📌 Definition of Functional Dependency (FD)

Functional Dependency in DBMS is a relationship between two sets of attributes in a relation (table), such that:

👉 **If two tuples (rows) have the same value for attribute(s) X, then they must also have the same value for attribute(s) Y.**

Formally:

- A functional dependency is denoted as **X → Y**.
- It means **X uniquely determines Y**.

🎓 Example Table: STUDENT

RollNo	Name	Branch	HostelRoom
101	Ramesh	CSE	A-12
102	Suresh	ECE	B-05
103	Priya	CSE	A-14
104	Ankit	ME	C-02

🔑 Functional Dependencies in this Table

1. **RollNo → Name, Branch, HostelRoom**
    - RollNo uniquely identifies each student.
    - Agar do rows ka RollNo same hai, toh unka Name, Branch, HostelRoom bhi same hoga.
    - RollNo is a **Primary Key** here.
2. **HostelRoom → RollNo, Name, Branch**
    - Agar HostelRoom unique assign hota hai, toh ek room se ek student hi linked hoga.
    - Means HostelRoom determines student details.
    - But agar ek room mein multiple students allowed ho, toh yeh FD valid nahi hogi.
3. **Branch → Name** ❌ (Not valid)
    - Ek branch mein multiple students ho sakte hain (CSE mein Ramesh aur Priya).
    - Isliye Branch se Name uniquely determine nahi hota.

🧠 Mnemonic to Remember

FD ka matlab: **Determinant → Dependent**

- Determinant attribute = jis se value decide hoti hai.
- Dependent attribute = jo us par depend karta hai.

👉 Think of it like: **Key → Value**

- RollNo (key) → Student details (value).

📌 Interview-Style Answer

> *"Functional Dependency means one attribute uniquely determines another. For example, in a STUDENT table, RollNo → Name, Branch, HostelRoom. This means RollNo uniquely identifies all other details of a student. FD is the foundation of normalization, ensuring data consistency and reducing redundancy."*

📌 Types of Functional Dependency (FD)

1. Trivial Functional Dependency

- Definition: Jab dependent attribute already determinant ka part ho.
- Form: **X** → **Y**, where **Y** ⊆ **X**.
- Example:
  - (RollNo, Name) → Name
  - Because Name is already part of determinant.

🏫 Example Table: STUDENT

RollNo	Name	Branch
101	Ramesh	CSE
102	Priya	ECE
103	Ankit	ME

📌 Trivial FD Explanation

- **Definition Recap**: Jab dependent attribute **already determinant ka part ho**, toh usse **Trivial FD** kehte hain. Form: **X → Y, where Y ⊆ X**

✅ Example in Table

- Determinant: (RollNo, Name)
- Dependent: Name

So FD: **(RollNo, Name) → Name**

👉 Why?

- Because **Name** is already included in the determinant set (RollNo, Name).
- Isliye ye dependency **always true** hogi, chahe table mein koi bhi data ho.

Another Example

- (RollNo, Branch) → Branch
- Here Branch is part of determinant, so it's also a **Trivial FD**.

🧠 Easy Way to Remember

Trivial FD = **Self-dependency**
👉 "A set of attributes always determines its own subset."

📌 Interview-Style Answer

> *"Trivial Functional Dependency occurs when the dependent attribute is already part of the determinant.*
>
> *For example, in a STUDENT table, (RollNo, Name) → Name is trivial because Name is already included in the determinant set."*

2. Non-Trivial Functional Dependency

- Definition: Jab dependent attribute determinant ka part na ho.
- Form: **X → Y**, where **Y ∉ X**.
- Example:
  - RollNo → Name
  - RollNo uniquely determines Name, but Name is not part of RollNo.

🧠 Non-Trivial FD with Overlap ka matlab kya hai?

- Jab **dependent attribute determinant ke andar nahi hota** → FD is **Non-Trivial**
- Agar **determinant aur dependent ke attribute sets mein kuch common ho** → matlab **overlap ho raha hai**

📚 Table: COURSE_ENROLLMENT

StudentID	StudentName	CourseID	CourseName	Instructor
S101	Ramesh	C01	DBMS	Dr. Sharma
S102	Priya	C02	CN	Prof. Mehta
S103	Ankit	C01	DBMS	Dr. Sharma

🔍 FD 1: `StudentID → StudentName`

- StudentID uniquely identifies StudentName.
- No overlap between determinant and dependent. ✅ **Completely Non-Trivial FD**

