Chapter 2

Relational Algebra - Operators

Outline

Relational Algebra

- Unary Relational Operations
- Relational Algebra Operations From Set Theory
- Binary Relational Operations
- Additional Relational Operations
- Examples of Queries in Relational Algebra

Introduction

- **Cartesian product** of two relations (A x B), gives us all the possible tuples that are paired together.
  - But it **might not be feasible** in certain cases to take a **Cartesian product** where we **encounter huge relations** with **thousands of tuples** having a considerable **large number of attributes.**

Join Operation ($\bowtie$)

- Join is an additional or derived operator which simplify the queries, but does not add any new power to the basic relational algebra.
- Join = cartesian product + selection
- Join pairs two tuples from different relations, if and if a given condition is satisfied.

Symbol: $\bowtie$

$$A \bowtie_c B = \sigma_c (A \times B)$$

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.

Difference

Joins (⋈)

- Combination of tuples that satisfy the filtering/matching conditions
- Fewer tuples then cross product, might be able to compute efficiently

Cartesian Product /Cross Product/Cross Join(X)

- All possible combination of tuples from the relations
- Huge number of tuples and costly to manage

R

A	B
1	a
2	b

S

B	C
a	3
b	4

R ⋈ S (Natural Join)

A	B	C
1	a	3
2	b	4

R x S

A	R.B	S.B	C
1	a	a	3
1	a	b	4
2	b	a	3
2	b	b	4

Join Types..

1. Inner Join:

- Contains only those tuples that satisfy the matching condition

- Theta(θ) / Conditional Join
  - $A \bowtie_\theta B$
  - uses all kinds of comparison operators($<, >, <=, >=, =, \neq$)
- Equi Join
  - Special case of theta join
  - uses only equality (=) comparison operator
- Natural join
  - $A \bowtie B$
  - Based on common attributes in both relation
  - does not use any comparison operator

2. Outer join:

- Contains matching tuples that satisfy the matching condition, along with some or all tuples that do not satisfy the matching condition
- Contains all rows from either one or both relation

- Left Outer Join
  - Left relation tuples will always be in result whether the value is matched or not
- Right Outer Join
  - Right relation tuples will always be in result whether the value is matched or not
- Full Outer Join
  - Tuples from both relations are present in result, whether the value is matched or not

Natural Join

- **Natural join** can only be performed ***if there is at least*** **one common attribute** (column) that exist between two relations. In addition, the **attributes** must have the **same name and domain.**
- Natural join does not use any comparison operator.
- ***It is same as equi join which occurs implicitly*** *by comparing all the* ***common attributes (columns)*** *in both relation,* **but** difference is that ***in Natural Join the common attributes appears only once.*** *The resulting schema will change.*
- **Notation**: $A \bowtie B$
- *The* ***result of the natural join*** *is the* ***set of all combinations of tuples*** *in two relations* ***A*** *and* ***B*** *that are* ***equal on their common attribute names.***

Natural Join..

Note:

- The **Natural Join** of two relations can be ***obtained*** *by applying a* ***Projection operation to Equi join*** *of two relations*. In terms of basic operators:

  Natural Join = Cartesian product + Selection + Projection

- **Natural Join (⋈) is by default inner join** because the tuples which does not satisfy the conditions of join does not appear in result set.
- **Natural Join** is very important.

Natural Join Example

Courses

Cid	Cname	Dept
EC101	Electronics	ECE
CS201	DBMS	CSE
ME301	Design	MECH

Faculty

Dept	Facultyname
ECE	A
CSE	B
MECH	C

Courses $\bowtie$ Faculty

Cid	Cname	Dept	Facultyname
EC101	Electro nics	ECE	A
CS201	DBMS	CSE	B
ME301	Design	MECH	C

Equivalent to:

π CID, Cname, dept, Facultyname(σCourses. Dept=Facultyname. dept(Courses X Faculty))

Query 1: Find the name of all the customers who have loan at the bank, along with the loan number and loan amount.

