Relational Algebra

Definition

Relational algebra 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 either **unary** or **binary**. They accept relations as their input and yield relations as their output. Relational algebra is performed recursively on a relation and intermediate results are also considered relations.

Relational Algebra

- The basic set of operations for the relational model is the relational algebra.
  - enable the specification of basic retrievals
- The result of a retrieval is a new relation, which may have been formed from one or more relations.
  - algebra operations thus produce new relations, which can be further manipulated the same algebra.
- A sequence of relational algebra operations forms a relational algebra expression,
  - the result will also be a relation that represents the result of a database query (or retrieval request).

What is an Algebra?

- A language based on *operators* and a *domain of values*
- Operators map values taken from the domain into other domain values
- Hence, an expression involving operators and arguments produces a value in the domain
- When the domain is a set of all relations we get the *relational algebra*

Relational Algebra Definitions

- *Domain*: set of relations
- *Basic operators*: select, project, union, set difference, Cartesian (cross) product
- *Derived operators*: set intersection, division, join
- *Procedural*: Relational expression specifies query by describing an algorithm (the sequence in which operators are applied) for determining the result of an expression

Unary Relational Operations

- **SELECT Operation:** used to select a *subset* of the tuples from a relation that satisfy a **selection condition**. It is a filter that keeps only those tuples that satisfy a qualifying condition.

  Examples:

  $\sigma_{DNO = 4}(EMPLOYEE)$

  $\sigma_{SALARY > 30,000}(EMPLOYEE)$

  – denoted by $\sigma_{<\text{selection condition}>}(R)$ where the symbol σ (sigma) is used to denote the select operator, and the selection condition is a *Boolean expression* specified on the attributes of relation R

SELECT Operation Properties

The SELECT operation $\sigma_{<\text{selection condition}>}(R)$ produces a relation S that has the same schema as R

The SELECT operation **σ** is **commutative;** i.e.,

$$\sigma_{<\text{condition1}>}(\sigma_{<\text{condition2}>}(R)) = \sigma_{<\text{condition2}>}(\sigma_{<\text{condition1}>}(R))$$

A cascaded SELECT operation **may be applied in any order;** i.e.,

$$\sigma_{<\text{condition1}>}(\sigma_{<\text{condition2}>}(\sigma_{<\text{condition3}>}(R)) = \sigma_{<\text{condition2}>}(\sigma_{<\text{condition3}>}(\sigma_{<\text{condition1}>}(R)))$$

A cascaded SELECT operation may be replaced by a single selection with a conjunction of all the conditions**;** i.e.,

$$\sigma_{<\text{condition1}>}(\sigma_{<\text{condition2}>}(\sigma_{<\text{condition3}>}(R)) = \sigma_{<\text{condition1}> \text{ AND } <\text{condition2}> \text{ AND } <\text{condition3}>}(R)))$$

Selection Condition

- Operators: $<$, $\leq$, $\geq$, $>$, $=$, $\neq$
- Simple selection condition:
  - *<attribute> operator <constant>*
  - *<attribute> operator <attribute>*
  - *<condition>* AND *<condition>*
  - *<condition>* OR *<condition>*
  - NOT *<condition>*

Select Examples

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

$\sigma_{Id>3000 \text{ OR } Hobby='hiking'}(\text{Person})$

$\sigma_{Id>3000 \text{ AND } Id<3999}(\text{Person})$

$\sigma_{\text{NOT}(Hobby='hiking')}(\text{Person})$

$\sigma_{Hobby\neq'hiking'}(\text{Person})$

Unary Relational Operations (cont.)

- **PROJECT Operation:** selects certain *columns* from the table and discards the others.

  Example:

$$\pi_{\text{LNAME, FNAME,SALARY}}(\text{EMPLOYEE})$$

  The general form of the project operation is:

  π<attribute list>(R) where π is the symbol used to represent the project operation and <attribute list> is the desired list of attributes.

  PROJECT *removes duplicate tuples,* so the result is a set of tuples and hence a valid relation.

PROJECT Operation Properties

The number of tuples in the result of $\pi_{\text{<list>}}(R)$ is always less or equal to the number of tuples in R.

If attribute list includes a key of R, then the number of tuples is equal to the number of tuples in R.

