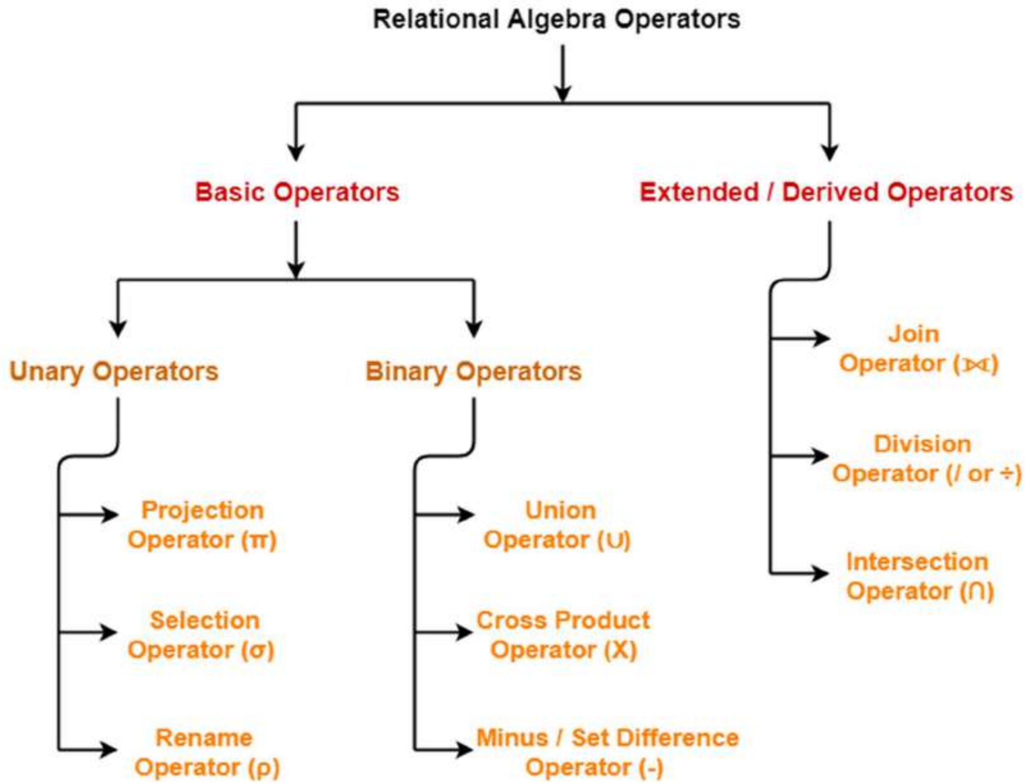
Relational Algebra

Relational algebra refers to a procedural query language that takes relation instances as input and returns relation instances as output. It performs queries with the help of operators. A binary or unary operator can be used. They take in relations as input and produce relations as output. Recursive relational algebra is applied to a relationship, and intermediate outcomes are also considered relations.



Relational Algebra Operations

The following are the fundamental operations present in a relational algebra:

- Select Operation
- Project Operation
- Union Operation
- Set Different Operation
- Cartesian Product Operation
- Rename Operation

1. **Select Operation (or σ)**

It selects tuples from a relation that satisfy the provided predicate.

The notation is − σ*p*(r)

Here **σ** stands for the selection predicate while **r** stands for the relation. *p* refers to the prepositional logic formula that may use connectors such as **or, and,** and **not**. Also, these terms may make use of relational operators such as − =, ≠, ≥, < , >, ≤.

Example

σ*subject = "information"*(Novels)

The output would be − Selecting tuples from the novels wherever the subject happens to be 'information'.

σsubject = "information" and cost = "150"(Novels)

The output would be − Selecting tuples from the novels wherever the subject happens to be 'information' and the 'price' is 150.

σsubject = "information" and cost = "150" or year > "2015"(Novels)

The output would be − Selecting tuples from the novels wherever the subject happens to be 'information' and the 'price' is 150 or those novels have been published after 2015.

2. **Project Operation (or ∏)**

It projects those column(s) that satisfy any given predicate.

Here B1, B2 , An refer to the attribute names of the relation **r**.