Table:

- branch(branch-name, branch-city, assets)
- customer(customer-name, customer-street, customer-city)
- account(account-number, branch-name, balance)
- loan(loan-number, branch-name, amount)
- depositor(customer-name, account-number)
- borrower(customer-name, loan-number)

Using Natural Join:

π customer-name, loan.loan-number, amount(borrower $\bowtie$ loan)

Using Cartesian Product:

π customer-name, loan.loan-number, amount(σ borrower. loan-number = loan.loan-number(borrower X loan)

Query 1: Find the names of all branches with customers who have an account in the bank and who live in Hubli.

Table

- branch(branch-name, branch-city, assets)
- customer(customer-name, customer-street, customer-city)
- account(account-number, branch-name, balance)
- loan(loan-number, branch-name, amount)
- depositor(customer-name, account-number)
- borrower(customer-name, loan-number)

- π branch-name(customer-city=“hubli” (customer $\bowtie$ account $\bowtie$ depositor))
- Natural Join is associative:

((customer $\bowtie$ account) $\bowtie$ depositor) = (customer $\bowtie$ (account $\bowtie$ depositor))

hence

- (customer $\bowtie$ account $\bowtie$ depositor)

Query 1: Find the customers who have both loan and an account at the bank.

Table:

- branch(branch-name, branch-city, assets)
- customer(customer-name, customer-street, customer-city)
- account(account-number, branch-name, balance)
- loan(loan-number, branch-name, amount)
- depositor(customer-name, account-number)
- borrower(customer-name, loan-number)

- π customer-name (borrower $\bowtie$ depositor)

- **Theta join / Conditional Join**
  - It combines tuples from different relations provided they satisfy the **theta (θ) condition.**
  - It is a **general case of join.** And it is **used** when we want to **join** two or more relation based on some **conditions.**
  - The **join condition** is denoted by the symbol θ.
  - It **uses** all kinds of **comparison operators** like $<, >, <=, >=, =, \neq$
- **Notation**: $A \bowtie_{\theta} B$

  Where θ is a **predicate/condition.** It can use any comparison operator ($<, >, <=, >=, =, \neq$)
- $A \bowtie_{\theta} B = \sigma_{\theta}(A \times B)$

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

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

Theta Join Example:

S1

sid	name	rating	age
22	dustin	7	45.0
31	lubber	8	55.0
58	rusty	10	35.0

R1

sid	bid	day
22	101	10/10/96
58	103	11/12/96

$S1 \bowtie_{S1.sid < R1.sid} R1$

S1.Sid	sname	rating	age	R1.sid	bid	day
22	dustin	7	45.0	58	103	11/12/96
31	lubber	8	55.0	58	103	11/12/96

Equivalent to: $\sigma_{S1.sid < R1.sid}(S1 \times R1)$

So, $A \bowtie_{\theta} B = \sigma_{\theta}(A \times B)$

Example:Theta Join

S1

sid	name	rating	age
22	dustin	7	45.0
31	lubber	8	55.0
58	rusty	10	35.0

R1

sid	bid	day
22	101	10/10/96
58	103	11/12/96

$S1 \bowtie_{S1.sid < R1.sid} R1$

S1.Sid	sname	rating	age	R1.sid	bid	day
22	dustin	7	45.0	58	103	11/12/96
31	lubber	8	55.0	58	103	11/12/96

Equivalent to: $\sigma_{S1.sid < R1.sid}(S1 \times R1)$

So, $A \bowtie_{\theta} B = \sigma_{\theta}(A \times B)$

Equi Join:

- When a theta join uses only equivalence (=) condition, it becomes a **Equi join**.
- **Equi join** is a **special case of theta (or conditional) join** where condition contains ***equalities*** (=).
- ***Notation:*** $A \bowtie_{A.a1 = B.b1 \wedge \ldots\ldots \wedge A.an = B.bn} B$

Example 1: Equi join

S1

sid	name	rating	age
22	dustin	7	45.0
31	lubber	8	55.0
58	rusty	10	35.0

R1

sid	bid	day
22	101	10/10/96
58	103	11/12/96

$$S1 \bowtie_{S1.sid = R1.sid} R1$$

S1.Sid	sname	rating	age	R1.sid	bid	day
22	dustin	7	45.0	22	101	10/10/96
58	rusty	10	35.0	58	103	11/12/96

