UNIT-5

The Relational Algebra and Relational Calculus:

Query Language → A query language is a language in which a user requests information from the database. It is of two types: procedural (in which user instructs system to perform sequence of operations on database. Example: Relational algebra) and non-procedural (in which user describes the desired information without giving specific procedure. Example: tuple relational calculus, SQL etc.).

⊛ Relational Algebra:

Concept only no need to remember all

It is a procedural query language which takes instances of relations as input and yeilds instances of relations as output. It uses operators to perform queries. An operator can be unary or binary.

Main operations of relational algebra

- **Fundamental Operations**
  - Select
  - Project
  - Union
  - Difference
  - Rename
  - Cartesian product
- **Additional Operations**
  - Set-insertion
  - Natural-join
  - Division
  - Assignment
- **Extended Relational Algebra Operations**
  - Generalized projection
  - Aggregate function
  - Outer join
  - Null values
- **Unary Relational Operations**
  - Select (symbol: σ)
  - Project (symbol: π)
  - Rename (symbol: ρ)

⊛ Unary Relational Operations: SELECT and PROJECT

The relational algebra operations that uses single relation (table) are called unary relational operations.

ⓐ Select (σ): The Select operation is used for selecting a subset of the tuples according to a given selection condition. It is denoted by Sigma (σ) symbol.

Syntax: σ <selection condition> (R)

where, R stands for relation which is the name of the table.

Comparision operators ($<, >, \leq, \geq, =, \neq$) can be used to specify conditions required for selection tuples from a relation. Furthermore logical operations AND ($\wedge$), OR ($\vee$) and NOT ($\neg$) are used to combine two or more conditions.

Example: Let's take a student relation.

Stu_id	Stu_name	Stu_address	Dept_id	Age
10	Maya	Palpa	1	22
11	Abin	Ktm	2	17
12	Aarav	Ktm	1	21
13	Ashna	Palpa	3	45
14	Anuj	Pokhara	4	23

i) Find records of all students of address 'Palpa'.

$\sigma_{Stu\_address = \text{"Palpa"}} (Student)$

ii) Find all students of age greater than 20 or of address 'Ktm'.

$\sigma_{Stu\_address = \text{"Ktm"} \vee Age > 20} (Student)$

b. Projection(π): The project operation is used to select certain columns from the table and discards the other columns. It is denoted by Pie (π) symbol.

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

Example: We are using above student table,

i) Display name and id of the student.

$\pi_{Stu\_id, Stu\_name}(Student)$

ii) Display name and age of students

$\pi_{Stu\_name, Age}(Student)$

* Combining Selection and Projection Operations:

The selection and projection operators are combined to perform projection with selection operation.

Syntax: $\pi_{<attribute\text{-}list>}(\sigma_{<selection\ condition>}(R))$

Example: We are using above student table,

i) Find name, address, age of student whose age less than or equal to 25.

$\pi_{Stu\_name, Stu\_address}(\sigma_{age \leq 25}(Student))$

ii) Find name of students whose age is greater than 20 and of address "Palpa".

$\pi_{Stu\_name}(\sigma_{age > 25 \wedge Stu\_address \text{"Palpa"}}(Student))$.

⊛. Sequence of Operations and the RENAME Operation:

We can either write the operations as a single relational algebra expression by nesting the operations, or we can apply one operation at a time and create intermediate result relations. We must give names to the relations that hold the intermediate results. For example: To retrive, first name, last name and salery of all employees who work in department number 5, we must apply SELECT and PROJECT operation as follows:-

$$\pi_{Fname, Lname, Salery}(\sigma_{Dno=5}(EMPLOYEE)).$$

This is in-line relational algebra expression, also known as in-line expression.

Alternatively, we can show the sequence of operations, giving a name to each intermediate relation, and using the assignment operation, denoted by ← (left arrow) as follows:-

$$DEP5\_EMPS \leftarrow \sigma_{Dno=5}(EMPLOYEE)$$

$$RESULT \leftarrow \pi_{Fname, Lname, Salery}(DEP5\_EMPS).$$

It is sometimes simpler to break down a complex sequence of operations by specifying intermediate result relations than to write a single relational algebra expression. This can be useful with more complex operations such as UNION and JOIN.

RENAME Operation:- We can also define a formal RENAME operation which can rename either the relation name or attribute names, or both as a unary operator. The general RENAME operation when applied to a relation R of degree n is denoted by any of the following three forms:

$$\rho_{S(B1,B2,\ldots,Bn)}(R) \quad \text{or} \quad \rho_S(R) \quad \text{or} \quad \rho_{(B1,B2,\ldots,Bn)}(R).$$

where the symbol ρ (rho) is used to denote the RENAME operator, S is the new relation name, and B1, B2, ... Bn are new attribute names.