🔍 FD 2: `(StudentID, CourseID) → Instructor`

- Determinant = StudentID + CourseID
- Dependent = Instructor
- **CourseID** is present in determinant
- **Instructor** is contextually linked to CourseID (because each course has a fixed instructor) ✅ **Non-Trivial FD with overlap**

🧠 Explanation

- Jab tum `(StudentID, CourseID)` se `Instructor` nikaalte ho, toh ye FD **Non-Trivial** hai kyunki Instructor determinant ke andar nahi hai.
- Lekin **CourseID** dono sides mein logically linked hai — woh **overlap** create karta hai.
- Isliye ye FD ban jaata hai: **Non-Trivial FD with overlap**

3. Completely Non-Trivial FD

- Definition: Jab determinant aur dependent dono disjoint ho (no overlap).
- Example:
  - RollNo → Branch
  - RollNo and Branch have no common attribute.

4. Transitive Dependency

- Definition: Jab ek attribute indirectly depend karta hai key par.
- Form: If **X** → **Y** and **Y** → **Z**, then **X** → **Z**.
- Example:
  - RollNo → DeptID, DeptID → DeptName
  - So RollNo → DeptName (transitive).
- Important: Transitive dependencies ko **3NF** mein remove karte hain.

📘 Armstrong's Axioms: Rules of FD

1. ✅ Reflexive Rule

- **Rule:** Agar Y ⊆ X, toh X → Y hamesha true hoga.
- **Hinglish:** Agar dependent attribute already determinant ke andar hai, toh FD valid hai.
- **Example:**
  - (RollNo, Name) → Name
  - Because Name ⊆ (RollNo, Name)

2. ✅ Augmentation Rule

- **Rule:** Agar X → Y valid hai, toh XZ → YZ bhi valid hoga (Z = extra attribute).
- **Hinglish:** Agar ek FD true hai, toh usme extra attribute add karne se bhi FD valid rahegi.
- **Example:**
  - RollNo → Name
  - So (RollNo, Branch) → (Name, Branch)

3. ✅ Transitive Rule

- **Rule:** Agar X → Y aur Y → Z, toh X → Z bhi valid hoga.
- **Hinglish:** Agar ek attribute indirectly kisi aur par depend karta hai, toh FD transitively valid hai.
- **Example:**
    - RollNo → DeptID
    - DeptID → DeptName
    - So RollNo → DeptName

4. ✅ Union Rule

- If X → Y and X → Z, then X → YZ
- Example: RollNo → Name, RollNo → Branch ⇒ RollNo → (Name, Branch)

5. ✅ Decomposition Rule

- If X → YZ, then X → Y and X → Z
- Example: RollNo → (Name, Branch) ⇒ RollNo → Name and RollNo → Branch

📘 What is Normalization?

- **Normalization** = Database ko multiple logically related tables mein todna
- **Goal** = Remove redundancy, avoid anomalies, aur maintain data integrity
- **Invented by**: E.F. Codd (relational model ke creator)

🔍 Why Normalize?

Without normalization:

- Same data bar-bar repeat hota hai (redundancy)
- Update karne par kuch jagah change hota hai, kuch jagah nahi (inconsistency)
- Naye data insert karna mushkil ho jaata hai (insert anomaly)
- Delete karne par important data bhi chala jaata hai (delete anomaly)

❌ Unnormalized Table: EMPLOYEE_DEPARTMENT

EmpID	EmpName	DeptName	DeptLocation
E101	Ramesh	IT	Mumbai
E102	Priya	HR	Delhi
E103	Ankit	IT	Mumbai

1️⃣ Insertion Anomaly

- Suppose ek **new department** bana hai: "Finance" at "Pune".
- Problem: Jab tak koi employee us department mein assign na ho, hum **Finance dept ka record insert nahi kar sakte**.
- Matlab: Dept info insert karne ke liye employee info bhi dena padta hai → ❌ Insertion anomaly.

2️⃣ Deletion Anomaly

- Agar hum **E101 (Ramesh)** ko delete karte hain:
  - Uske saath IT dept ka record bhi delete ho jaata hai (DeptLocation = Mumbai).
- Matlab: Ek employee delete karne se **department info bhi chala jaata hai** → ❌ Deletion anomaly.

3️⃣ Update Anomaly

- Agar IT department ka location change ho jaaye (Mumbai → Pune):
  - Har row jisme IT dept hai, usme location update karna padega.
  - Agar ek row miss ho gayi, toh **inconsistency** aa jaayegi.
- Matlab: Same info multiple jagah update karna padta hai → ❌ Update anomaly.