$\pi_{\text{<list1>}}(\pi_{\text{<list2>}}(R)) = \pi_{\text{<list1>}}(R)$ as long as <list2> contains the attributes in <list1>

SELECT and PROJECT Operations

(a) $\sigma_{(DNO=4\ AND\ SALARY>25000)\ OR\ (DNO=5\ AND\ SALARY>30000)}(EMPLOYEE)$

(b) $\pi_{LNAME,\ FNAME,\ SALARY}(EMPLOYEE)$

(c) $\pi_{SEX,\ SALARY}(EMPLOYEE)$

(a)

FNAME	MINIT	LNAME	SSN	BDATE	ADDRESS	SEX	SALARY	SUPERSSN	DNO
Franklin	T	Wong	333445555	1955-12-08	638 Voss,Houston,TX	M	40000	888665555	5
Jennifer	S	Wallace	987654321	1941-06-20	291 Berry,Bellaire,TX	F	43000	888665555	4
Ramesh	K	Narayan	666884444	1962-09-15	975 FireOak,Humble,TX	M	38000	333445555	5

(b)

LNAME	FNAME	SALARY
Smith	John	30000
Wong	Franklin	40000
Zelaya	Alicia	25000
Wallace	Jennifer	43000
Narayan	Ramesh	38000
English	Joyce	25000
Jabbar	Ahmad	25000
Borg	James	55000

(c)

SEX	SALARY
M	30000
M	40000
F	25000
F	43000
M	38000
M	25000
M	55000

Relational Algebra Operations from Set Theory

- The UNION, INTERSECTION, and MINUS Operations
- The CARTESIAN PRODUCT (or CROSS PRODUCT) Operation

Set Operators

- A relation is a *set* of tuples, so set operations apply:

  $\cap$, $\cup$, $-$ (set difference)

- Result of combining two relations with a set operator is a relation => all elements are tuples with the same structure

UNION Operation

Denoted by R ∪ S

Result is a relation that includes all tuples that are either in R or in S or in both. Duplicate tuples are eliminated.

Example: Retrieve the SSNs of all employees who either work in department 5 or directly supervise an employee who works in department 5:

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

RESULT1 ← π_{SSN}(DEP5_EMPS)

RESULT2(SSN) ← $\pi_{SUPERSSN}$(DEP5_EMPS)

RESULT ← RESULT1 ∪ RESULT2

The union operation produces the tuples that are in either RESULT1 or RESULT2 or both. The two operands must be "type compatible".

UNION Operation

Type (Union) Compatibility

The operand relations $R_1(A_1, A_2, ..., A_n)$ and $R_2(B_1, B_2, ..., B_n)$ must have the same number of attributes, and the domains of corresponding attributes must be compatible, i.e.

- $dom(A_i) = dom(B_i)$ for i=1, 2, ..., n.

Example

Tables:

Person (*SSN, Name, Address, Hobby*)

Professor (*Id, Name, Office, Phone*)

are not union compatible.

But

π_{Name} (Person) and π_{Name} (Professor)

are union compatible so

π_{Name} (Person) - π_{Name} (Professor)

makes sense.

UNION Example

STUDENT ∪ INSTRUCTOR:

STUDENT	FN	LN
	Susan	Yao
	Ramesh	Shah
	Johnny	Kohler
	Barbara	Jones
	Amy	Ford
	Jimmy	Wang
	Ernest	Gilbert

INSTRUCTOR	FNAME	LNAME
	John	Smith
	Ricardo	Browne
	Susan	Yao
	Francis	Johnson
	Ramesh	Shah

(b)

FN	LN
Susan	Yao
Ramesh	Shah
Johnny	Kohler
Barbara	Jones
Amy	Ford
Jimmy	Wang
Ernest	Gilbert
John	Smith
Ricardo	Browne
Francis	Johnson

What would STUDENT ∩ INSTRUCTOR be?

Set Difference Operation

Set Difference (or MINUS) Operation

The result of this operation, denoted by R - S, is a relation that includes all tuples that are in R but not in S. The two operands must be "type compatible”.

Set Difference Example

S1

SID	SName	Age
473	Popeye	22
192	Jose	22
715	Alicia	28
914	Hal	24

S2

SID	SName	Age
202	Rusty	21
403	Marcia	20
914	Hal	24
192	Jose	22
881	Stimpy	19

Relational Algebra Operations From Set Theory (cont.)

