Unit-3 The Relational Algebra & Relational Calculus

-Ankit Pangeni

DATE

* Query Languages:-

A query language is a language in which a user requests information from the database. It can be categorized as either procedural or non-procedural

-> In procedural language, the user interacts the system to perform a sequence of operations on the database to compute the desired result. Ex: Relational Algebra

-> In nonprocedural language, the user describes the desired information without giving a specific procedure for obtaining that information. Ex: tuple relational calculus, domain relational calculus, SQL etc

* Unary Relational Operations: SELECT & PROJECT & RENAME

* Relational Algebra:

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

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

DATE

* Select (σ)

→ The select operation is used for selecting a subset of the ~~types~~ tuples according to a given selection condition.
It is denoted by sigma(σ) symbol.

→ It is used as an expression to choose tuples which meet the selection condition.

→ selection operation selects tuples that satisfy a given predicate.

→ Syntax: σ <selection condition> (R)
(where, R = relation which is the name of table

→ Comparison operators (<, >, ≤, ≥, =, ≠) can be used to specify conditions required for selection tuples from a relation.

→ Logical operators AND (∧), OR (∨) and NOT(¬) can be used to combine two or more conditions

Ex: Lets take a student relation.

Student

Stu_id	Stuname	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
15	Manish	Banepa	2	23
16	Pinky	Ktm	1	16

DATE

Q. Find records of all students of address 'Palpa'.

$\sigma_{stu\_address = "Palpa"}(student)$

Q. Find all students whose address "Ktm" & of department id 1.

$\sigma_{stu\_address = "Ktm" \wedge Dept\_id = 1}(student)$

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

$\sigma_{stu\_address = "Ktm" \vee Age \geq 1}(student)$

* Projection (Π)

→ The Project operation is used to select certain columns from the table & discard other columns.

→ If we are interested in only certain attributes of a relation, we use the Project operation to project the relation over these attributes only

Syntax: $\Pi_{<attribute\_list>}(R)$

Eg: In above table.

Q. Display Id & name of the student

$\Pi_{stu\_id, stu\_name}(student)$

Q. Display name & age of student.

$\Pi_{stu\_name, Age}(student)$

classmate

PAGE

DATE

* Combining Selection & Projection Operations.

→ The selection and projection operators can be combined to perform projection with selection operation Here we use selection operation with projection operation.

→ At first inner operation (selection operation) extract given specified rows and then outer operation ie. projection operation extract only specified columns from selected rows.

Syntax: Π <attribute_list> (σ <selection condition> (R))

Ex. From the above table.

Q. Find name age address of students who age less than or equal to 25

Π stu_name, stu_address (σ age ≤ 25 (student))

Q. Find name of the students who age greater than 20 and of address "Patpa"

Π stu_name (σ age > 25 ∧ stu_address = "Patpa" (Studen

DATE

Ex: Assume the following table tbl_employee has the following attributes & tuples.

Name	Office	Dept	Rank
Smith	400	CS	Assistant
Jones	220	Econ	Adjunct
Greens	160	Econ	Assistant
Brown	420	CS	Associate
Smith	500	FIN	Associate

1. Select the employees in the CS department

=) $\sigma_{Dept = "CS"}(tbl\_employee)$

Name	Office	Dept	Rank
Smith	400	CS	Assistant
Brown	420	CS	Associate

2. select those employees with name smith and who are assistant

=) $\sigma_{Name = "Smith" \wedge Rank = "Assistant"}(tbl\_employee$

Name	Office	Dept	Rank
Smith	400	CS	Assistant

3. Project only name & department of the employee

=) $\Pi_{Name, Department}(tbl\_employee)$

Name	Dept
=	=

4. Show the Name of all employees working in CS department

=) $\Pi_{Name}(\sigma_{Dept = "CS"}(tbl\_employee))$

PAGE

classmate

DATE

* Sequence of operations & 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 to hold the intermediate results. Ex: To retrieve first name, last name & salary of all employees who work in department number 5, we must apply SELECT and PROJECT operation as follows

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

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, & using the assignment operation, denoted by ← (left arrow) as follow

DEP5_EMPS ← $\sigma_{DNO=5}(EMPLOYEE)$

RESULT ← $\pi_{Fname, Lname, Salary}(DEP5\_EMPS)$

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

RENAME Operation: Used to rename either the relation name or attribute names, or both as a unary operator.