Equivalent to $\sigma_{S1.sid = R1.sid}(S1 \times R1)$

Example 2: Equi join

Student

SID	Name	Std
101	Rohan	11
102	Mira	12

Subjects

Class	Subject
11	Maths
11	Physics
12	English
12	Chemistry

$$Student \bowtie_{Student.\,Std\,=\,Subjects.\,Class} Subjects$$

$$Student \bowtie_{Student.\,Std\,=\,Subjects.\,Class} Subjects$$

SID	Name	Std	Class	Subject
101	Rohan	11	11	Maths
101	Rohan	11	11	Physics
102	Mira	12	12	English
102	Mira	12	12	Chemist

Outer Join

*An **Inner join** includes only those tuples with matching attributes and the rest are discarded in the resulting relation. Therefore, we need to use **outer joins** to include all the rest of the tuple from the participating relations in the resulting relation.*

The **outer join** operation is an **extension of the join operation** that **avoids loss of information.**

Outer Join contains matching tuples that satisfy the matching condition, along with some or all tuples that do not satisfy the matching condition.

- It is based on both matched or unmatched tuple.
- It contains all rows from either one or both relations are present

*It uses **NULL** values.*

- *NULL* signifies that the value is unknown or does not exist

Inner Join(Natural Join)

Courses

CID	Course
100	Database
101	Mechanics
102	Electronics

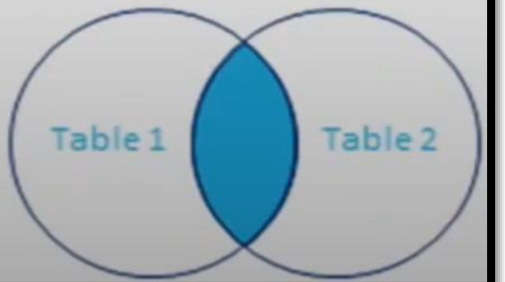
HOD

CID	Name
100	Rohan
102	Sara
104	Jiya

Courses $\bowtie$ HOD

CID	Course	Name
100	Database	Rohan
102	Electronics	Sara

Inner Join



Outer Join

- **Outer Join = Natural Join + Extra information**

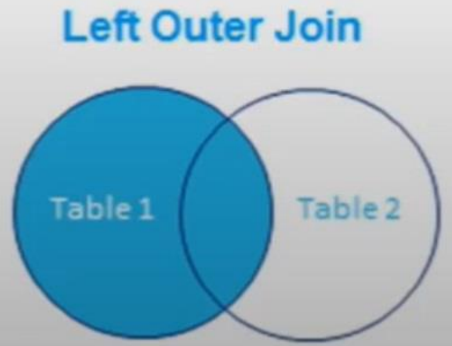
  (from left table, right table or both table)

- There are **three kinds of outer joins:**
  1. **Left outer join**
  2. **Right outer join**
  3. **Full outer join**

Left Outer Join

When applying **join** on two relations R1 and R2, some tuples of R1 or R2 does not appear in result set which does not satisfy the join conditions. **But..**

- In **Left outer join**, ***all the tuples from the Left relation R1 are included in the resulting relation.*** The tuples of R1 which do not satisfy join condition will have values as **NULL** for attributes of R2.
- In short:
  - **All** record from **left** table
  - Only matching records from right table
- **Symbol**: ⟕
- **Notation**: **R1 ⟕ R2**



Left Outer Join: Example

Courses

CID	Course
100	Database
101	Mechanics
102	Electronics

HOD

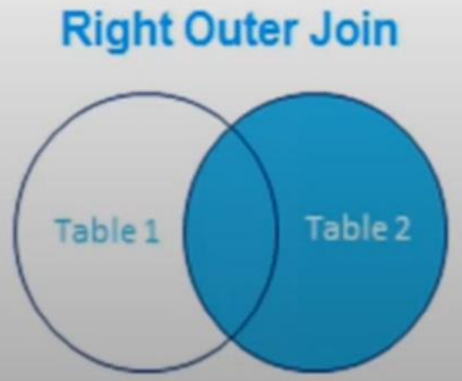
CID	Name
100	Rohan
102	Sara
104	Jiya

Courses ⟕ HOD

CID	Course	Name
100	Database	Rohan
101	Mechanics	**NULL**
102	Electronics	Sara

Right Outer Join

- In **Right outer join**, ***all the tuples from the right relation R2 are included in the resulting relation.*** The tuples of R2 which do not satisfy join condition will have values as **NULL** for attributes of R1.
- In short:
  - **All** record from **right** table
  - Only matching records from left table
- **Symbol**: ⟖
- **Notation**: **R1 ⟖ R2**