The notation is − ∏B1, B2, Bn (r)

Remember that duplicate rows are eliminated automatically, since relation is a set.

Example

∏subject, writer (Novels)

The output would be − Selecting and projecting columns named as writer as well as the subject from the relation Novels.

3. **Union Operation (or ∪)**

It would perform binary union between two relations.

The notation is − r U s

It is defined as follows:

r ∪ s = { t | t ∈ r or t ∈ s}

Here **r** and **s** either refer to DB relations or the relation result set (or temporary relation).

The given conditions must hold if we want any union operation to be valid:

- **s**, and **r** must contain a similar number of attributes.
- The domains of an attribute must be compatible.
- The duplicate tuples are eliminated automatically.

∏ writer (Novels) ∪ ∏ writer (Articles)

The output would be − Projecting the names of those writers who might have written either an article or a novel or both.

4. **Set Different Operation (or −)**

Tuples refers to the result of the set difference operation. These are present in just one of the relations but not at all in the second one.

The notation is − r − s

Finding all the tuples present in **r** and not present in **s**.

∏ writer (Novels) − ∏ writer (Articles)

The output would be − Providing the writer names who might have written novels but have not written articles.

5. **Cartesian Product Operation (or X)**

It helps in combining data and info of two differing relations into a single one.

The notation is − r X s

Where **s** and **r** refer to the relations. Their outputs would be defined as the follows:

s X r = { t ∈ s and q t | q ∈ r}

σwriter = 'mahesh'(Novels X Articles)

The output would be − Yielding a relation that shows all the articles and novels written by mahesh.

6. **Rename Operation (or ρ)**

Relations are the results of the relational algebra, but without any name. Thus, the rename operation would allow us to rename the relation output. The 'rename' operation is basically denoted by the small Greek letter ρ or **rho**.

The notation is − ρ x (E)

Here the result of the E expression is saved with the name of **x**.

The additional operations are as follows:

- Natural join

- Assignment
- Set intersection

Relational algebra operations with examples:

1. **Select Operation (or σ)**

A selection operator (σ) is a unary operator in relational algebra. We use it to select the relation's records or rows that satisfy its condition(s).

Syntax:

$\sigma(condition(s))(relation)$

Parameters

In the syntax above:

- σ denotes the selection operation.
- *condition* denotes any relational conditions. Relational conditions can be anything from these operators[=,!=,>,<,<=,>=][=,!=,>,<,<=,>=].
- *relation* denotes table from the database onto which selection operation is being performed.

Example 1

- $\sigma_{\text{Place = 'Mumbai' or Salary >= 1000000}}$ (Citizen)
- $\sigma_{\text{Department = 'Analytics'}}(\sigma_{\text{Location = 'NewYork'}}(\text{Manager}))$
- The query above(immediate) is called nested expression, here, as usual, we evaluate the inner expression first (which results in relation say Manager1), then we calculate the outer expression on Manager1(the relation we obtained from evaluating the inner expression), which results in relation again, which is an instance of a relation we input.

Example-2:

- Given a relation Student(Roll, Name, Class, Fees, Team) with the following tuples:

Roll	Name	Department	Fees	Team
1	Bikash	CSE	22000	A
2	Josh	CSE	34000	A
3	Kevin	ECE	36000	C
4	Ben	ECE	56000	D

- ‘Select all the student of Team A :
- $\sigma_{\text{Team = 'A'}}$ (Student)

Roll	Name	Department	Fees	Team
1	Bikash	CSE	22000	A
2	Josh	CSE	34000	A

- Select all the students of department ECE whose fees is greater than equal to 10000 and belongs to Team other than A.
- $\sigma_{\text{Fees >= 10000}}(\sigma_{\text{Class != 'A'}}$ (Student))

Roll	Name	Department	Fees	Team
3	Kevin	ECE	36000	C
4	Ben	ECE	56000	D

Example 3:

- Select tuples from a relation “Books” where subject is “database”

$\sigma_{\text{subject = "database"}}$ (Books)

- Select tuples from a relation "Books" where subject is "database" and price is "450"

$\sigma_{\text{subject = "database"} \wedge \text{price = "450"}}$ (Books)

- Select tuples from a relation "Books" where subject is "database" and price is "450" or have a publication year after 2010

$\sigma_{\text{subject = "database"} \wedge \text{price = "450"} \vee \text{year >"2010"}}$ (Books)

Important Points-

Point-01:

- We may use logical operators like ∧ , ∨ , ! and relational operators like = , ≠ , > , < , <= , >= with the selection condition.