CLASSMATE PAGE

DATE

* Relational Algebra Operations from set theory

(1) Union operation (∪).

→ Consider the following relations R and S. The union of relations R and S is ~~def~~ denoted by R∪S and it is a set of tuples that are either in R or S or in both. It returns the union of two compatible relation

→ For a union operation to be legal, we require that invoked relations must have the same number of attributes & corresponding attributes have same type.

$$R \cup S = \{t / t \in R \text{ or } t \in S\}$$

Ex: Consider two tables R & S.

R

A	B
α	1
α	2
β	1

S

A	B
α	2
β	3

Then R∪S =

A	B
α	1
α	2
β	1
β	3

R∩S =>

A	B
α	2

(2) Intersection operation (∩)

→ Set intersection is denoted by symbol ∩ it returns a relation that contains tuples that are in both of its argument relations. It is denoted by ∩ & defined as

$$R \cap S = \{t / t \in R \text{ and } t \in S\}$$

native same as union

PAGE

DATE

(3) Difference (-)

Set difference is denoted by the minus sign(-). It finds tuples that are in one relation, but not in another. Thus results in a relation containing tuples that are in R but not in S. It is denoted by R-S & is defined as: $R-S = \{t \mid t \in R \text{ and } t \notin S\}$

Nature same as Union.

Ex: R-S:

A	B
α	1
β	1

(4) Cartesian (X)

All of the above operations except Cartesian product Set operation take two input relations, which must be union compatible.

→ This type of operation is helpful to merge columns from two relations.

→ It doesn't require relations to union-compatible ie. the involved relations may have different schemas

→ It is denoted by R x S, is the set of all possible combinations of tuples of the two relations.

→ Formally, $R \times S = \{t\,q \mid t \in R \text{ and } q \in S\}$

Where R and S are any two relations & t and q are tuples of relation R and S respectively.

classmate

PAGE

DATE

Ex: Department

Dept_id	Dept_name	Dept_block no
1	Computer	100
2	Math	200

Staff

Staff id	Staff_name	Dept_id
11	Mohan	1
22	Prativa	2

Now, Department × Staff is:

D.Dept_id	Dept_name.	Dept_blockno	staff id	staff_na	S.Dept
1	Computer	100	11	Mohan	1
1	Computer	100	22	Prativa	2
2	Math	200	11	Mohan	1
2	Math	200	22	prativa	2

PAGE

classmate

DATE

* Binary relational operations: JOIN and DIVISON

These operations are used to perform operations into multiple tables. Set relational algebra operations are also called binary relational operations but here explain join operation & divison operations as binary relational operations

(1) JOIN Operation

Join operation is essentially a cartesian product followed by a selection criterion. Join operation is denoted by ⋈. It also allows joining variously related tuples from different relations

- a) Inner joins
  - Theta join
  - Equi join
  - Natural join
- b) Outer join.
  - Left outer Join
  - Right outer Join
  - Full outer Join.

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

- Theta join.

The general case of join operation is called a Theta join. It is denoted by symbol θ. It is an extension to the natural join operation that allows us to specify the join condition. It consists one of the comparision operators {=, <, <=, >, >=, <>}

DATE

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

ex: Department

Dept_id	Dept_name	Dept_block_no
1	Computer	100
2	Mathematics	200
3	Economics	300
4	Account	400
5	Physics	500

Staff

Staff_id	Staff_name	Dept_id
11	Mohan	1
22	Prabha	2
33	Madan	1
44	kamala	3
55	Sandhya	4
[redacted]	[redacted]	[redacted]

a) Find the information on every department dept_id greater than staff. dept_id.