*. Relational Algebra Operations from Set Theory:-

i) Union Operation (U):- Let two union-compatible relations be R and S. Then, the union operation (U) ~~is den~~ between R and S is denoted by RUS, is a relation that includes all tuples that are either in R or in S or in both R and S. Duplicate tuples are eliminated.

Example: Let's take following two tables namely morning shift Employee as "Memployee" and Day shift employee as "Demployee".

Memployee

eid	name	salery
e1	Rajan	34,000
e2	Aarab	45,000
e3	Abin	55,000
e4	Ashna	24,000

Demployee

eid	name	salery
e1	Rajan	34,000
e5	Umesh	78,000
e8	Anisha	55,000
e4	Ashna	33,000

Now Memployee U Demployee is as follows:-

eid	name	salery
e1	Rajan	34,000
e2	Aarab	45,000
e3	Abin	55,000
e4	Ashna	24,000
e5	Umesh	78,000
e8	Anisha	55,000
e4	Ashna	33,000

Since e1 Rajan 34,000 of Demployee is repeating (i.e, duplicate data) so eliminated

ii) Intersection Operation (∩):- It is denoted by symbol ∩ and it returns a relation that contains tuples that are in both of its argument relations.

For example:- from above tables Memployee and Demployee Memployee ∩ Demployee is as follows:-

eid	name	Salery
e1	Rajan	34,000

iii) Difference (−):- It is denoted by minus sign (−). It finds tuples that are in one relation, but not in another. Thus results in relation containing tuples that are in R but not in S.

For example: Memployee − Demployee is:-

eid	name	salery
e2	Aarav	45000
e3	Abin	55000
e4	Ashna	24000

Scanned by TapScanner

iv) Cartesian Product (x) :- This type of operation is helpful to merge columns from two relations. The cartesian product operation does not require relations to union-compatible i.e, the involved relations may have different schemas. The cartesian product of two relations R and S is denoted by R x S, is the set of all possible combinations of ~~tubl~~ tuples of two relations.

For Example :

Department

Dept_id	Dept_name	Dept_block no
1	Computer	100
2	Mathematics	200
3	Economics	300
4	Account	400

Staff

Staff_id	staff_name	Dept_id
11	Mohan	1
22	Pratima	2
33	Madan	1

Now Department x Staff is as follows:-

D.Dept_id	Dept_name	Dept_block no	Staff_id	Staff_name	S.Dept_id
1	Computer	100	11	Mohan	1
1	Computer	100	22	Pratima	2
1	Computer	100	33	Madan	1
2	Mathematics	200	11	Mohan	1
2	Mathematics	200	22	Pratima	2
2	Mathematics	200	33	Madan	1
3	Economics	300	11	Mohan	1
3	Economics	300	22	Pratima	2
3	Economics	300	33	Madan	1
4	Account	400	11	Mohan	2
4	Account	400	22	Pratima	1
4	Account	400	33	Madan	2

⊛. Binary Relation Operations : JOIN and DIVISION:-

The operations that are used to perform operations into multiple tables are called binary relation operations.

(a) Join operation: Join operation is essentially a cartesian product followed by a selection criteria. It is denoted by ⋈.

Types of Join operations

- Inner Join
  - Theta join
  - Equi join
  - Natural join
- Outer Join
  - Left outer join
  - Right outer join
  - Full outer join

1) **Inner Join:** In an inner join, only those tuples that satisfy the matching criteria are included, while the rest are excluded.

i) **Theta join** → The general case of join operation is called a theta join. It is denoted by symbol θ. The theta condition consists one of the comparision operators (=, <, <=, >, >=, <>).

Syntax: $A \bowtie_{\theta} B$ where, A and B are any two relations and θ is a join condition.

Example:

Department

Dept_id	Dept_name	Dept_block_no
1	Computer	100
2	Mathematics	200
3	Economics	300

Staff

Staff_id	Staff_name	Dept_id
11	Mohan	1
22	Pratima	2
33	Madan	1

Now Department $\bowtie_{D.Dept\_id > S.Dept\_id}$ (Staff).