✅ Normalized Tables (Fix)

EMPLOYEE Table

EmpID	EmpName	DeptName
E101	Ramesh	IT
E102	Priya	HR
E103	Ankit	IT

DEPARTMENT Table

DeptName	DeptLocation
IT	Mumbai
HR	Delhi

🔥 Benefits

- **Insertion**: Finance dept add kar sakte ho bina employee ke.
- **Deletion**: Employee delete karne se dept info safe rahega.
- **Update**: DeptLocation ek hi jagah update karna hoga.

📘 Concept of 1NF

- **Rule:** Har cell mein **atomic (single) value** honi chahiye.
- **No repeating groups / multi-valued attributes** allowed.
- Matlab: Table ke andar ek hi column mein multiple values nahi likh sakte.

❌ Example: Unnormalized Table (Not in 1NF)

StudentID	StudentName	Courses
101	Ramesh	DBMS, CN
102	Priya	OS, DBMS, Compiler
103	Ankit	CN

🔍 Problem

- **Courses column** mein ek student ke liye multiple subjects ek hi cell mein likhe hain.
- Ye **multi-valued attribute** hai → ❌ Not in 1NF.

✅ Normalized Table (In 1NF)

StudentID	StudentName	Course
101	Ramesh	DBMS
101	Ramesh	CN
102	Priya	OS
102	Priya	DBMS
102	Priya	Compiler
103	Ankit	CN

🔍 Fix

- Har row mein **sirf ek course** likha gaya hai.
- Ab har cell atomic hai → ✅ Table is in **1NF**.

📌 Hinglish Summary

> *"1NF ka matlab hai ki har cell mein ek hi value ho. Agar ek student ke multiple courses ek hi column mein likhe ho, toh wo 1NF violate karta hai. Normalization karke hum unko alag-alag rows mein tod dete hain."*

🔍 Difference Between Super Key and Primary Key

Feature	Super Key	Primary Key
Definition	Any set of attributes that uniquely identify a row	Minimal set of attributes that uniquely identify a row
Uniqueness	Ensures uniqueness ✅	Ensures uniqueness ✅
Minimality	Not necessarily minimal ❌	Must be minimal ✅
Null Allowed?	Not allowed in any part of key ❌	Not allowed ❌
Number per table	Multiple possible super keys ✅	Only one primary key ✅
Example	`{StudentID}`, `{StudentID, Name}`, `{StudentID, Email}`	`{StudentID}` (if it's sufficient alone)

🧠 Real-Life Analogy:

Soch ek student database hai:

- **Super Keys:**
  - `{RollNo}`
  - `{RollNo, Name}`
  - `{RollNo, Email}` Sab uniquely identify kar sakte hain — toh ye super keys hain.
- **Primary Key:**
  - Sirf `{RollNo}` agar wo alone sufficient hai → ye minimal hai → **primary key banega**.

🎯 Summary:

- **Har primary key ek super key hoti hai**, but **har super key primary key nahi hoti**.
- Primary key = minimal + unique
- Super key = unique but may have extra columns

🔍 Detailed Difference Between Super Key and Candidate Key

Feature	Super Key	Candidate Key
Definition	Any set of attributes that uniquely identify a row	Minimal set of attributes that uniquely identify a row
Uniqueness	Ensures uniqueness ✅	Ensures uniqueness ✅
Minimality	Not necessarily minimal ❌	Must be minimal ✅
Extra Attributes	Allowed (e.g., RollNo + Name + Email) ✅	Not allowed ❌
Number per table	Many possible super keys ✅	Multiple candidate keys possible ✅
Primary Key Relation	Primary key is chosen from super keys ✅	Primary key is chosen from candidate keys ✅
Example	`{RollNo}`, `{RollNo, Name}`, `{RollNo, Email}`	`{RollNo}`, `{Email}` (if both are minimal and unique)

🧠 Real-Life Analogy:

Soch ek student database hai:

- **Super Keys:**
  - `{RollNo}`
  - `{RollNo, Name}`
  - `{RollNo, Email}` Sab uniquely identify kar sakte hain → super keys hain.
- **Candidate Keys:**
  - `{RollNo}`
  - `{Email}` Ye minimal hain → candidate keys hain.
- **Primary Key:**
  - Tu in candidate keys me se ek choose karta hai → jaise `{RollNo}`.

