Relational Algebra:-

It consists of set of operations that take one or more tables or relation as a input and produce the new relation (or) new tables.

- It consists of 2 types of Operations

① Simple operations ② Extended operations

Query examples on Simple operations

① Selection (σ): It is used to select the data from existing database

Syntax: σ Column_name = Value (Table_name)

Student

S_id	S_name	S_marks	S_age
12	Jack	95	25
16	Elena	93	22
3	Ravi	98	14
4	Swathi	89	18

① Find the student details whose age is 22 from the student relation

Query: σ S_age = 22 (Student)

o/p:

Sid	S_name	S_marks	S_age
16	Elena	93	22

② Select tuples from a relation "Student" where name is "Ravi"

Query: σ S_name = "Ravi" (Student)

Sid	S_name	Smarks	Sage
3	Ravi	98	14

③ Select tuples from the relation whose age is less than 20

Query: $\sigma_{Sage<20}(student)$

o/p:

Sid	S_name	S_marks	S_age
3	Ravi	98	14
4	Swathi	89	18

④ Find the student details who are having marks >=95 and age > 20 in student relation

Query: $\sigma_{S\_marks >= 95 \wedge Sage > 20}(student)$

o/p:-

Sid	S_name	S_marks	Sage
12	Jack	95	25

⑤ find the student details who are having marks >95 or age or< 20 in student relation

Query: $\sigma_{S\_marks > 95 \vee Sage < 20}(student)$

o/p:-

Sid	S_name	S_marks	Sage
3	Ravi	98	14
4	Swathi	89	18

② Projection (Π):-

It is used to select attributes or fields

Syntax:- Π column_name1, column_name2, --- (Table_name)

① find all the student names in the student relation

Query: $\Pi_{S\_name}(student)$

② o/p:-

S_name
Jack
Elena
Ravi
swathi

② Find the names of all the students who have attempted the exam and registered with student id.

Query: Π S_name (student)

o/p:-

S_name
Jack
Elena
Ravi
Swathi

③ Find the student names who are having Student Id < 10

Query: Π S_name (σ s_id < 10 (student))

o/p:-

S_name
Ravi
Swathi

④ Find the Student names and their marks from student relation

Query: Π S_name, S_marks (Student)

o/p:-

S_name	S_marks
Jack	95
Elena	93
Ravi	98
Swathi	89

⑤ project the names of students who are having marks < 90 and age < 20 from student relation.

Query: Π S_name (σ s_marks < 90 $\wedge$ S_age < 20 (student))

o/p:-

S_name
Swathi

* Rename:- It is used to rename the existing table (or) relation

SYNTAX:- Rename (new_name, old name)

Query to change Student to STU

Query:- P(STU, student) (table name is rename as STU)

new old

UNION:- It is used to combine values by removing duplicating ones (∪)

Table A (R_1)

Reg_no	Branch	Section
1	CSE	A
2	ECE	B
3	CIVIL	B
4	MECH	A

Table B (R_2)

Reg_no	Branch	Section
1	CSE	A
2	CIVIL	A
3	ECE	B

① Query to display all the regno of R1 & R2 from relational tables above

Query: Π Reg_no (Table A) ∪ Π (Reg_no(Table B)

o/p

Reg_no
1
2
3
4

② Query to display all the Branches available from relational tables above

Query: Π Branch (Table A) ∪ Π Branch (Table B)

Branch
CSE
ECE
CIVIL
MECH

③ find the Reg_no of all Branches where the branch belongs to Section or sectionB or both.

Query: Π Reg_no(TableA) ∪ Π(Reg_no (Table B)

o/p:-

Regno
1
2
3
4

Intersection:- It is used to display the common values or common content in both tables. (∩)

① find all the Branch names which are common in both the tables.

o/p:-

Branch
CSE
CIVIL
ECE

Query: Π Branch(Table A) ∩ Π Branch(Table B)

② Query to display the sections that are common in both tables

Query: Π Section(Table A) ∩ Π Section (Table B)

o/p:-

Section
A
B

Difference:- It is used to display the information from 1st table which is not common in both tables. (−)

① Query to display all the branch names which are in table 1 but not table 2

o/p:-

Branch
MECH

Query: Π Branch (Table A) − Π Branch (Table B)

Cross product:- It is used to generate a paired combination of each row of the first table with each row of the second table (×)

① Query on performing Cartesian product

deposit

C-ID	Deposits
1	12000
2	14000

loan

S-ID	loan
3	7000
4	3500

Find the cross product of the customers who have deposited amount >10000 × loan < 10000

Query: $(\sigma_{Amount > 10000}(Deposit) \wedge \sigma_{Amount < 10000}(loan))$

$\therefore (Deposit \times loan)$

C_ID	Deposits	S_ID	loan
1	12000	3	7000
1	12000	4	3500
2	14000	3	7000
2	14000	4	3500

Table 1

cust_name	Account_num
AAA	96294563
BBB	11200692 9
CCC	63488000

Table 2

cust_name	loan_amount
FFF	100000
AAA	220000
DDD	10000

UNION:

① find customer name of all customers who have either an account number or a loan amount or both.

