Relational Algebra

By :
Dr. Rinkle Rani
Associate Professor, CSED
TIET, Patiala

Relational Query Languages

- Languages for describing queries on a relational database
- *Structured Query Language* (SQL)
  - Predominant application-level query language
  - Declarative
- *Relational Algebra*
  - Intermediate language used within DBMS
  - Procedural

Formal Relational Query Languages

Two mathematical Query Languages form the basis for "real" languages (e.g. SQL), and for implementation:

1. *Relational Algebra*: More operational, very useful for representing execution plans.
2. *Relational Calculus*: Lets users describe what they want, rather than how to compute it. (Non-operational, *declarative*.)

Relational Algebra

A **query language** is a language in which user requests information from the database. it can be categorized as either **procedural** or **nonprocedural.**

In a procedural language the user instructs the system to do a sequence of operations on database to compute the desired result.

In nonprocedural language the user describes the desired information without giving a specific procedure for obtaining that information.

The relational algebra is a procedural query language. It consists of a set of operations that take one or two relations as input and produces a new relation as output.

Types of operations in relational algebra

We have divided these operations in two categories:

1. Basic Operations
2. Derived Operations

Basic/Fundamental Operations:

1. Select (σ)
2. Project (∏)
3. Union (∪)
4. Set Difference (-)
5. Cartesian product (X)
6. Rename (ρ)

Derived Operations:

1. Natural Join (⋈)
2. Intersection (∩)
3. Division (÷)

Select Operator (σ)

Select Operator is denoted by sigma (σ) and it is used to find the tuples (or rows) in a relation (or table) which satisfy the given condition.

it works as the where clause in SQL, which is used for the same purpose.

Syntax of Select Operator (σ)

σ Condition/Predicate(Relation/Table name)

$$\sigma_{condition}(relation)$$

- Example:

Person

Id	*Name*	*Address*	*Hobby*
1123	John	123 Main	stamps
1123	John	123 Main	coins
5556	Mary	7 Lake Dr	hiking
9876	Bart	5 Pine St	stamps

$\sigma_{Hobby=\text{'stamps'}}(\text{Person})$

Id	*Name*	*Address*	*Hobby*
1123	John	123 Main	stamps
9876	Bart	5 Pine St	stamps

Selection Condition - Examples

- $\sigma_{\mathit{Id}>3000 \text{ OR } \mathit{Hobby}=\text{'hiking'}}(\text{Person})$
- $\sigma_{\mathit{Id}>3000 \text{ AND } \mathit{Id}<3999}(\text{Person})$
- $\sigma_{\text{NOT}(\mathit{Hobby}=\text{'hiking'})}(\text{Person})$
- $\sigma_{\mathit{Hobby}\neq\text{'hiking'}}(\text{Person})$

Project Operator (∏)

Project operator is denoted by ∏ symbol and it is used to select desired columns (or attributes) from a table (or relation).

Project operator in relational algebra is similar to the Select statement in SQL.

Syntax of Project Operator (∏)

∏ column_name1, column_name2, ..., column_nameN(table_name)

$$\pi_{attribute\ list}(relation)$$

- Example:

Person

Id	*Name*	*Address*	*Hobby*
1123	John	123 Main	stamps
1123	John	123 Main	coins
5556	Mary	7 Lake Dr	hiking
9876	Bart	5 Pine St	stamps

$\pi_{Name,Hobby}(\text{Person})$

Name	*Hobby*
John	stamps
John	coins
Mary	hiking
Bart	stamps

Project Operator

- Example:

Person

Id	*Name*	*Address*	*Hobby*
1123	John	123 Main	stamps
1123	John	123 Main	coins
5556	Mary	7 Lake Dr	hiking
9876	Bart	5 Pine St	stamps

$\pi_{Name,Address}$(Person)

Name	*Address*
John	123 Main
Mary	7 Lake Dr
Bart	5 Pine St

Result is a table (no duplicates); can have fewer tuples than the original

Expressions

$$\pi_{Id,\ Name}\,(\sigma_{Hobby=\text{'stamps'}\ \text{OR}\ Hobby=\text{'coins'}}\,(\text{Person})\,)$$

Id	*Name*	*Address*	*Hobby*
1123	John	123 Main	stamps
1123	John	123 Main	coins
5556	Mary	7 Lake Dr	hiking
9876	Bart	5 Pine St	stamps

Person

Id	*Name*
1123	John
9876	Bart

Result

Set Operators

- Relation is a set of tuples, so set operations should apply: $\cap$, $\cup$, $-$ (set difference)
- Result of combining two relations with a set operator is a relation => all its elements must be tuples having same structure
- Hence, scope of set operations limited to *union compatible relations*

Union Compatible Relations

- Two relations are *union compatible* if
  - Both have same number of columns
  - Names of attributes are the same in both
  - Attributes with the same name in both relations have the same domain
- Union compatible relations can be combined using *union*, *intersection*, and *set difference*

The union operation: - is used when we need some attributes that appear in either or both of the two relations.
It is denoted as **U**.

For a union operation **r U s** to be valid, two conditions must hold:

1. The relation r and s must be of the same arity, i.e. they must have the same number of attributes.
2. The domains of the ith attribute of r and the ith attribute of s must be the same for all i.