⇒ Department $\bowtie$ D.Dept_id > S.Dept_id (Staff)

D.Dept_id	Dept_name	Dept_block_no	staff_id	staff_na	S.Dept_id
2	Mathematics	200	11	Mohan	1
2	Mathematics	200	33	Madan	1
3	Economics	300	11	Mohan	1
3	Economics	300	22	Prabha	2
3	Economics	300	33	Madan	1
4	Account	400	11	Mohan	1
4	Account	400	22	Prabha	2
4	Account	400	33	Madan	1
4	Account	400	44	kamala	3
5	Physics	500	11	Mohan	1
5	Physics	500	22	Prabha	2
5	Physics	500	33	Madan	1
5	Physics	500	44	Kamala	3
5	Physics	500	55	Sandhya	4

PAGE

classmate

0 Equi-Join.
When join condition is '=' ie Θ is =, the operation is called an equijoin. Syntax: $A \bowtie = B$

DATE

*) Find all the information on every department including regarding their ~~staff~~ dept.id

=) Department ⋈ D.Dept.id = STaff . Dept.id.

D.Dept.id	Dept_name	Dep_blockno	S.id	S.name	S.Dept.id
1	Computer	100	11	Mohan	1
1	Computer	100	33	Madan	1
2	Mathematics	200	22	Prathma	2
3	Economics	300	44	Kamala	3
4	Account	400	55	Sandhya	4

0 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 join performs a join by equating the attributes with the same name & then eliminates duplicate attributes

Ex: Lets take above two relations department & staff.

Department ⋈ Staff.

D.Dept.id	Dept_name	Dept_block_no	S.id	S.name
1	Computer	100	11	Mohan
1	Computer	100	33	Madan
2	Mathematics	200	22	Prathma
3	Economics	300	44	Kamala
4	Account	400	45	Sandhya

PAGE

DATE

→ 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.

o Left outer join (⟕):

It allows keeping all tuple in left relation However, if there is no matching type is found in the right relation, then the attributes of right relation in the join result are filled with NULL values

Ex: Department ⟕ staff.

D.Deptid	Dept.name	Dept_block.no	staff_id	staff_name	S.Dept_id
1	Computer	100	11	Mohan	1
1	Computer	100	33	Hadan	1
2	Mathematics	200	22	Prathima	2
3	Economics	300	44	Kamala	3
4	Account	400	55	Sandhya	4
5	Physics.	500	NULL	NULL	NULL

o Right Outer Join (⟖)

~~Replace left~~ Interchange left & right in Left over join then Department ⟖ staff. (Note: in qn, add 23 Ramesh 7 in staff)

D-Deptid	Dept_name	Dept_block no	staffid	staff_name	S.Dept_id
1	computer	100	11	Mohan	1
1	computer	100	33	Hadan	1
2	Math..	200	22	Prathima	2
3	Eco..	300	44	Kamala	3
4	Account	400	55	Sandhya	4
NULL	NULL	NULL	23	Ramesh	7

DATE

- Full outer join (⟗) (outer union)

It includes all tuples in the left hand relation & from the right hand relation. All tuples from both relations are included in the result, irrespective of the matching condition.

D.Deptid	Dept_name	Dept_block_no	staff_id	staff_name	S.Deptid
1	"	"	"	"	"
1	"	"	"	"	"
2	"	"	"	"	"
3	"	"	"	"	"
4	"	"	"	"	"
5	Physics	500	NULL	NULL	NULL
NULL	NULL	NULL	23	Ramesh	7

(2) Division operation (÷)

It is denoted by symbol ÷ and is suited to queries that include the phrase "for all". It takes two relations & builds another relation, consisting of values of an attribute of one relation that match all the values in the other relation.