🎯 Summary:

- **Super Key = Unique Identification (may be extra attributes)**
- **Candidate Key = Minimal Unique Identification (no extras)**
- **Primary Key = One chosen candidate key**

🔑 Relationship Between Super Key, Candidate Key, and Primary Key

1. Super Key

- Any attribute set jo ek row ko uniquely identify kar sake.
- Example: `{RollNo}`, `{RollNo, Name}`, `{RollNo, Email}` sab super keys hain.
- **Super set** hai → sabse bada group.

2. Candidate Key

- Minimal super key (no extra attributes).
- Example: `{RollNo}`, `{Email}` agar dono alone unique hain.
- **Subset of Super Key** → har candidate key ek super key hoti hai, but minimal.

3. Primary Key

- Ek candidate key jo database designer choose karta hai as main identifier.
- Example: `{RollNo}` ko primary key bana diya.
- **Subset of Candidate Key** → sirf ek select hota hai.

📊 Hierarchy Diagram (Text Form)

```
Super Key (largest set)
   ⬇
Candidate Key (minimal super keys)
   ⬇
Primary Key (chosen candidate key)
```

🎯 Summary

- **Super Key ⊃ Candidate Key ⊃ Primary Key**
- Har primary key ek candidate key hai.
- Har candidate key ek super key hai.
- Har super key primary key nahi hoti.

🔑 Composite Key Definition

- **Composite Key** = Jab ek single column se record uniquely identify nahi hota, toh **multiple columns ko combine karke ek unique identifier banaya jata hai**.
- Matlab: **2 ya zyada attributes milke ek primary key banate hain**.

📊 Example

Student_Project Table

StudentID	ProjectID	Marks
101	P01	85
101	P02	90
102	P01	88

- Yahan **StudentID** alone unique nahi hai (ek student multiple projects kar sakta hai).
- **ProjectID** alone bhi unique nahi hai (ek project multiple students ke saath ho sakta hai).
- Lekin **StudentID + ProjectID** milke har row ko uniquely identify karte hain → ye hi **Composite Key** hai.

🎯 Key Points

- Composite key = **Combination of 2+ attributes** to uniquely identify a record.
- Har attribute individually unique nahi hota, par saath mein unique ban jata hai.
- Composite key ko **Primary Key** banaya ja sakta hai agar wo minimal aur unique ho.

🔍 Composite Primary Key Case

- Composite Primary Key = jab **2 ya zyada attributes milke ek row ko uniquely identify karte hain**.
- Example: `(StudentID, CourseID)`

👉 **Rule:**

- **Dono attributes mein NULL allowed nahi hai.**
- Agar ek attribute NULL ho jaaye, toh composite key apna uniqueness lose kar dega.

❌ Violation Example

StudentID	CourseID	Marks
S101	C01	85
S102	NULL	90

- Yahan `(StudentID, CourseID)` composite primary key hai.
- Lekin ek row mein `CourseID = NULL` hai → ❌ violation.
- Kyunki ab hum uniquely identify nahi kar paayenge ki S102 kis course mein enrolled hai.

📌 Summary

> *"Composite Primary Key mein bhi har attribute NOT NULL hona chahiye. Agar ek bhi NULL ho jaaye, toh row uniquely identify nahi ho paayegi. Isliye composite key ke saare parts mandatory hote hain."*

1️⃣ Primary Key Example

Student Table

StudentID	StudentName	Email
101	Divya	divya@abc.com
102	Jacob	jacob@abc.com
103	Aria	aria@abc.com

- **Primary Key:** `StudentID`
  - Har student ko uniquely identify karta hai.
  - Not Null + Unique hona zaroori hai.

2️⃣ Foreign Key Example

📊 Example: Student & Project Tables

1. Student Table

StudentID (PK)	StudentName
101	Divya
102	Jacob
103	Aria

- Yahan **Primary Key (PK)** = `StudentID`
- Har student ko uniquely identify karta hai.

2. Project Table

ProjectID (PK)	ProjectName	StudentID (FK)
P01	IoT	101
P02	Cloud	102
P03	CAD	103

- Yahan **Primary Key (PK)** = ProjectID
- **Foreign Key (FK)** = StudentID → references Student.StudentID

🔑 How Foreign Key Works

- `Project.StudentID` foreign key hai jo **Student table ke StudentID** ko refer karta hai.
- Matlab: Project table mein jo bhi StudentID hoga, wo Student table mein exist karna **zaroori** hai.
- Agar tu Project table mein `StudentID = 999` dalne ki koshish karega (jo Student table mein nahi hai), toh DBMS error dega.