- Union and intersection are *commutative operations:*

  R ∪ S = S ∪ R, and R ∩ S = S ∩ R

- Both union and intersection can be treated as n-ary operations applicable to any number of relations as both are *associative operations;* that is

  R ∪ (S ∪ T) = (R ∪ S) ∪ T, and

  (R ∩ S) ∩ T = R ∩ (S ∩ T)

- The minus operation is *not commutative;* that is, in general

  R - S ≠ S – R

Cartesian (Cross) Product

- If *R* and *S* are two relations, $R \times S$ is the set of all concatenated tuples $<x,y>$, where *x* is a tuple in *R* and *y* is a tuple in *S*
  - *R* and *S* need not be union compatible
- $R \times S$ is expensive to compute:
  - Factor of two in the size of each row; Quadratic in the number of rows

A	*B*
x1	x2
x3	x4

R

C	*D*
y1	y2
y3	y4

S

A	*B*	*C*	*D*
x1	x2	y1	y2
x1	x2	y3	y4
x3	x4	y1	y2
x3	x4	y3	y4

$R \times S$

Cartesian Product

	R1	R2	R1*R2
Attributes	X	Y	X+Y
Tuples	N1	N2	N1*N2

JOIN Operation

Join in relational Algebra

Join is a combination of a Cartesian product followed by a selection process.

A Join operation pairs two tuples from different relations, if and only if a given join condition is satisfied.

Various forms of join operation are:

- Inner Joins:
  - Theta join
  - EQUI join
  - Natural join
- 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(θ)

- The general case of JOIN operation is called a Theta join.
- It is denoted by symbol **θ.**
- Also Known as Conditional Join.
- Used when you want to join two or more relation based on some conditions.

Example

$$P \bowtie_{\theta} Q$$

Customer

Cid	Cname	Age
101	Ajay	20
102	Vijay	19
103	Sita	21

Order

Oid	Oname
101	Pizza
101	Noodles
103	Burger

$$\text{Customer} \bowtie_{\text{Customer.cid}>\text{Order.oid}} \text{Order}$$

Cid	Cname	Age	Oid	Oname
102	Vijay	19	101	Pizza
102	Vijay	19	101	Noodles
103	Sita	21	101	Pizza
103	Sita	21	101	Noodles

$$\sigma_{\text{customer.cid}>\text{order.oid}}(\text{customer} \times \text{order})$$

Equi Join

Equi join(⋈)

- Equijoin is a **special case of conditional join**
- When a theta join uses only equivalence condition, it becomes a equi join.
- As values of two attributes will be equal in result of equijoin, only one attribute will be appeared in result.

Example

P ⋈ Q

Customer

Cid	Cname	Age
101	Ajay	20
102	Vijay	19
103	Sita	21

Order

Oid	Oname
101	Pizza
101	Noodles
103	Burger

Customer $\bowtie_{Customer.cid=Order.oid}$ Order

Cid	Cname	Age	Oname
101	Ajay	20	Pizza
101	Ajay	20	Noodles
103	Sita	21	Burger

Πcustomer.cid,customer,cname,austomer.age,order.oname
(σcustomer.cid=order.oid (customer X order))

Join-Natural Join (⋈)

- Natural join can only be performed if there is a common attribute (column) between the relations.
- The name and type (domain) of the attribute must be same.
- Natural join does not use any comparison operator.
- It does not concatenate the way a Cartesian product does.
- Natural Join will also return the similar attributes only once as their value will be same in resulting relation.
- It is a special case of equijoin in which equality condition hold on all attributes which have same name in relations R and S.

Customer

Cid	Cname	Age
101	Ajay	20
102	Vijay	19
103	Sita	21
104	Gita	22

Order

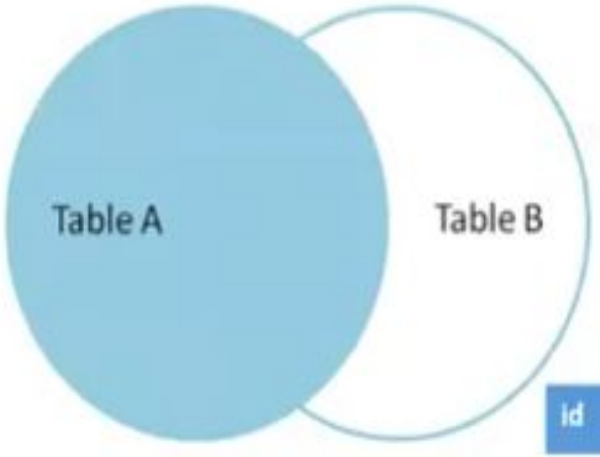
Cid	Oname	Cost
101	Pizza	500
103	Noodles	300
108	Burger	99