Ex:

Sno	Pno
S_1	P_1
S_1	P_2
S_1	P_3
S_1	P_4
S_2	P_1
S_2	P_2
S_3	P_2
S_4	P_2
S_4	P_4

Pno
P_2

B_1

Pno
P_2
P_4

B_2

Pno
P_1
P_2
P_4

B_3

Sno
S_1
S_2
S_3
S_4

A/B_1

Sno
S_1
S_4

A/B_2

Sno
S_1

A/B_3

DATE

Ex: Assume the following two relations R(Subject) & S(course) respectively.

Subject name	Course
DBMS	CMP
C++	ELX
C++	CMP
OS	CMP

Course
CMP
ELX

Q. Retrive all the name of subject in all course where $R \div S$

=>

Subject name
C++

DATE

* Additional Relational operations.

Concept of Generalized Projection:

It is an extension of the projection operation that allows arithmetic functions to be used in the projection list. It is enhanced version of projection operation. It allows us to write projection operations containing attribute names & aromatic operations containing attribute names & constants in projection list.

General form of generalized projection is:

$\Pi_{F_1, F_2, F_3, \ldots, F_n}(E)$

Where, E is a relational algebra expression and $F_i (i = 1, 2, 3, \ldots n)$ is an attribute or arithmetic expression containing attributes and constants

Ex: Let's take an employee relation

eid	name	Age	salary
e_1	Rajan	33	34000
e_2	Aarav	19	45000
e_3	Abin	22	55000
e_4	Ashna	19	24000

Q. Find name & salary of all employees by increasing their salary by 15%.

⇒ Π name, salary = salary + salary * 0.15 (Employee)

Increase the salary of all employees whose age greater than 20 by 5%.

⇒ Π eid, name, age, salary = Salary + salary * 0.05 (Page > 20 (Employee))

classmate

PAGE

DATE

Aggregate functions

Aggregate functions are algebric functions that take a collection of values as input and return a single value as a result. It is denoted by Symbol $\mathcal{G}$ and read as "Calligraphic G".

General form of aggregate operation in relational algebra is:

$$G_1, G_2, \ldots G_n \; \mathcal{G} \; F_1(A_1), F_2(A_2), \ldots F_n(A_n)(E)$$

Where, E is any relational-algebra expression.
$G_1, G_2, \ldots G_n$ is a list of attributes on which to group (can be empty)
Each F_i is an aggregate function and each A_i is an attribute name.

→ All aggregate functions are deterministic.
→ There are five aggregate functions that are included with most relational database systems: SUM, AVG, MIN, MAX, COUNT.

Ex: Let's take an employee relation Employee.

Eid	Name	Address	Age	Salary
e_1	Rajan	ktm	33	34000
e_2	Aarav	Pokhara	17	45000
e_3	Abin	Palpa	22	55000
e_4	Ashna	ktm	19	24000
e_5	Manmohan	Pokhara	30	33000

① Find total no. of employees ⇒ $\mathcal{G}$ COUNT (Eid) (Employee)
② Find average age of employee of 'KTM' ⇒ $\mathcal{G}_{AVG}$ (Age) ($\sigma_{Address = "ktm"}$ (Employee)).
③ Find minimum & maximum age of student
$\mathcal{G}$ MIN(Age), MAX(Age) (Employee)

classmate PAGE

DATE

Q. Find average salary of employee in each address level.

Address $\mathcal{G}$ AVG(salary) (Employee)

Q. Find total salary of the employees

$\mathcal{G}$ SUM (Salary) (Employee)

¥ Relational Calculus

→ Relational calculus is a non procedural query language. It uses mathematical predicate calculus instead of algebra.

→ It provides the description about the query to get the result where as relational algebra gives the method to get the result.

→ It informs the system what to do with the relation, but doesnot inform how to perform it.

There are two forms:

- Tuple Relational Calculus (TRC)
- Domain Relational Calculus (DRC)

DATE

* The tuple Relational Calculus.

→ It is used for selecting those tuples that satisfy the given condition.

→ It is a logical language with variables ranging over tuples.

