# UNIT -II

## RELATIONAL MODEL AND SQL

## . Schema

- A **schema** is the logical structure of the database.
- It defines how data is organized in tables: table names, attributes, their types, and relationships.

**Example**

```
Student(RollNo, Name, Dept, Age)
```

This describes the schema of the **Student** table.

---

## 2. Tuples

- A **tuple** is a single row/record in a table.
- Each tuple represents one instance of data.

**Example**

From the *Student* table:

```
(101, 'Arun', 'CSE', 20)
```

This is one tuple.

---

## 3. Domains

- A **domain** is the **allowable set of values** for an attribute.
- Each attribute has a domain.

**Examples**

- Age → integer between 1 and 120
- Dept → {CSE, ECE, MECH, IT}
- Name → string of letters

---

## 4. Integrity Rules

Integrity rules ensure **accuracy** and **consistency** of data in the database.

**Types of Integrity Rules**

*A. Domain Integrity*

- Values must come from the defined domain.
- Example: Age cannot be “abc”.

*B. Entity Integrity*

- Primary key cannot be **NULL**.
- Ensures each row is uniquely identified.

*C. Referential Integrity*

- Foreign key must match a primary key in another table or be NULL.

**Example**

If `DeptID` in *Student* refers to `DeptID` in *Department*, it must exist.

# Relational algebra: selection, projection, joins, set operations

Relational Algebra is a **procedural query language** used to manipulate and retrieve data stored in relational databases. It provides a set of operations that take one or two relations as input and produce a new relation as output. Important operations include **selection, projection, joins, and set operations**.

---

## 1. Selection (σ)

Selection is used to **choose rows/tuples** from a relation that satisfy a given condition.

**Definition**

- Denoted by **σ (sigma)**.
- It filters rows based on a **predicate (condition)**.
- Output is a subset of rows of the input relation.

**Example**

```
σ(Dept = 'CSE')(Student)
```

This returns all students whose department is CSE.

**Properties**

- Does not change the number of columns.
- Only reduces the number of rows.

---

## 2. Projection (π)

Projection is used to **choose specific columns/attributes** from a relation.

**Definition**

- Denoted by **π (pi)**.
- Eliminates unwanted columns and removes duplicate rows.

**Example**

```
π(Name, Dept)(Student)
```

This returns only the Name and Dept attributes of all students.

**Properties**

- Reduces number of columns.
- Removes duplicates automatically.

---

## 3. Joins

Join operations combine tuples from two relations based on a related attribute. Joins are fundamental for retrieving data that is spread across multiple tables.

**Types of Joins**

**(a) Theta Join (⋈θ)**

- Combines tuples using a **general condition** (θ).
- Example:

```
Student ⋈ Student.DeptID = Dept.DeptID Dept
```

**(b) Equi-Join**

- A special case of theta join where the condition uses **=** only.

```
Student ⋈ Student.DeptID = Dept.DeptID Dept
```

**(c) Natural Join (⋈)**

- A type of equi-join.
- Automatically matches common attributes **with the same name**.
- Removes duplicate columns.

**(d) Outer Joins**

Outer joins also include unmatched tuples by padding NULLs.

- **Left Outer Join (⟕)**
- **Right Outer Join (⟖)**
- **Full Outer Join (⟗)**

**Example**

```
Student ⟕ Dept
```

Returns all students even if they are not assigned to a department.

---

## 4. Set Operations

Relational Algebra includes several set-based operations because relations are sets of tuples.

**(a) Union (∪)**

Combines tuples from two relations, removing duplicates.
Relations must be **union-compatible** (same number and type of attributes).

Example:

```
Teacher ∪ GuestLecturer
```

**(b) Intersection (∩)**

Produces tuples common to both relations.

Example:

```
Student ∩ Scholar
```

**(c) Difference (−)**

Returns tuples present in the first relation but not in the second.

Example:

```
Student − Alumni
```

**(d) Cartesian Product (×)**

Combines every tuple of one relation with every tuple of another.

Example:

```
Student × Dept
```

Useful for defining joins.