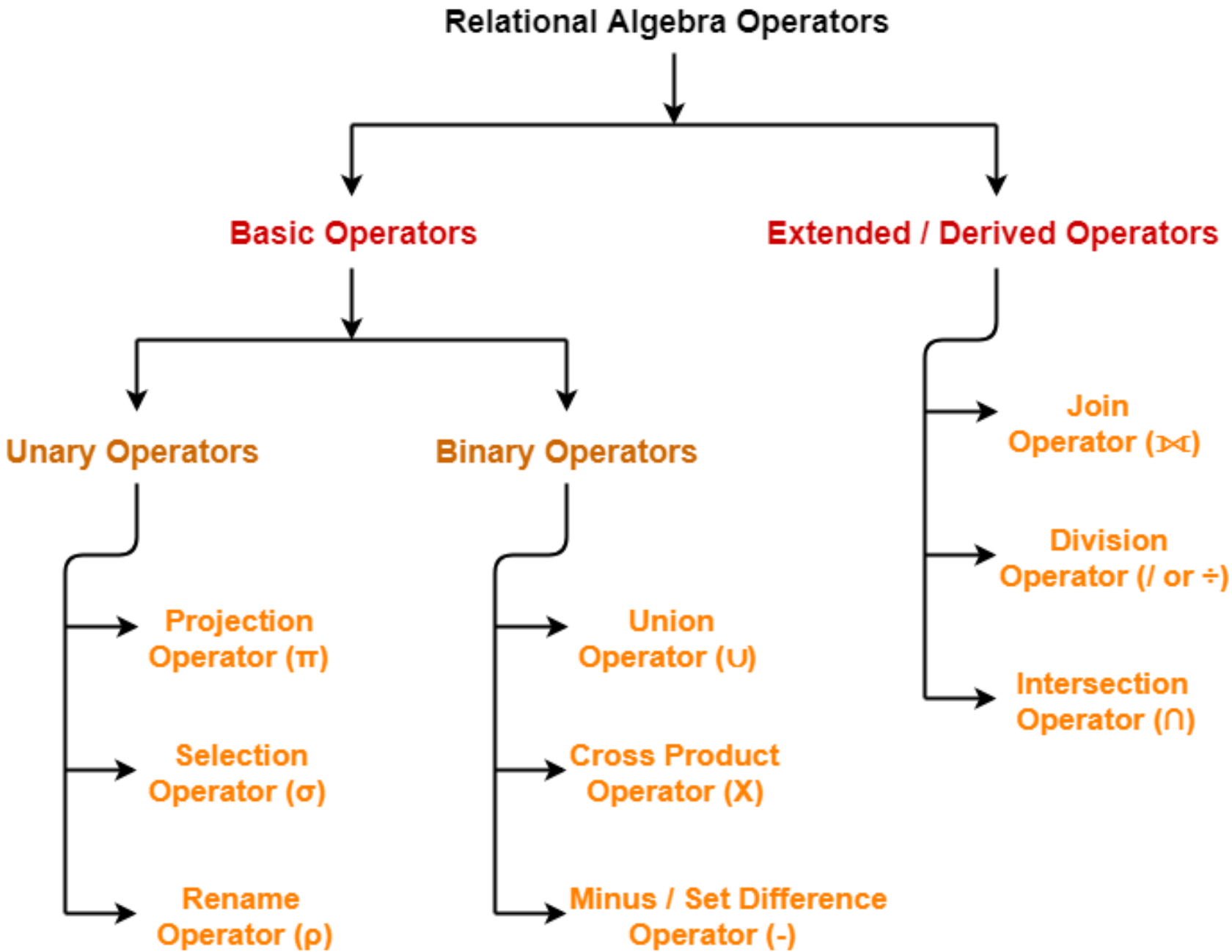
# CS500-Data Science Tools and Technique
# Relational Algebra

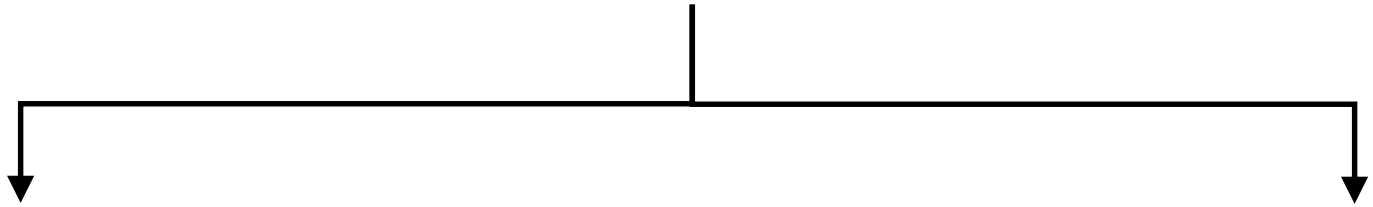
Bahar Ali
PhD Scholar,
National University Of Computer and Emerging Sciences,
Peshawar.