Point-02:

- Selection operator only selects the required tuples according to the selection condition.
- It does not display the selected tuples.
- To display the selected tuples, projection operator is used.

Point-03:

- Selection operator always selects the entire tuple. It can not select a section or part of a tuple.

Point-04:

- Selection operator is commutative in nature i.e.

$$\sigma_{A \wedge B}(R) = \sigma_{B \wedge A}(R)$$

OR

$$\sigma_B(\sigma_A(R)) = \sigma_A(\sigma_B(R))$$

Point-05:

- Degree of the relation from a selection operation is same as degree of the input relation.

Point-06:

- The number of rows returned by a selection operation is obviously less than or equal to the number of rows in the original table.

Thus,

- Minimum Cardinality = 0
- Maximum Cardinality = |R|

Projection Operator-

- Projection Operator (π) is a unary operator in relational algebra that performs a projection operation.
- It displays the columns of a relation or table based on the specified attributes.

Syntax-

$\pi_{\text{<attribute list>}}(R)$

Example-

Consider the following Student relation-

ID	Name	Subject	Age
100	Ashish	Maths	19
200	Rahul	Science	20
300	Naina	Physics	20
400	Sameer	Chemistry	21

Student

Then, we have-

Result for Query $\pi_{\text{Name, Age}}$(Student)-

Name	Age
Ashish	19
Rahul	20
Naina	20
Sameer	21

Result for Query $\pi_{ID\,,\,Name}$(Student)-

ID	Name
100	Ashish
200	Rahul
300	Naina
400	Sameer

Important Points-

Point-01:

- The degree of output relation (number of columns present) is equal to the number of attributes mentioned in the attribute list.

Point-02:

- Projection operator automatically removes all the duplicates while projecting the output relation.
- So, cardinality of the original relation and output relation may or may not be same.
- If there are no duplicates in the original relation, then the cardinality will remain same otherwise it will surely reduce.

Point-03:

- If attribute list is a super key on relation R, then we will always get the same number of tuples in the output relation.
- This is because then there will be no duplicates to filter.

Point-04:

- Projection operator does not obey commutative property i.e.

$$\pi_{<list2>} (\pi_{<list1>} (R)) \neq \pi_{<list1>} (\pi_{<list2>} (R))$$

Point-05:

- Following expressions are equivalent because both finally projects columns of list-1

$$\pi_{<list1>} (\pi_{<list2>} (R)) = \pi_{<list1>} (R)$$

Point-06:

- Selection Operator performs horizontal partitioning of the relation.
- Projection operator performs vertical partitioning of the relation.

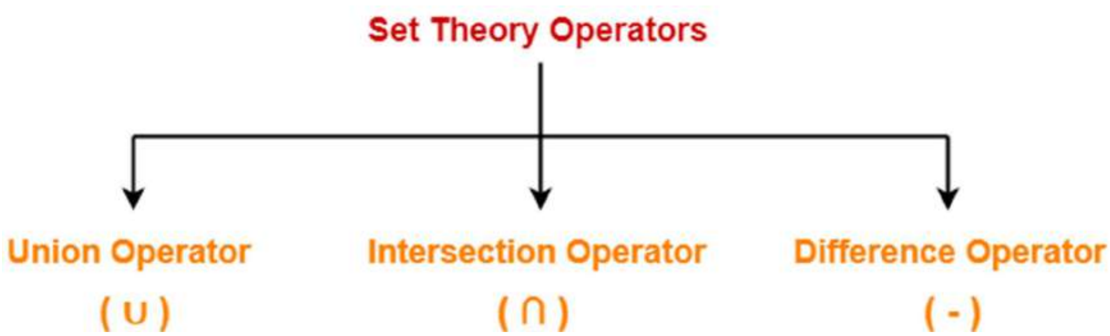
Point-07:

- There is only one difference between projection operator of relational algebra and SELECT operation of SQL.