🎯 Summary

- **Primary Key** = Table ka unique identifier.
- **Foreign Key** = Dusre table ke primary key ko refer karta hai.
- Ye dono milke **referential integrity** maintain karte hain.

🟢 Easy Definition

Referential Integrity = Foreign Key hamesha ek valid Primary Key ko point kare.
Matlab: Child table ka data parent table ke data se match hona chahiye.

📊 Simple Example

Student Table (Parent)

StudentID (PK)	Name
101	Divya
102	Jacob

Project Table (Child)

ProjectID	ProjectName	StudentID (FK)
P01	IoT	101 ✅ (valid)
P02	Cloud	102 ✅ (valid)
P03	CAD	999 ❌ (invalid)

👉 Yahan `999` Student table mein exist hi nahi karta. Isliye ye **referential integrity violation** hai.

3️⃣ Alternate Key

- Candidate keys that are **not chosen** as primary key.
- **Example:** If `StudentID` is primary key, then `RollNo` and `Phone` are alternate keys.

📘 Concept of Unique Key

- **Unique Key** ensures ki column ke values **unique** honi chahiye (no duplicates).
- Difference from **Primary Key**:
  - Primary Key → **unique + NOT NULL**
  - Unique Key → **unique but NULL allowed (once)**

🗒️ Example Table: EMPLOYEE

EmpID	EmpName	Email	Phone
E101	Ramesh	ramesh@gmail.com	9876543210
E102	Priya	priya@gmail.com	9123456789
E103	Ankit	NULL	9988776655
E104	Neha	neha@gmail.com	9876543210 ❌

🔍 Explanation

- **Primary Key**: `EmpID` (unique + no NULL)
- **Unique Key**: `Email`
  - Har employee ka email unique hona chahiye.
  - Ek employee ke liye email NULL ho sakta hai (allowed).
- **Problem**: Phone number repeat ho gaya (9876543210 for Ramesh & Neha) → agar Phone ko Unique Key banate toh violation hota.

✅ Correct Design with Unique Key

EmpID	EmpName	Email	Phone
E101	Ramesh	ramesh@gmail.com	9876543210
E102	Priya	priya@gmail.com	9123456789
E103	Ankit	NULL	9988776655
E104	Neha	neha@gmail.com	9112233445

- Ab **Email** column unique hai (no duplicates, ek NULL allowed).
- **Phone** bhi unique banaya jaa sakta hai agar har employee ka alag number ho.

📌 Summary

> *"Unique Key ek column ko unique banata hai, lekin ek NULL value allow karta hai. Primary Key hamesha unique + not null hoti hai. Example: Employee table mein EmpID primary key hai, aur Email unique key hai — har email unique hoga, ek employee ke liye NULL bhi allowed hai."*

📘 Simple Definition

👉 **Partial Dependency** = Jab composite primary key ho (matlab key banane ke liye 2 ya zyada attributes chahiye), aur koi non-key attribute us composite key ke **sirf ek part** par depend kare.

❌ Table with Violation (Not in 2NF)

StudentID	CourseID	StudentName	CourseName
S101	C01	Ramesh	DBMS
S102	C02	Priya	CN
S103	C01	Ankit	DBMS

🔍 Primary Key = (StudentID, CourseID)

- **StudentName** depends only on StudentID.
- **CourseName** depends only on CourseID.
  👉 Ye dono **partial dependencies** hain → ❌ Not in 2NF.

✅ Correct Tables (After Removing Partial Dependency → In 2NF)

STUDENT Table

StudentID	StudentName
S101	Ramesh
S102	Priya
S103	Ankit

COURSE Table

CourseID	CourseName
C01	DBMS
C02	CN

ENROLLMENT Table

StudentID	CourseID
S101	C01
S102	C02
S103	C01

🎯 Summary

- **Violation:** Jab composite key ke ek part pe hi non-key attribute depend kare → Partial Dependency.
- **Fix:** Non-key attributes ko alag tables mein daal do (Student aur Course), aur linking ke liye ENROLLMENT table banao.
- **Result:** Redundancy kam ho jaati hai, aur table ab 2NF mein aa jaata hai.