Right Outer Join: Example

Courses

CID	Course
100	Database
101	Mechanics
102	Electronics

HOD

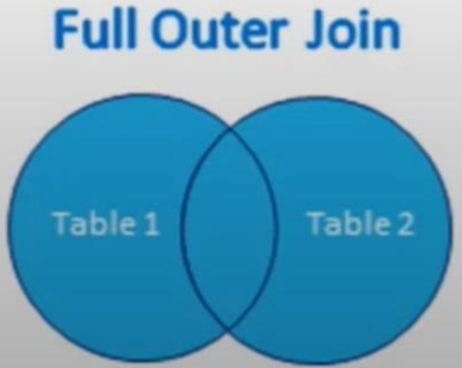
CID	Name
100	Rohan
102	Sara
104	Jiya

Courses ⟖ HOD

CID	Course	Name
100	Database	Rohan
102	Electronics	Sara
104	**NULL**	Jiya

Full Outer Join

- In **Full outer join**, ***all the tuples from both Left relation R1and right relation R2 are included in the resulting relation.*** The tuples of both relations R1 and R2 which do not satisfy join condition, their respective unmatched attributes are made **NULL**.
- In short:
  - All record from **all** table
- **Symbol**: ⟗
- **Notation**: **R1 ⟗ R2**



Example: Full Outer Join

Courses

CID	Course
100	Database
101	Mechanics
102	Electronics

HOD

CID	Name
100	Rohan
102	Sara
104	Jiya

Courses ⟗ HOD

CID	Course	Name
100	Database	Rohan
101	Mechanics	**NULL**
102	Electronics	Sara
104	**NULL**	Jiya

Additional Relational Operations

Extended Relational Algebra ***increases power over basic relational algebra.***

1. Generalized Projection
2. Aggregate Functions
3. Outer Join

Generalized Projection

- Normal ***projection*** only projects the columns where as ***generalized projection*** allows arithmetic operations on those projected columns.
- **Generalized Projection** extends the ***projection operation*** by allowing ***arithmetic functions*** to be used in the ***projection list***.

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

  - ***E*** is any relational-algebra **expression**
  - Each of $F_1, F_2, \ldots, F_n$ are **arithmetic expressions** involving ***constants and attributes*** in the schema of *E*.

Example 1

r

A	B	C
a	1	5
a	2	5
b	3	8
b	4	10

$\Pi_{A,\ C-B}\ (r)$

A	C - B
a	4
a	3
b	5
b	6

Example 2

Given relation:

credit_info(customer_name, limit, credit_balance)

customer_name	*limit*	*credit_balance*
A	5000	2000
B	6000	4000
C	10000	6000

"Find how much more each person can spend ?"

$$\Pi_{customer\_name,\ limit\ -\ credit\_balance}\ (credit\_info)$$

customer_name	*Limit - credit_balance*
A	3000
B	2000
C	4000

Aggregate functions and operations

- **Aggregation function** *takes **a collection of values** and **returns a single value** as **a result**.*

  avg: **average value**
  min: **minimum value**
  max: **maximum value**
  sum: **sum of values**
  count: **number of values**

  - These operations can be applied on **entire relation** or **certain groups of tuples**.
  - It ignore **NULL** values except **count**

- *Generalize form* ($\mathcal{g}$) of **Aggregate operation**:

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

  - 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**
  - Each A_i is an **attribute name**

Aggregate Operation: Example 1

Relation **r**

A	B	C
a	a	5
a	b	5
b	c	2
b	d	3

$\mathcal{G}_{sum(c)}(r)$

F A

sum (c)
20

${}_{A}\mathcal{G}_{sum(c)}(r)$

A	sum (c)
a	10
b	5

Aggregate Operation: Example 2

Relation '*account*' grouped by *branch-name*:

branch_name	*account_number*	*balance*
Perryridge	A-102	400
Perryridge	A-201	900
Brighton	A-217	750
Brighton	A-215	750
Redwood	A-222	700

branch_name $\mathcal{G}$ sum(balance) (*account*)