# Relational algebra

- The theory of Relational algebra has been introduced by Edgar F. Codd in 1970.
- It is a procedural query language or formal query language
- It is a collection of mathematical expression (Theoretical model)
- Structural query language (SQL) is based on relational algebra
- Uses unary or binary operators to perform queries.
- It takes instances of relations as input and yields instances of relations as output.


Relational Algebra Operators
Basic Operators
Extended / Derived Operators
Unary Operators
Binary Operators
Join Operator (⋈)
Division Operator (/ or ÷)
Intersection Operator (∩)
Projection Operator (π)
Selection Operator (σ)
Rename Operator (ρ)
Union Operator (∪)
Cross Product Operator (X)
Minus / Set Difference Operator (-)

**Relational Algebra Operations**
**(Can also be divided as)**



**Set Operations From Mathematical Set Theory**

- Set Union (U)
- Set Difference (-)
- Set Cartesian Product (U)
- Set Intersection (∩)

**Operations developed Specifically for Relational Databases**

- Selection (σ)
- Projection (π)
- Join (⋈)
- Division (÷)

# Selection (σ)

- The Selection operation is used for selecting a subset of the tuples according to a given selection condition.
- It is used as an expression to choose tuples which meet the selection condition.
- Denoted by Sigma(σ) Symbol

Customers

| CustomerID | CustomerName | Sales |
|---|---|---|
| 1 | Customer1 | 50050 |
| 2 | Customer2 | 40900 |
| 3 | Customer3 | 64050 |
| 4 | Customer4 | 6050 |

$\sigma_{sales > 50000}$ (Customers)

| CustomerID | CustomerName | Sales |
|---|---|---|
| 1 | Customer1 | 50050 |
| 3 | Customer3 | 64050 |

# Projection (Π)

- The projection method defines a relation that contains a vertical subset of Relation.
- This operator helps you to keep specific columns from a relation and discards the other columns.
- Denote by pi (Π) symbol.

Customers

| CustomerID | CustomerName | Status |
|---|---|---|
| 1 | Google | Active |
| 2 | Amazon | Active |
| 3 | Apple | Inactive |
| 4 | Alibaba | Active |

$\Pi_{\text{CustomerName, Status}}$ (Customers)

| CustomerName | Status |
|---|---|
| Google | Active |
| Amazon | Active |
| Apple | Inactive |
| Alibaba | Active |

# Rename (ρ)

- Rename is a unary operation used for renaming attributes of a relation.
- ρ (a/b)R will rename the attribute 'b' of relation by 'a‘
- Denoted by rho (ρ) symbol

**Employee**

| Name | EmployeeId |
|---|---|
| Harry | 3415 |
| Sally | 2241 |

**ρ** EmployeeName/Name **(Employee)**

| EmployeeName | EmployeeId |
|---|---|
| Harry | 3415 |
| Sally | 2241 |

# Set-Difference (-)

- Set-Difference is defined as a relation which includes all tuples that are in A but not in B. However, A and B must be union-compatible.
- For a union operation to be valid, the following conditions must hold:
  - A and B must be the same number of attributes.
  - Attribute domains need to be compatible.
  - Duplicate tuples should be automatically removed.

| Table A | |
|---|---|
| **column 1** | **column 2** |
| 1 | 1 |
| 1 | 2 |

| Table B | |
|---|---|
| **column 1** | **column 2** |
| 1 | 1 |
| 1 | 3 |

| Table A - B | |
|---|---|
| **column 1** | **column 2** |
| 1 | 2 |

# Union (∪):

- UNION includes all tuples that are in tables A or in B. It also eliminates duplicate tuples.
- For a union operation to be valid, the following conditions must hold:
  - A and B must have the same number of attributes.
  - Attribute domains need to be compatible.
  - Duplicate tuples should be automatically removed.

| Table A | |
|---|---|
| **column 1** | **column 2** |
| 1 | 1 |
| 1 | 2 |

| Table B | |
|---|---|
| **column 1** | **column 2** |
| 1 | 1 |
| 1 | 3 |

| Table A ∪ B | |
|---|---|
| **column 1** | **column 2** |
| 1 | 1 |
| 1 | 2 |
| 1 | 3 |

# Intersection (∩): **Derived Operator => A - (A - B)**

- Defines a relation consisting of a set of all tuple that are in both A and B. However, A and B must be union-compatible.
- For a union operation to be valid, the following conditions must hold:
  - A and B must be the same number of attributes.
  - Attribute domains need to be compatible.
  - Duplicate tuples should be automatically removed.

| **Table A** | | | **Table B** | |
|---|---|---|---|---|
| **column 1** | **column 2** | | **column 1** | **column 2** |
| 1 | 1 | | 1 | 1 |
| 1 | 2 | | 1 | 3 |

| **Table A ∩ B** | |
|---|---|
| **column 1** | **column 2** |
| 1 | 1 |

# Cross-Product (X)

- **Cartesian Product in DBMS** is an operation used to merge columns from two relations. Generally, a Cartesian product is never a meaningful operation when it performs alone. However, it becomes meaningful when it is followed by other operations. It is also called Cross Product or Cross Join.