Query:- Π cust_name (Table 1) $\cup$ Π cust_name (Table 2)

o/p:-

cust_name
AAA
BBB
CCC
FFF
DDD

② Intersection:

① find all the customer names who have both account number and taken loan from relation tables above

Query: Πcust_name (Table 1) $\cap$ Π cust_name (Table 2)

o/p:-

cust_name
AAA

Difference:-

① find the name of all the customers who have account number and not taken loan

Query: Π cust_name (Account_num) - Π cust_name (loan_amount)

o/p:-

cust_name
BBB
CCC

Division

It is denoted as '/'

rules:-

1) All the attributes of B are proper subset of 'A' (A, B are two relations)
2) All the attributes of A - All the attributes of B
3) It will return tuple relations from relation A which are associated to every 'B' tuple

Ex-①

Enroll table (A)

Sid	Cid
S_1	C_1
S_2	C_1
S_1	C_2
S_3	C_2

course (B)

Cid
C_1
C_2

Ans 1) B is subset to A ($B \subseteq A$)

2) all attributes of B - all attributes of A
   = (Sid, Cid) - Cid = Sid

3) resultant relation is

Sid
S_1

Ex-②

(A) Table

rollno	Sports
1	Badminton
2	Cricket
2	Badminton
4	Badminton

(B) Table

Sports
Badminton
cricket

result =

rollno
2

Extended Operators

The join operation:

The join operation denoted by "join" ($\bowtie$) is a relational algebra operation, which is used to combine (join) two relations like cartesian-product but finally removes duplicate attributes (same column-to only one-column) and makes the operations (selection, projection, etc) very simple. In simple words, we can say that join connects relations on columns containing comparable information.

There are three types of join

(Inner join)

① Natural join (or) Equijoin: The natural join is a binary operation that allows us to combine two different relations into one relation and makes the same column in two different relations into one-column in the resulting relation.

employee (emp_name, street, city)
employee_works (emp_name, branch_name, salary)

The relation are:

employee

emp_name	Street	City
Coyote	Town	Hollywood
Rabbit	Tunnel	Carrot ville
Smith	Revolver	Valley
williams	Seaview	Seattle

Employee_works

emp_name	Branch_name	Salary
Coyote	Mesa	1500
Rabbit	Mesa	1300
Gates	Redmond	5300
Williams	Redmond	1500

Query: Generate a single relation with all the information about full-time employees.

employee ⋈ employee_works

result:-

emp_name	street	city	branch_name	Salary
Coyote	Town	Hollywood	Mesa	1500
Rabbit	Tunnel	Carrotville	Mesa	1300
Williams	Seaview	Seattle	Redmond	1500

Notice, that we have lost street and city information about Smith, since tuples describing Smith is absent in emp_works. Similarly, we have lost branch_name and salary information about Gates, since the tuple describing Gates is absent from employee relation.

Now, we can easily select or project query on new join relation.

Query: find the employee names and city who have Salary details

$\Pi_{employee\_name, salary, city}$ (employee ⋈ emp_works)

emp_name	city	Salary
coyote	Hollywood	1500
Rabbit	Carrotville	1300
williams	Seate	1500

③ Outer join:- The drawback of natural join is some loss of information, to overcome this, we use Outer-join Operation

These are three types:

ⓐ Left-outer join (⟕)
ⓑ Right-outer join (⟖)
ⓒ full-outer join (⟗)

ⓐ Left-outer join: The left outer-join takes all tuples in left relation that did not match with any tuples in right relation, adds the tuples with null values for all other columns from right relation and adds them to the result of natural join

employee ⟕ employee_works

emp_name	Street	city	branch_name	Salary
Coyote	Town	Hollywood	Mesa	1500
Rabbit	Tunnel	Carrotville	Mesa	1300
Williams	Seaview	Seattle	Redmond	1500
Smith	Revolver	Valley	null	null

ⓑ Right_outer join: The right outer-join operation takes all the tuples in the right (left) relation that did not match with any tuple in left relation, adds the tuples with null values for all other columns from left relation and adds to the result of natural-join.

employee ⟖ employee_works

emp_name	Street	city	branch_name	Salary
coyote	Town	Hollywood	Mesa	1500
Rabbit	Tunnel	Carrotville	Mesa	1300
Williams	Seaview	Seattle	Redmond	1500
Gate's	null	null	Redmond	5300

© Fullouter-join: the full outer-join operation does ~~not~~ both of those operations, by adding tuples from left relation that did not match any tuples from the right relation, as well as adds tuples from the right relation that did not match any tuple from the left relation and adding them to the result of natural-join

employee ⟗ employee_works

emp_name	street	city	branch_name	salary
coyote	Town	Hollywood	Mesa	1500
Rabbit	Tunnel	Carrotville	Mesa	1300
Williams	Seaview	Seattle	Redmond	1500
Smith	Revolver	Valley	null	null
Gates	null	null	Redmond	5300

③ Theta join (or) condition join: The theta join operations, denoted by symbol, ⋈θ is an extension to the natural-join operation that combines two relations into one relation with a selection condition (θ).

Query: find the details of employee whose salary is <1400

employee ⋈ salary<1400 emp_works

emp_name	street	city	branch_name	salary
Rabbit	Tunnel	Carrotville	Mesa	1300

④ cross-join (X)

for clubbing two relations that do not have any common column, we use cross join (or) (Cartesian product of join

Student

S_id	Sname
1	A
2	B
3	C

Course

C_id	Cname	Camount
10	CSE	50000
20	ECE	44000
30	EEE	40000

S_id	Sname	C_id	Cname	Camount
1	A	10	CSE	50000
1	A	20	ECE	44000
1	A	30	EEE	40000
2	B	10	CSE	50000
2	B	20	ECE	44000
2	B	30	EEE	40000
3	C	10	CSE	50000
3	C	20	ECE	44000
3	C	30	EEE	40000