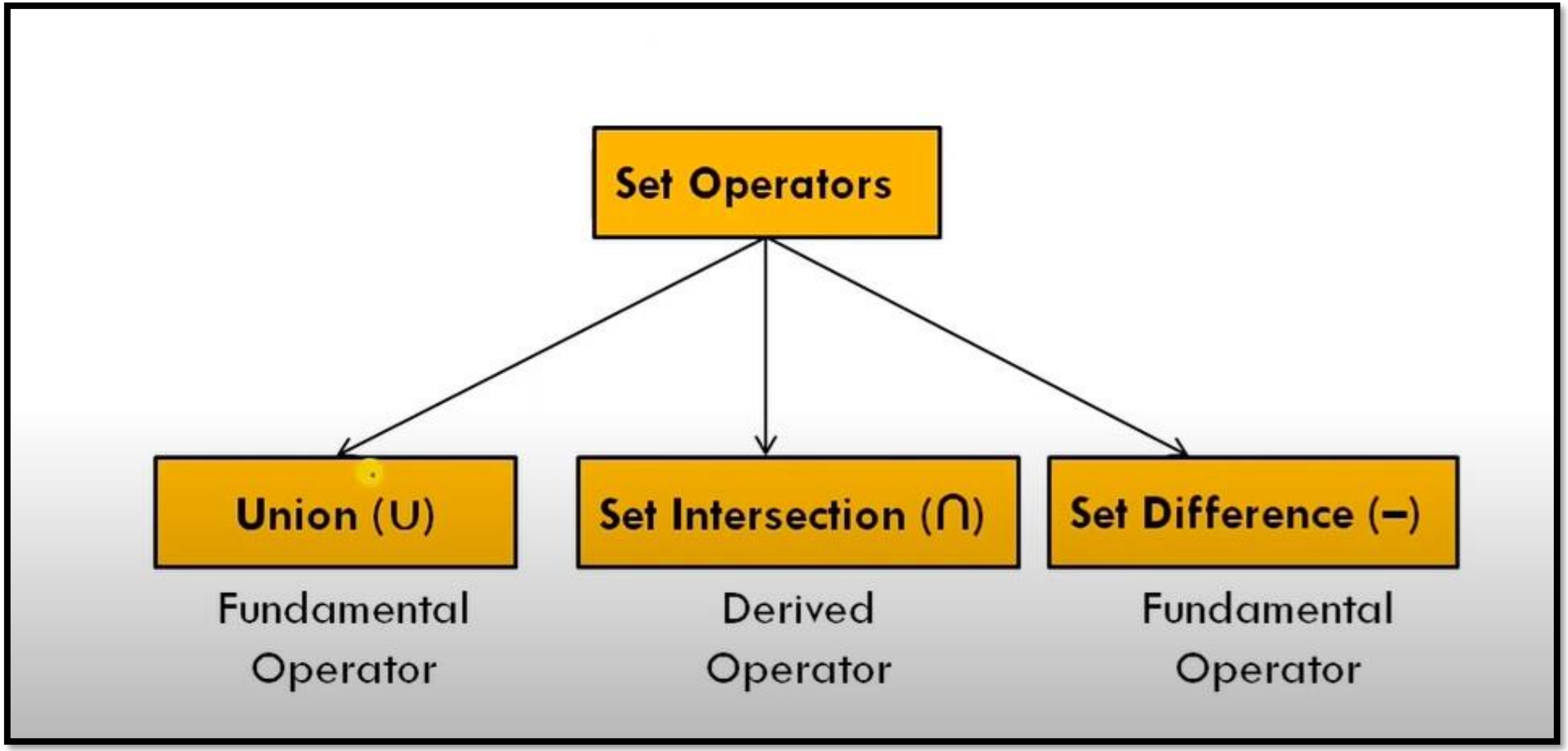
branch_name	*sum(balance)*
Perryridge	1300
Brighton	1500
Redwood	700

Aggregate Functions..