Example:
Borrower (customer-name, loan-number)
Depositor (customer-name, account-number)
Customer (customer-name, street-number, customer-city)

List all the customers who have either an account or a loan or both
Code:

Π customer-name (Borrower) U Π customer-name (Depositor)

Intersection Operator (∩)

Intersection operator is denoted by ∩ symbol and it is used to select common rows (tuples) from two tables (relations).

Table 1: COURSE

Course_Id	Student_Name	Student_Id
C101	Aditya	S901
C104	Aditya	S901
C106	Steve	S911
C109	Paul	S921
C115	Lucy	S931

Table 2: STUDENT

Student_Id	Student_Name	Student_Age
S901	Aditya	19
S911	Steve	18
S921	Paul	19
S931	Lucy	17
S941	Carl	16
S951	Rick	18

```
∏ Student_Name (COURSE) ∩ ∏ Student_Name (STUDENT)
```

Output:

Student_Name
Aditya
Steve
Paul
Lucy

The set difference operation: - finds tuples that in one relation but not in other.
It is denoted as **–**

Example:
Find the names of all customers who have an account but not a loan.
Code:
Π customer-name (Depositor) - Π customer-name (Borrower)

Cartesian product (X)

Cartesian Product is denoted by X symbol. Lets say we have two relations R1 and R2 then the cartesian product of these two relations (R1 X R2) would combine each tuple of first relation R1 with the each tuple of second relation R2. I know it sounds confusing but once we take an example of this, you will be able to understand this.

Syntax of Cartesian product (X)

R1 X R2

Table 1: R

Col_A	Col_B
-----	------
AA	100
BB	200
CC	300

Table 2: S

Col_X	Col_Y
-----	-----
XX	99
YY	11
ZZ	101

```
R X S
```

Output:

Col_A	Col_B	Col_X	Col_Y
-----	------	------	------
AA	100	XX	99
AA	100	YY	11
AA	100	ZZ	101
BB	200	XX	99
BB	200	YY	11
BB	200	ZZ	101
CC	300	XX	99
CC	300	YY	11
CC	300	ZZ	101

Rename (ρ) Example

Lets say we have a table customer, we are fetching customer names and we are renaming the resulted relation to CUST_NAMES.

$\rho(CUST\_NAMES, \prod(Customer\_Name)(CUSTOMER))$

Table: CUSTOMER

Customer_Id	Customer_Name	Customer_City
C10100	Steve	Agra
C10111	Raghu	Agra
C10115	Chaitanya	Noida
C10117	Ajeet	Delhi
C10118	Carl	Delhi

Output

CUST_NAMES
Steve
Raghu
Chaitanya
Ajeet
Carl

Division

- Goal: Produce the tuples in one relation, r, that match *all* tuples in another relation, s
  - $r\ (A_1, \ldots A_n, B_1, \ldots B_m)$
  - $s\ (B_1 \ldots B_m)$
  - r/s, with attributes $A_1, \ldots A_n$, is the set of all tuples $<a>$ such that for every tuple $<b>$ in s, $<a,b>$ is in r

It is denoted as ÷.

Examples of Division A/B

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

A

pno
p2

B1

pno
p2
p4

B2

pno
p1
p2
p4

B3

sno
s1
s2
s3
s4

A/B1

sno
s1
s4

A/B2

sno
s1

A/B3

Natural Join (⋈)

Natural join is a binary operator. Natural join between two or more relations will result set of all combination of tuples where they have equal common attribute. Eg **Emp ⋈ Dep**

Emp

Name	Id	Dept_name
A	120	IT
B	125	HR
C	110	Sale
D	111	IT

Dep

Dept_name	Manager
Sale	Y
Prod	Z
IT	A

Emp ⋈ Dep

Name	Id	Dept_name	Manager
A	120	IT	A
C	110	Sale	Y
D	111	IT	A

Consider following database

Sailors(sid: integer, *sname: string, rating: integer, age: real)*

Boats(*bid: integer, bname: string, color: string)*

Reserves *(sid: integer, bid: integer, day: date)*

sid		rating	age
22	Dustin	7	45.0
29	Brutus	1	33.0
31	Lubber	8	55.5
32	Andy	8	25.5
58	Rusty	10	35.0
64	Horatio	7	35.0
71	Zorba	10	16.0
74	Horatio	9	35.0
85	Art	3	25.5
95	Bob	3	63.5

sid	bid	day
22	101	10/10/98
22	102	10/10/98
22	103	10/8/98
22	104	10/7/98
31	102	11/10/98
31	103	11/6/98
31	104	11/12/98
64	101	9/5/98
64	102	9/8/98
74	103	9/8/98

bid	bname	color
101	Interlake	blue
102	Interlake	red
103	Clipper	green
104	Marine	red

Find out all sailor names and ratings.

$$\pi_{sname,rating}(\text{Sailors})$$

Find names of sailors who've reserved boat #103

- Solution
- :

$$\pi_{sname}(\sigma_{bid=103}(\mathrm{Re}serves \bowtie Sailors))$$

Find names of sailors who've reserved a red boat

Å Information about boat color only available in Boats; so need an extra join:

$$\pi_{sname}((\sigma_{color='red'} Boats) \bowtie \text{Re}serves \bowtie Sailors)$$

❖ OR A more efficient solution:

$$\pi_{sname}(\pi_{sid}((\pi_{bid}\sigma_{color='red'} Boats) \bowtie \text{Re}s) \bowtie Sailors)$$