- Projection operator does not allow duplicates while SELECT operation allows duplicates.
- To avoid duplicates in SQL, we use “distinct” keyword and write SELECT distinct.
- Thus, projection operator of relational algebra is equivalent to SELECT operation of SQL.

Set Theory Operators-

Following operators are called as set theory operators-



1. Union Operator (∪)
2. Intersection Operator (∩)
3. Difference Operator (-)

Condition For Using Set Theory Operators

To use set theory operators on two relations,

The two relations must be union compatible.

Union compatible property means-

- Both the relations must have same number of attributes.
- The attribute domains (types of values accepted by attributes) of both the relations must be compatible.

1. Union Operator (∪)-

Let R and S be two relations.

Then-

- R ∪ S is the set of all tuples belonging to either R or S or both.
- In R ∪ S, duplicates are automatically removed.
- Union operation is both commutative and associative.

Example-

Consider the following two relations R and S-

ID	Name	Subject
100	Ankit	English
200	Pooja	Maths
300	Komal	Science

Relation R

ID	Name	Subject
100	Ankit	English
400	Kajol	French

Relation S

Then, R ∪ S is-

ID	Name	Subject

100	Ankit	English
200	Pooja	Maths
300	Komal	Science
400	Kajol	French

Relation R ∪ S

2. Intersection Operator (∩)-

Let R and S be two relations.

Then-

- R ∩ S is the set of all tuples belonging to both R and S.
- In R ∩ S, duplicates are automatically removed.
- Intersection operation is both commutative and associative.

Example-

Consider the following two relations R and S-

ID	**Name**	**Subject**
100	Ankit	English
200	Pooja	Maths
300	Komal	Science

Relation R

ID	Name	Subject
100	Ankit	English
400	Kajol	French

Relation S

Then, R ∩ S is-

ID	Name	Subject
100	Ankit	English

Relation R ∩ S

3. Difference Operator (-)-

Let R and S be two relations.

Then-

- R – S is the set of all tuples belonging to R and not to S.
- In R – S, duplicates are automatically removed.
- Difference operation is associative but not commutative.

Example-

Consider the following two relations R and S-

ID	Name	Subject

100	Ankit	English
200	Pooja	Maths
300	Komal	Science

Relation R

ID	**Name**	**Subject**
100	Ankit	English
400	Kajol	French

Relation S

Then, R – S is-

ID	**Name**	**Subject**
200	Pooja	Maths
300	Komal	Science

Relation R – S

Cartesian Product/Cross Product:

On applying CARTESIAN PRODUCT on two relations that is on two sets of tuples, it will take every tuple one by one from the left set(relation) and will pair it up with all the tuples in the right set(relation).

So, the CROSS PRODUCT of two relation A(R1, R2, R3, …, Rp) with degree p, and B(S1, S2, S3, …, Sn) with degree n, is a relation C(R1, R2, R3, …, Rp, S1, S2, S3, …, Sn) with degree p + n attributes.

CROSS PRODUCT is a binary set operation means, at a time we can apply the operation on two relations. But the two relations on which we are performing the operations do not have the same type of tuples, which means Union compatibility (or Type compatibility) of the two relations is not necessary.

Notation:
A ⨉ S

where A and S are the relations,
the symbol '⨉' is used to denote the CROSS PRODUCT operator.

Example:
Consider two relations STUDENT(SNO, FNAME, LNAME) and DETAIL(ROLLNO, AGE) below:

SNO	FNAME	LNAME
1	Albert	Singh
2	Nora	Fatehi

ROLLNO	AGE
5	18
9	21

On applying CROSS PRODUCT on STUDENT and DETAIL:

STUDENT ⨉ DETAILS

SNO	FNAME	LNAME	ROLLNO	AGE
1	Albert	Singh	5	18
1	Albert	Singh	9	21
2	Nora	Fatehi	5	18
2	Nora	Fatehi	9	21

We can observe that the number of tuples in STUDENT relation is 2, and the number of tuples in DETAIL is 2. So the number of tuples in the resulting relation on performing CROSS PRODUCT is 2*2 = 4.