- **Result of aggregation** ***does not have a name***
  - Can use **rename** operation to give it a name
  - For convenience, we permit renaming as part of aggregate operation using **'as'** keyword

$$_{branch\_name}\,\mathcal{G}_{\,sum(balance)\ as\ sum\_balance}\,(account)$$

Set operators in relational algebra



Set Operators

- **Set operators:** Union, intersection and difference, binary operators as they takes two input relations
- **To use set operators on two relations,**
  - The two relations must be **Compatible**
- **Two relations are Compatible** if -
  1. Both the relations must have **same number of attributes (or columns).**
  2. Corresponding **attribute (or column)** have the **same domain (or type).**
- Duplicate tuples are automatically eliminated

Union operator

Suppose R and S are two relations. ***The Union operation selects all the tuples that are either in relations R or S or in both relations R & S.***

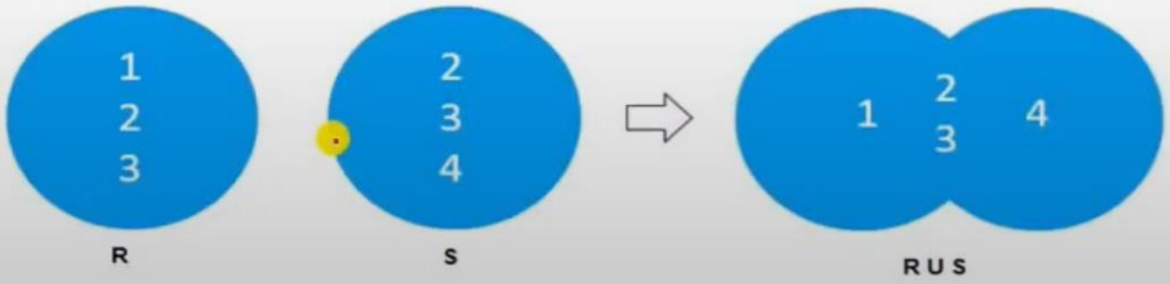
It ***eliminates*** *the* ***duplicate tuples.***

For a **union operation** to be **valid**, the following conditions must hold -

1. Two relations R and S both have **same number of attributes.**
2. Corresponding **attribute (or column)** have the **same domain (or type)**..
   - The attributes of R and S must occur in the same order.
3. Duplicate tuples should be automatically removed

Symbol: U

Notation: R U S



- RA: **R U S**
- SQL: **(SELECT * FROM R)**

  UNION

  (SELECT * FROM S);

Example 1

Student

Roll_no	Name
1	A
2	B
3	C
4	D

Employee

Emp_no	Name
2	B
8	G
9	H

(Student) U (Employee)

Roll_no	Name
1	A
2	B
3	C
4	D
8	G
9	H

Note: Union is commutative: A U B = B U A

Example 2, 3

Student

Roll_no	Name
1	A
2	B
3	C
4	D

Employee

Emp_no	Name
2	B
8	G
9	H

Π_{Name} (Student) U Π_{Name} (Employee)

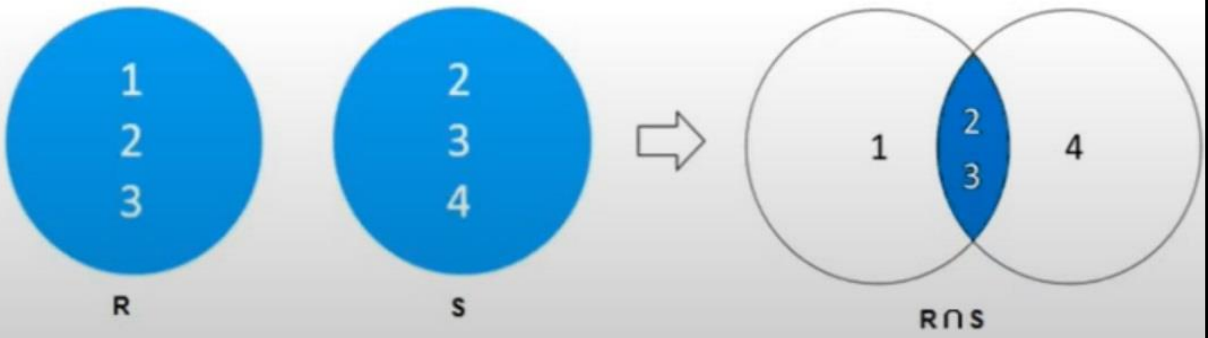
Name
A
B
C
D
G
H

- Find the names of the authors who have either written a book or an article or both.