→ In tuple relational calculus, we work on filtering tuples based on the given condition.

{t | P(t)} or {t | condition (t)}.

where, t is the resulting tuples, P(t) is the condition used to fetch t. P(t) may have OR(V), AND(∧), NOT(¬), ∃, ∀.

→ In this form of relational calculus, we define a tuple variable, specify the relation name in which the tuple is to be searched for, along with the condition.

→ We can also specify column name using a dot operator with the tuple variable only to get a certain attribute (column) in result.

Eg: Given a relational schema:

Instructor (id, name, dept_name, salary)

i) Display all records from Instructor

⇒ {t | t ∈ Instructor}

ii) Find the id, name, department_name, salary for Instructor whose salary is greater than 80,000

⇒ {t | t ∈ Instructor ∧ t[salary] > 80000}

DATE

* Domain Relational Calculus

→ In Contrast to tuple relational calculus, domain relational calculus uses list of attribute to be selected from the relation based on the condition.

→ It is same as TRC, but differs by selecting the attributes rather than selecting the whole tuples. It is denoted as below.

$$\{ \langle a_1, a_2, a_3, \ldots a_n \rangle \mid P(a_1, a_2, a_3, \ldots a_n) \}$$

where $a_1, a_2, a_3, \ldots a_n$ are attributes of the relation & P is the condition

Ex: Consider the relation Loan

Loan (Loan_no, Branch_name, Amount)

Q. Find the loan_no, branch_name, amount of loans greater than or equal to 1000 amount

$$\{ \langle l, B, A \rangle \mid \langle l, B, A \rangle \in loan \wedge (A \geq 1000) \}$$

~~No.~~ l = loan_no
B = Branch_name
A = Amount

Another ex

- $\{ \langle name, age \rangle \mid \in Student \wedge age > 17 \}$.
  This query will return names & ages of the students in the table student who are older than 17

- $\{ \langle Sid, name, address \rangle \mid \langle sid, name, address \rangle \in STUDENT \}$
  This will return all records of student relation

DATE

o) {<name> | <sid, name, address> ∈ STUDENT ∧ Sid = 45}
Find the name of the student whose id is 45.

Difference between Relational algebra & relational calculus

Relational algebra	Relational calculus
i) It is a procedural language.	i) It is a declarative language.
ii) Relational algebra 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 dependant.	iv) It can be domain dependant.
v) It is close to a programming language.	v) It is close to the natural language.

classmate

PAGE

DATE

Questions asked from this Chapter in board Exam

Q. Explain the difference between "Join" and "Natural Join" of algebric operators with example. (2078-5 marks)

Q. Consider the following Supplier database, where Primary keys are underlined. (2070-5 marks)

Supplier (S_id, S_name, city)
Supplies (S_id, part_id, quantity)
Parts (part_id, part_name, color, weight)

Construct the following relational algebra queries for this relational database

a) Find the name of all Suppliers located in city "ktm".

=) ~~$\sigma_{city = "ktm"}$ (supplier)~~
$\Pi_{S\_name}(\sigma_{city = "ktm"}(supplier))$.

b) Find the name of all parts that are supplied by "ABC Company"

=) $\Pi_{part\_name}(\sigma_{S\_name = "ABC\ Company"}(part \bowtie Supplies \bowtie supplier))$

c) Find the name of all parts that are applied in quantity greater than 100

=) $\Pi_{part\ name}(\sigma_{quantity > 100}($ ~~parts ⋈ supplies ⋈ supplies ⋈ parts ⋈~~ supplies$)$

d) Find the number of parts supplied by "ABC Company".

=) $G_{count(part_id)}(\sigma_{company = "ABC\ Company"}($parts ⋈ supplies ⋈ comp.$))$

e) Find the name of all the suppliers who supply more than 30 different parts

=) $\Pi_{S_name}(\sigma_{quantity > 30}($supplier ⋈ supplies ⋈ parts$))$.