Important points on CARTESIAN PRODUCT(CROSS PRODUCT) Operation:

1. The cardinality (number of tuples) of resulting relation from a Cross Product operation is equal to the number of attributes(say m) in the first relation multiplied by the number of attributes in the second relation(say n).
   Cardinality = m*n
2. The Cross Product of two relation A(R1, R2, R3, …, Rp) with degree p, and B(S1, S2, S3, …, Sn) with degree n, is a relation C(R1, R2, R3, …, Rp, S1, S2, S3, …, Sn) with degree p + n attributes.
   Degree = p+n
3. In SQL, CARTESIAN PRODUCT(CROSS PRODUCT) can be applied using CROSS JOIN.
4. In general, we don't use cartesian Product unnecessarily, which means without proper meaning we don't use Cartesian Product. Generally, we use Cartesian Product followed by a Selection operation and comparison on the operators as shown below :
   $\sigma_{A=D} (A \times B)$
   The above query gives meaningful results.

   And this combination of Select and Cross Product operation is so popular that JOIN operation is inspired by this combination.
5. CROSS PRODUCT is a binary set operation means, at a time we can apply the operation on two relations.

Rename Operation:

The RENAME operation is used to rename the output of a relation.

Sometimes it is simple and suitable to break a complicated sequence of operations and rename it as a relation with different names. Reasons to rename a relation can be many, like –

- We may want to save the result of a relational algebra expression as a relation so that we can use it later.
- We may want to join a relation with itself, in that case, it becomes too confusing to specify which one of the tables we are talking about, in that case, we rename one of the tables and perform join operations on them.

Notation:

ρ x (R)

where the symbol 'ρ' is used to denote the RENAME operator and R is the result of the sequence of operation or expression which is saved with the name X.

- **Example-1:** Query to rename the relation Student as Male Student and the attributes of Student – RollNo, SName as (Sno, Name).
- $\rho_{\text{MaleStudent (Sno, Name)}}\ \pi_{\text{RollNo, SName}}\ (\sigma_{\text{Condition}}\ (\text{Student}))$

Sno	**Name**
2600	Ronny
2655	Raja

- Example-2: Query to rename the attributes Name, Age of table Department to A, B.

$\rho_{\text{(A, B)}}\ (\text{Department})$

- Example-3: Query to rename the table name Project to Pro and its attributes to P, Q, R.

$\rho_{\text{Pro(P, Q, R)}}\ (\text{Project})$

- Example-4: Query to rename the first attribute of the table Student with attributes A, B, C to P.

$\rho_{\text{(P, B, C)}}\ (\text{Student})$

Join Operations:

A Join operation combines related tuples from different relations, if and only if a given join condition is satisfied. It is denoted by ⋈.

Example:

EMPLOYEE

EMP_CODE	EMP_NAME
101	Stephan
102	Jack
103	Harry

SALARY

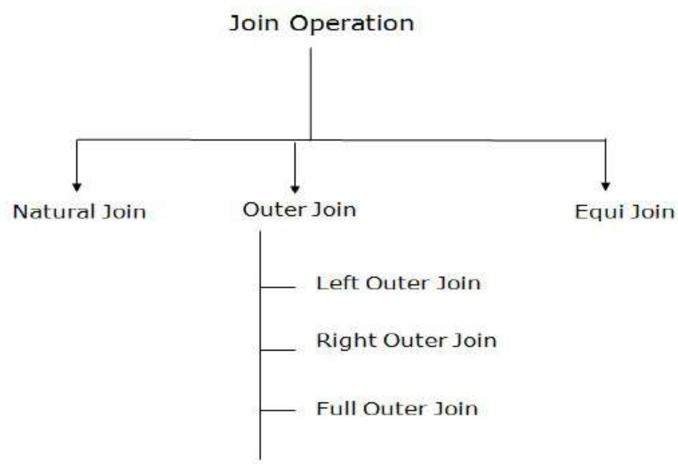
EMP_CODE	SALARY
101	50000
102	30000
103	25000

1. Operation: (EMPLOYEE ⋈ SALARY)

Result:

EMP_CODE	EMP_NAME	SALARY
101	Stephan	50000
102	Jack	30000
103	Harry	25000

Types of Join operations:



1. Natural Join:

- A natural join is the set of tuples of all combinations in R and S that are equal on their common attribute names.
- It is denoted by ⋈.