$$\Pi_{author}(\text{Books}) \cup \Pi_{author}(\text{Articles})$$

Set Intersection operator

- Suppose R and S are two relations. ***The Set Intersection operation selects all the tuples that are in both relations R & S.***
- For a **Set Intersection** to be **valid**, the following conditions must hold –
  1. Two relations R and S both have **same number of attributes.**
  2. Corresponding **attribute (or column)** have the **same domain (or type)**.
     - The attributes of R and S must occur in the same order.
- **Symbol:** ∩
- **Syntax**: R ∩ S
  - RA: **R ∩ S**
  - SQL: (**SELECT * FROM R**)
    INTERSECT
    (**SELECT * FROM S**);



Example 1

Student

Roll_no	Name
1	A
2	B
3	C
4	D

Employee

Emp_no	Name
2	B
8	G
9	H

(Student) ∩ (Employee)

Roll_no	Name
2	B

Note: Set Intersection is commutative: A ∩ B = B ∩ A

Example 2, 3

Student

Roll_no	Name
1	A
2	B
3	C
4	D

Employee

Emp_no	Name
2	B
8	G
9	H

Π_{Name} **(Student)** $\cap$ Π_{Name} **(Employee)**

→

Name
B

- Find the names of the authors who have written a book and an article both.

$$\Pi_{author}(\text{Books}) \cap \Pi_{author}(\text{Articles})$$

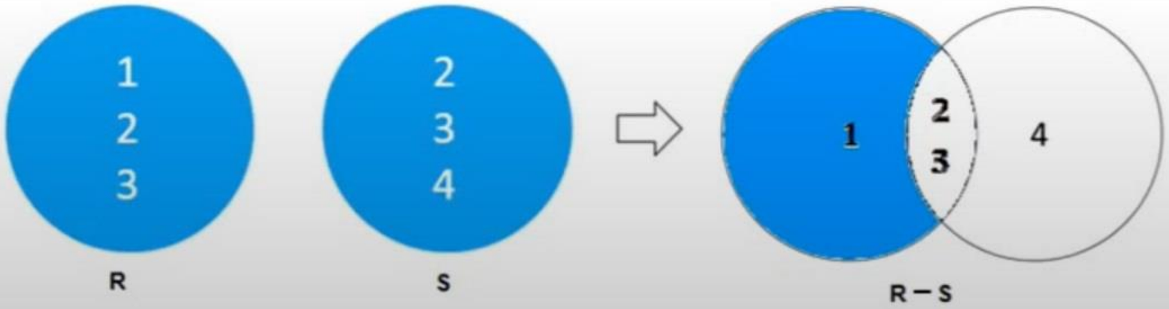
Set Difference

- Suppose R and S are two relations. ***The Set Difference operation selects all the tuples that are present in first relation R but not in second relation S.***
- For a **Set Difference** to be **valid**, the following conditions must hold -
  1. Two relations R and S both have **same number of attributes.**
  2. Corresponding **attribute (or column)** have the **same domain (or type).**
     - The attributes of R and S must occur in the same order.

- **Symbol:** **–**
- **Syntax:** **R – S**
  - RA: **R – S**
  - SQL: (**SELECT * FROM R**)

    EXCEPT

    (**SELECT * FROM S**);



Example 1

Student

Roll_no	Name
1	A
2	B
3	C
4	D

Employee

Emp_no	Name
2	B
8	G
9	H

(Student) – (Employee)

Roll_no	Name
1	A
3	C
4	D

Note: **1. Set Difference is non-commutative: $A - B \neq B - A$**

2. $R - (R - S) = R \cap S$

Intersection can be is derived from set difference that's why intersection is derived operator

Example 2, 3

Student

Roll_no	Name
1	A
2	B
3	C
4	D

Employee

Emp_no	Name
2	B
8	G
9	H

Π_{Name} (**Student**) – Π_{Name} (**Employee**)

→

Name
A
C
D

- Find the names of the authors who have written books but not articles.

$$\Pi_{author}\ (\text{Books}) - \Pi_{author}\ (\text{Articles})$$