D.Dept_id	Dept_name	Dept_block_no	Staff_id	Staff_name	S.Dept_id
2	~~Computer~~ Mathematics	200	11	Mohan	1
2	~~Computer~~ Mathematics	200	33	Madan	1
3	Economics	300	11	Mohan	1
3	Economics	300	22	Pratima	2
3	Economics	300	33	Madan	1

ii) **Equi Join** → When join condition is '=' i.e, θ is =, the operation is called equijoin.

Syntax: $A \bowtie_{=} B$ where, A and B are any two relations and = is a join operation.

Scanned by TapScanner

Example:- Let we take above two relations Department and Staff.

Department ⋈ D.Dept_id = S.Dept_id (Staff).

D.Dept_id	Dept_name	Dept_block_no	Staff_id	Staff_name	S.Dept_id
1	Computer	100	11	Mohan	1
1	Computer	100	33	Madan	1
2	Mathematics	200	22	Pratima	2

iii) Natural Join → Natural join can only be performed if there is a common attribute (column) between the relations. The name and type of the attribute must be same. It allows us to combine certain selections and a cartesian product into one operation. It is denoted by join symbol, ⋈. The natural ~~joins~~ join performs a join by equating the attributes with the same name and then eliminates duplicate attributes.

Example:- Let we take above two relations Department and Staff.

Department ⋈ Staff

D.Dept_id	Dept_name	Dept_block_no	Staff_id	Staff_name.
1	Computer	100	11	Mohan
1	Computer	100	33	Madan
2	Mathematics	200	22	Pratima

Same id भाको लेखेर आर्को id eliminate गरेको

2> Outer Join: In an outer join, along with tuples that satisfy the matching criteria, we also include some or all tuples that do not match the criteria. Thus the outer join is an extension of the join operation to deal with missing information.

i) Left Outer Join (⟕) → It allows keeping all tuple in left relation. However, if there is no matching tuple is found in right relation, then the attributes of right relation in the join result are filled with NULL values.

Example: Department ⟕ Staff.

D.Dept_id	Dept_name	Dept_block_no	Staff_id	Staff_name	S.Dept_id
1	Computer	100	11	Mohan	1
1	Computer	100	33	Madan	1
2	Mathematics	200	22	Pratima	2
3	Economics	300	NULL	NULL	NULL

ii) Right Outer Join (⟖) → This operation allows keeping all tuple in the right relation. However, if there is no matching tuple found in the left relation, then the attributes of the left relation in the join result are filled with NULL values.

Example:- Let we have following two relations:-

Department

Dept_id	Dept_name	Dept_block_no
1	Computer	100
3	Economics	300

Staff.

Staff_id	Staff_name	Dept_id
11	Mohan	1
22	Pratima	2
33	Madan	3

Now Department ⟖ Staff is as follows:-

D.Dept_id	Dept_name	Dept_block_no	Staff_id	Staff_name	S.Dept_id
1	Computer	100	11	Mohan	1
~~1~~	~~Computer~~	~~100~~	~~33~~	~~Madan~~	~~4~~
NULL	NULL	NULL	22	Pratima	2
3	Economics	300	33	Madan	3

iii) Full Outer Join (⟗) → It includes all tuples in left hand relation and from the right hand relation. In a full outer join, all ~~two~~ tuples from both relations are included in the result, irrespective of matching condition.

Example:- Let we have following two relations:

Department

Dept_id	Dept_name	Dept_block_no
1	Computer	100
3	Economics	300

Staff.

Staff_id	Staff_name	Dept_id
11	Mohan	1
22	Pratima	2

Now Department ⟗ Staff is as follows:-

D.Dept_id	Dept_name	Dept_block_no	Staff_id	Staff_name	S.Dept_id
1	Computer	100	11	Mohan	1
3	Economics	300	NULL	NULL	NULL
NULL	NULL	NULL	22	Pratima	2

⊛. **Division operation (÷)**:- It is denoted by symbol ÷ and is suited to queries that include the phrase "for all". It takes two relations and builds another relation, consisting of values of an attribute of one relation that match all the values in the other relation.

Examples:-

A

sno	pno
s1	p1
s1	p2
s1	p3
s1	p4
s2	p1
s2	p2
s3	p2
s4	p2
s4	p4

B1

pno
p2

B2

pno
p2
p4

B3

pno
p1
p2
p4

A/B1

sno
s1
s2
s3
s4

A/B2

sno
s1
s4

A/B3

sno
s1

⊛ Additional Relational Operations:

⊛. **Concept of Generalized Projection**:- Generalized projection is enhanced version of project operation. It allows us to write aromatic operations containing attribute names and constants in projection list. General form of generalized projection is as follows:

$$\Pi_{F1, F2, F3, \ldots, Fn}(E).$$

where, E is a relational algebra expression and F_i $(i=1, 2, \ldots n)$ is an attribute or arithmetic expression containing attributes and constants.

