unmatched relation.

* **Left Outer Join:** A left outer join returns all rows from the left relation and the matching rows from the right relation. If there is no match, the result will include NULL values for the right relation's attributes. **Example:** Joining Employees with Departments using a left outer join ensures all employees are listed, even those who aren't assigned to any department, with NULL values for the department columns.

* **Right Outer Join:** A right outer join returns all rows from the right relation and the matching rows from the left relation. If no match exists, the left relation's columns will contain NULL values. Example: Joining Departments with Employees using a right outer join includes all departments, even those with no employees assigned, filling unmatched employee columns with NULL.

* **Full Outer Join:** A full outer join returns all rows when there is a match in either the left or right relation. If a row from one relation does not have a match in the other, NULL values are included for the missing side. **Example:** Joining Customers and Orders using a full outer join will return all customers and orders, even if there's no corresponding order for a customer or no customer for an order.

2. Set Intersection (∩) :s

Set Intersection basically allows to fetches only those rows of data that are common between two sets of relational tables. Set Intersection in relational algebra is the same set intersection operation in set theory.

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

Relation FRENCH

Student_Name	Roll_Number
Ram	01
Mohan	02
Vivek	13
Geeta	17

Relation GERMAN

Student_Name	Roll_Number
Vivek	13
Geeta	17
Shyam	21
Rohan	25

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

$\prod$(Student_Name)(FRENCH $\bigcap$ $\prod$(Student_Name)(GERMAN)

Output:

Student_Name

Vivek

Geeta

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

3. Division (÷) :

The Division Operator is used to find tuples in one relation that are related to all tuples in another relation. It's typically used for "for all" queries.

Student_Course (Dividend Table):

Student_ID	Course_ID
101	C1
101	C2
102	C1
103	C1
103	C2

Course (Divisor Table):

Course_ID
C1
C2

Example: Query is to find students who are enrolled in all courses listed in the Course table. In this case, students must be enrolled in both C1 and C2.

Student_Course(Student_ID, Course_ID)÷ Course(Course_ID)

Output:

Student_ID
101
103

Relational Calculus :

Relational calculus is a non-procedural query language used in the context of relational algebra. It focuses on what data to retrieve, rather than how to retrieve it, making it different from relational algebra, which is procedural. In relational calculus, queries are expressed using logical formulas that describe the desired result, without specifying the exact steps to get there.

There are two types of Relational Calculus

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

1. Tuple Relational Calculus (TRC) :

In Tuple Relational Calculus, we describe the tuples (rows) we want to select from a relation based on certain conditions.

General Form :

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

Where:

* ttt: represents a tuple variable (like one row from a table)
* P(t)P(t)P(t): is a predicate (condition) that must be true for the tuple to be included in the result

Example :

Suppose we have a table Students(Name, Age, Marks).

Name	Age	Marks
Riya	20	92
Arjun	22	85
Neha	19	78

We want to find the names and marks of students who scored more than 80.

TRC Expression :

{t.Name,t.Marks|t∈ Students?t.Marks>80}\{t.Name, t.Marks\ mid t\in Students\ land t.Marks>80 \} {t.Name, t.Marks|t∈ Students^ t.Marks > 80}

Explanation :

* Read as: "Find tuples ttt from the relation Students where Marks are greater than 80."

* It returns:

Name	**Marks**
Riya	92
Arjun	85

Another Example :

Find all students aged 20.

{t|t∈ Students^t.Age=20}\{t\mid t \in Students\land t.Age=20 \} {t|t∈ Students^t.Age=20}

Result:

Name	Age	Marks
Riya	20	92

Key Points (TRC) :

* Queries are expressed using tuple variables.

* Gives the condition that tuples must satisfy.

* It is like saying, "I want all rows (tuples) where this condition holds true."

* Used more often when you care about whole rows of data.

2. Domain Relational Calculus (DRC) :

In Domain Relational Calculus, instead of tuples, we use domain variables, which stand for values of fields (columns).

This means we express the query in terms of columns (attributes) rather than entire rows.

General Form:

{<X1,X2,…,Xn>|P(X1,X2,…,Xn)}\{ \langle X_1, X_2, \ldots, X_n\rangle\midP(X_1,X_2,\ldots,X_n) \} {<X1,X2,…,Xn>|P(X1,X2,…,Xn)}

Where:

* Each XiX_iXi represents a domain variable (value of an attribute).

* PPP is a predicate (logical condition).

Example :

Using the same Students(Name, Age, Marks) table.

We want to find names and marks of students who scored more than 80.

DRC Expression :

{<N,M|∃ A(Students(N,A,M) ^ M>80)}\{ \langle N, M \rangle \mid \exists A (Students(N,A,M)\land M > 80) \} {<N,M|∃ A(Students(N,A,M)^M>80)}

Explanation:

* NNN, AAA, MMM represent the domains of Name, Age, and Marks.

* The symbol ∃ (exists) means "there exists."

* This expression means: Find all (Name, Marks) such that there exists an Age for which the student has Marks greater than 80.

Result:

Name	**Marks**
Riya	92
Arjun	85

Another Example:

Find names of students with age less than 21.

{<N>|∃ A,M(Students(N,A,M)^A<21)}\{ \langle N \rangle \mid \exists A, M (Students(N,A,M)\land A <21) \} {<N>|∃ A,M(Students(N,A,M)^A<21)}

Result:

Name
Riya
Neha

Key Points (DRC):

* Queries are expressed using domain variables rather than tuple variables.

* Focuses on the values (columns) rather than entire rows.

* Used when you need specific fields, not the whole record.

Comparison: TRC vs DRC :

Aspect	Tuple Relational Calculus (TRC)	Domain Relational Calculus (DRC)
Based on	Tuples (rows)	Domains (columns/ attributes)
Variable Type	Tuple variable (e.g., ttt)	Domain variables (e.g., x,y,zx, y, zx,y,z)
Expression Format	{t|P(t)}\{t\mid P(t)\} {t|P(t)}	{<X1,X2,X3>|P(X1,X2, X3)}\{\langle X_1, X_2, X_3 \rangle \mid P(X_1, X_2, X_3) \}{<X1,X2,X3> ||P(X1,X2,X3)}
Example	{t|t ∈ Students^t. Marks>80}\{ t \mid t\in Students \land t.Marks > 80 \}{t|t ∈ Studentst^ Marks>80}	{<N,M|∃ A(Students (N, A, M) ^M>80)}\{ \langle N, M \rangle \mid \exists A (Students(N,A,M) \land M > 80) \}{<N,M>|∃ A (Students (N,A,M)^M>80)}
Focus Level	Whole tuples More concrete	Individual attribute values More abstract