| Table A | |
|---|---|
| **column A1** | **column A2** |
| 1 | 1 |
| 1 | 2 |

| Table B | |
|---|---|
| **column B1** | **column B2** |
| 1 | 1 |
| 1 | 3 |

| Table A X B | | | |
|---|---|---|---|
| **column A1** | **column A2** | **column B1** | **column B2** |
| 1 | 1 | 1 | 1 |
| 1 | 1 | 1 | 3 |
| 1 | 2 | 1 | 1 |
| 1 | 2 | 1 | 3 |

# Join Operations (⋈)

- Join operation is essentially a Cartesian product followed by a selection criterion
- JOIN operation also allows joining variously related tuples from different relations.
- **Types of JOIN:**
- Inner Joins: In an inner join, only those tuples that satisfy the matching criteria are included, while the rest are excluded. Let's study various types of Inner Joins:
  - Theta join
  - EQUI join
  - Natural join
- 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.
  - Left Outer Join
  - Right Outer Join
  - Full Outer Join

# Theta Join ($A \bowtie_\theta B$):

- The general case of JOIN operation is called a Theta join. It is denoted by symbol **θ**
- Theta join can use any conditions in the selection criteria.
- For example

| **Table A** | |
|---|---|
| **column 1** | **column 2** |
| 1 | 1 |
| 1 | 2 |

| **Table B** | |
|---|---|
| **column 1** | **column 2** |
| 1 | 1 |
| 1 | 3 |

| **$A \bowtie_{A.column\ 2 > B.column\ 2} (B)$** | |
|---|---|
| **column 1** | **column 2** |
| 1 | 2 |

# EQUI Join:

- When a theta join uses only equivalence condition, it becomes a EQUI join
- For example

| Table A | |
|---|---|
| **column 1** | **column 2** |
| 1 | 1 |
| 1 | 2 |

| Table B | |
|---|---|
| **column 1** | **column 2** |
| 1 | 1 |
| 1 | 3 |

| $A \bowtie_{A.column\ 2 = B.column\ 2} (B)$ | |
|---|---|
| **column 1** | **column 2** |
| 1 | 1 |

# 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.
- For example

| Table A | |
|---|---|
| **No** | **Name** |
| 1 | Ali |
| 2 | Khan |

| Table B | |
|---|---|
| **No** | **Contact#** |
| 1 | 0333 |
| 3 | 0345 |

| A ⋈ B | | |
|---|---|---|
| **No** | **Name** | **Contact#** |
| 1 | Ali | 0333 |

# Left Outer Join(A ⟕ B)

- In the left outer join, operation allows keeping all tuple in the 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.
- For example

| **Table A** | |
|---|---|
| **No** | **Name** |
| 1 | Ali |
| 2 | Khan |

| **Table B** | |
|---|---|
| **No** | **Contact#** |
| 1 | 0333 |
| 3 | 0345 |

| **A ⟕ B** | | |
|---|---|---|
| **No** | **Name** | **Contact#** |
| 1 | Ali | 0333 |
| 2 | Khan | - |

# Right Outer Join(A ⟖ B)

- In the right outer join, operation allows keeping all tuple in the right relation. However, if there is no matching tuple is found in the left relation, then the attributes of the left relation in the join result are filled with null values.
- For example

| Table A | |
|---|---|
| **No** | **Name** |
| 1 | Ali |
| 2 | Khan |

| Table B | |
|---|---|
| **No** | **Contact#** |
| 1 | 0333 |
| 3 | 0345 |

| A ⟖ B | | |
|---|---|---|
| **No** | **Name** | **Contact#** |
| 1 | Ali | 0333 |
| 3 | - | 0345 |

# Full Outer Join(A ⟗ B)

- In the left outer join, operation allows keeping all tuple in the 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.
- For example

| Table A | |
|---|---|
| **No** | **Name** |
| 1 | Ali |
| 2 | Khan |

| Table B | |
|---|---|
| **No** | **Contact#** |
| 1 | 0333 |
| 3 | 0345 |

| A ⟗ B | | |
|---|---|---|
| **No** | **Name** | **Contact#** |
| 1 | Ali | 0333 |
| 2 | Khan | - |
| 3 | - | 0345 |

# Division (÷)

- The relation returned by division operator will return those tuples from relation A which are associated to every B's tuple.
- For a division operation to be valid, attributes of B is proper subset of attributes of A.

**Table Course ÷ Table Student**

- Attributes in resulting relation = {Student#, Code} - {Student#} = Code
- Tuples in resulting relation = {Code} associated with all Student#'s tuple {1, 2}.

| **Table Student** | **Table Course** | |
|---|---|---|
| **Student#** | **Student#** | **Code** |
| 1 | 2 | C++ |
| 2 | 1 | Java |
| | 2 | Java |

| **A ÷ B** |
|---|
| **Code** |
| Java |