Customer ⋈ Order

Cid	Cname	Age	Oname	Cost
101	Ajay	20	Pizza	500
103	Sita	21	Noodles	300

LEFT JOIN (⟕)

- This join returns all the rows of the table on the left side of the join and matching rows for the table on the right side of join.
- The rows for which there is no matching row on right side, the result-set will contain *null*.
- LEFT JOIN is also known as LEFT OUTER JOIN.



A

Id	Name
10	Jay
20	Veer
30	John

B

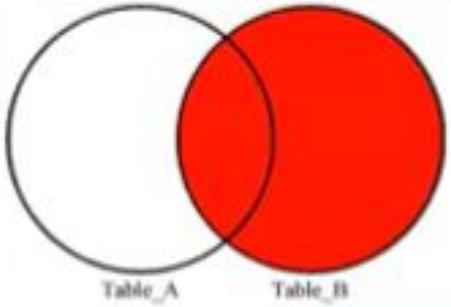
Name	Marks
Rohan	20
Veer	18
John	14
Sam	13

A ⟕ B

id	Name	Marks
10	Jay	NULL
20	Veer	18
30	John	14

RIGHT JOIN(⋈)

- RIGHT JOIN is similar to LEFT JOIN.
- This join returns all the rows of the table on the right side of the join and matching rows for the table on the left side of join.
- The rows for which there is no matching row on left side, the result-set will contain *null*.
- RIGHT JOIN is also known as RIGHT OUTER JOIN.



A

Id	Name
10	Jay
20	Veer
30	John

B

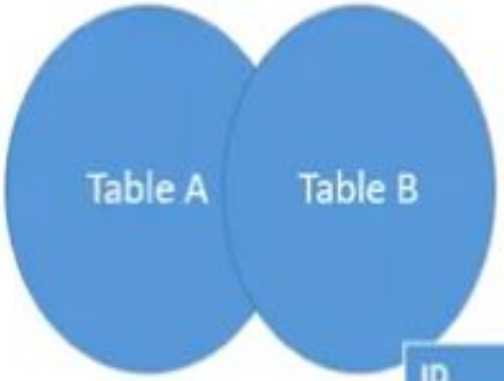
Name	Marks
Rohan	20
Veer	18
John	14
Sam	13

A ⋈ B

id	Name	Marks
Null	Rohan	20
20	Veer	18
30	John	14
Null	Sam	13

FULL JOIN (⋈)

- FULL OUTER JOIN creates the result-set by combining result of both LEFT JOIN and RIGHT JOIN.
- The result-set will contain all the rows from both the tables.
- The rows for which there is no matching, the result-set will contain *NULL* values.



A ⋈ B

ID	Name	Marks
10	Jay	NULL
20	Veer	18
30	John	14
NULL	Rohan	20
Null	Sam	13

A

Id	Name
10	Jay
20	Veer
30	John

B

Name	Marks
Rohan	20
Veer	18
John	14
Sam	13

Division Operator (/, ÷)

- Binary , Derived Operator.
- Division operator A÷B can be applied if and only if:
- Attributes of B is proper subset of Attributes of A.
- The relation returned by division operator will have attributes = (All attributes of A – All Attributes of B)
- The relation returned by division operator will return those tuples from relation A which are associated to every B's tuple.

Student

ID No	Sports
101	Chess
102	Hockey
102	Chess
104	Chess

Sports_detail

Sports
Chess
Hockey

Student÷Sports_detail

ID No
102

Division Examples

Division Operator (/, ÷)

- Case 3:

X	Y
X1	Y1
X2	Y2
X1	Y2
X4	y4

÷

X
X1

=

Y
Y1
Y2

- Case 4:

X	Y
X1	Y1
X2	Y2
X1	Y2
X4	y4

÷

Y
Y1
Y2
Y3
Y4

=

X
Null