Example:- Let's take an employee relation as follows:-

Employee

eid	name	Age	Salery	Address
e1	Rajan	33	34000	Ktm
e2	Arav	17	45000	Pokhara
e3	Abin	22	55000	Palpa
e4	Ashna	19	24000	Ktm

i) Find name and ~~salary~~ salery of all employees by increasing their salery by 15%.

$$\Pi_{name, salery = salery + salery * 0.15}(Employee)$$

ii) Increase the salery of all employees whose age greater than 20 by 5%.

$$\Pi_{eid, name, age, salery = salery + salery * 0.05}(\sigma_{age>20}(Employee))$$

Aggregate Functions:

Aggregate functions are algebraic functions that take a collection of values as input and return a single value as a result. It is denoted by symbol $\mathcal{G}$ read as "calligraphic G". General form of aggregate operation in relational algebra is;

$$G1, G2, \ldots Gn\ \mathcal{G}_{F1(A1), F2(A2), \ldots, Fn(An)}(E).$$

where, E is any relational-algebra expression.
G1, G2, ... Gn is a list of attributes on which to group and it can be empty.
Each F_i is an aggregate function and each A_i is an attribute name.

There are five aggregate functions:

→ SUM: sum of values
→ AVG: average value
→ MIN: minimum value
→ MAX: maximum value
→ COUNT: number of values.

Example:- Consider Employee relation that we have in example of generalized projection before;

i) Find total number of employees.

$$\mathcal{G}_{COUNT(eid)}(Employee)$$

ii) Find average age of employees of address 'ktm'

$$\mathcal{G}_{AVG(Age)}(\sigma_{address = "ktm"}(Employee))$$

iii) Find minimum and maximum age of the employee.

$$\mathcal{G}_{MIN(Age), MAX(Age)}(Employee).$$

iv) Find average salery of employee in each address level.

$$Address\ \mathcal{G}_{AVG(Salery)}(Employee).$$

v) Find total salery of employees.

$$\mathcal{G}_{SUM(Salery)}(Employees).$$

⊛ Tuple Relational Calculus:

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. In Tuple calculus, a query is expressed as;

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

where, t = resulting tuples.
$P(t)$ = known as predicate and these are the conditions that are used to fetch t.

Thus, it generates set of all tuples t, such that Predicate $P(t)$ is true for it. $P(t)$ may have various conditions logically combined with OR ($\vee$), AND ($\wedge$), NOT ($-$). It also uses quantifiers $\exists$ (there exists) and $\forall$ (for all).

Example:- Consider a loan relation as follows:-

Loan number	Branch name	Amount
L33	ABC	10,000
L35	DEF	15,000
L49	GHI	9000
L98	DEF	65000

i) Find the loan number, branch, amount of loans of greater than or equal to 10000 amount.

$$\{t \mid t \in loan \wedge t[amount] >= 10000\}$$

⊛. Domain Relational Calculus:

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;

$$\{< x_1, x_2, x_3, \ldots, x_n > \mid P(x_1, x_2, x_3, \ldots, x_n)\}$$

where, $<x_1, x_2, x_3, \ldots, x_n>$ represents resulting domain variables and $P(x_1, x_2, x_3, \ldots, x_n)$ represents the condition or formula equivalent to Predicate calculus.

Predicate Calculus formula:

→ Set of all comparision operators
→ Set of connectives like and, or, not.
→ Set of quantifiers.

Example: Consider the following relation Loan ~~and Borrower~~.

Loan

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

Borrower

Customer name	Loan number
Ritu	L01
Debamit	L08
Soumya	L03

i) Find the loan number, branch, amount of loans of greater than or equal to 100 amount.

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

Result:

Loan number	Branch name	Amount
L01	Main	200
L03	Main	150

Differences between Relational Algebra & Relational Calculus:

Relational algebra	Relational Calculus
i) Relational algebra is a procedural language.	i) Relational calculus is a declarative language.
ii) It states how to obtain the result.	ii) It states what result we have to obtain.
iii) It describes the order in which operations have to be performed.	iii) It does not specify the order of operations.
iv) It is not domain dependent.	iv) It can be domain dependent.
v) It is close to programming language.	v) It is close to natural language.

Note:- In addition have a look at relational algebra examples page no. 85 of kec book.