Example: Let's use the above EMPLOYEE table and SALARY table:

Input:

1. ∏EMP_NAME, SALARY (EMPLOYEE ⋈ SALARY)

Output:

EMP_NAME	SALARY
Stephan	50000
Jack	30000
Harry	25000

2. Outer Join:

The outer join operation is an extension of the join operation. It is used to deal with missing information.

Example:

EMPLOYEE

EMP_NAME	STREET	CITY
Ram	Civil line	Mumbai
Shyam	Park street	Kolkata
Ravi	M.G. Street	Delhi
Hari	Nehru nagar	Hyderabad

FACT_WORKERS

EMP_NAME	BRANCH	SALARY
Ram	Infosys	10000
Shyam	Wipro	20000
Kuber	HCL	30000
Hari	TCS	50000

Input:

1. (EMPLOYEE ⋈ FACT_WORKERS)

Output:

EMP_NAME	STREET	CITY	BRANCH	SALARY
Ram	Civil line	Mumbai	Infosys	10000
Shyam	Park street	Kolkata	Wipro	20000
Hari	Nehru nagar	Hyderabad	TCS	50000

An outer join is basically of three types:

a. Left outer join
b. Right outer join
c. Full outer join

a. Left outer join:

- Left outer join contains the set of tuples of all combinations in R and S that are equal on their common attribute names.
- In the left outer join, tuples in R have no matching tuples in S.
- It is denoted by ⟕.

Example: Using the above EMPLOYEE table and FACT_WORKERS table

Input:

1. EMPLOYEE ⟕ FACT_WORKERS

EMP_NAME	STREET	CITY	BRANCH	SALARY
Ram	Civil line	Mumbai	Infosys	10000
Shyam	Park street	Kolkata	Wipro	20000
Hari	Nehru street	Hyderabad	TCS	50000
Ravi	M.G. Street	Delhi	NULL	NULL

b. Right outer join:

- Right outer join contains the set of tuples of all combinations in R and S that are equal on their common attribute names.
- In right outer join, tuples in S have no matching tuples in R.
- It is denoted by ⟖.

Example: Using the above EMPLOYEE table and FACT_WORKERS Relation

Input:

1. EMPLOYEE ⟖ FACT_WORKERS

Output:

EMP_NAME	BRANCH	SALARY	STREET	CITY
Ram	Infosys	10000	Civil line	Mumbai
Shyam	Wipro	20000	Park street	Kolkata
Hari	TCS	50000	Nehru street	Hyderabad
Kuber	HCL	30000	NULL	NULL

c. Full outer join:

- Full outer join is like a left or right join except that it contains all rows from both tables.
- In full outer join, tuples in R that have no matching tuples in S and tuples in S that have no matching tuples in R in their common attribute name.
- It is denoted by ⟗.

Example: Using the above EMPLOYEE table and FACT_WORKERS table

Input:

1. EMPLOYEE ⟗ FACT_WORKERS

Output:

EMP_NAME	STREET	CITY	BRANCH	SALARY
Ram	Civil line	Mumbai	Infosys	10000
Shyam	Park street	Kolkata	Wipro	20000
Hari	Nehru street	Hyderabad	TCS	50000
Ravi	M.G. Street	Delhi	NULL	NULL
Kuber	NULL	NULL	HCL	30000

3. Equi join:

It is also known as an inner join. It is the most common join. It is based on matched data as per the equality condition. The equi join uses the comparison operator(=).

Example:

CUSTOMER RELATION

CLASS_ID	NAME
1	John
2	Harry
3	Jackson

PRODUCT

PRODUCT_ID	CITY

1	Delhi
2	Mumbai
3	Noida

Input:

1. CUSTOMER ⋈ PRODUCT

Output:

CLASS_ID	NAME	PRODUCT_ID	CITY
1	John	1	Delhi
2	Harry	2	Mumbai
3	Harry	3	Noida