📘 Simple Definition of 3NF

👉 **3NF ka rule:**

1. Table must be in **2NF**.
2. **No Transitive Dependency** → Matlab: Non-key attribute kisi aur non-key attribute par depend nahi karna chahiye.

- Har non-key attribute **directly** primary key par depend kare, **indirectly** nahi.

❌ Example: STUDENT Table (Not in 3NF)

StudentID	StudentName	DeptID	DeptName
S101	Ramesh	D01	CSE
S102	Priya	D02	ECE
S103	Ankit	D01	CSE

🔍 Analysis

- **Primary Key** = StudentID
- **StudentName** → depends directly on StudentID ✅
- **DeptID** → depends directly on StudentID ✅
- **DeptName** → depends on DeptID, not directly on StudentID ❌ (transitive dependency)

Explanation:

- **StudentID** → **DeptID** (direct dependency)
- **DeptID** → **DeptName** (direct dependency)
- Iska matlab hai: **StudentID** → **DeptName** indirectly (transitive dependency) ❌

👉 Isliye table **Not in 3NF**.

✅ Correct Tables (In 3NF)

STUDENT Table

StudentID	StudentName	DeptID
S101	Ramesh	D01
S102	Priya	D02
S103	Ankit	D01

DEPARTMENT Table

DeptID	DeptName
D01	CSE
D02	ECE

📘 BCNF Simple Definition

- **3NF** remove karta hai transitive dependency.
- **BCNF** aur strict hai: 👉 *Har functional dependency mein left side (determinant) ek superkey hona chahiye.*

BCNF (Boyce–Codd Normal Form) ek stricter version hai 3NF ka. Simple shabdon mein: har functional dependency (X → Y) ke liye, X ek superkey hona chahiye. Agar koi determinant superkey nahi hai, toh table BCNF violate karta hai.

❌ Example: Not in BCNF

Student	Subject	Teacher
Jhansi	Database	P. Naresh
Jhansi	C	K. Das
Subbu	Database	P. Naresh
Subbu	C	R. Prasad

📘 Pehle basics: Superkey kya hota hai?

- **Superkey** = Aisa attribute set jo table ke har row ko uniquely identify kar sake.
- Matlab: Agar kisi combination se ek row uniquely mil jaaye, toh wo superkey hai.
- Primary key hamesha ek superkey hoti hai, lekin superkey mein extra attributes bhi ho sakte hain.

🔍 Functional Dependencies (FDs)

1. **(Student, Subject) → Teacher**
    - Agar tumhe Student aur Subject dono pata hain, toh Teacher uniquely identify ho jaata hai.
    - Example: (Jhansi, Database) → Naresh
    - Isliye `(Student, Subject)` ek **superkey** hai. ✅
2. **(Student, Teacher) → Subject**
    - Agar tumhe Student aur Teacher dono pata hain, toh Subject uniquely identify ho jaata hai.
    - Example: (Jhansi, Das) → C
    - Isliye `(Student, Teacher)` bhi ek **superkey** hai. ✅
3. **Teacher → Subject**
    - Agar tumhe sirf Teacher pata hai, toh Subject uniquely identify ho jaata hai?
    - Example: Naresh → Database, Das → C
    - Lekin Teacher alone **poore table ke rows uniquely identify nahi kar sakta** (kyunki ek teacher multiple students ko padha sakta hai).
    - Matlab Teacher superkey nahi hai → ❌ BCNF violation.

⚠️ Why Violation?

- BCNF ka rule: *Har FD mein left side (determinant) ek superkey hona chahiye.*
- Yahan FD `Teacher → Subject` mein **Teacher superkey nahi hai**, isliye violation hota hai.

✅ Correct Design (In BCNF)

TEACHER_SUBJECT Table

Teacher	Subject
P. Naresh	Database
K. Das	C
R. Prasad	C

STUDENT_TEACHER Table

Student	Teacher
Jhansi	P. Naresh
Jhansi	K. Das
Subbu	P. Naresh
Subbu	R. Prasad

👉 Ab har FD ka left side ek superkey hai → ✅ BCNF achieved.

📘 Simple Definitions

- **Normalization**: Data ko tod-tod ke alag tables mein store karna → redundancy kam, integrity strong.
- **Denormalization**: Kuch tables ko merge karna ya redundant data dobara store karna → queries fast, joins kam.

👉 Matlab: *Normalization = clean design, Denormalization = performance boost with controlled redundancy.*

Denormalization ka matlab hai: intentionally redundancy add karna taaki queries fast ho jaayein aur joins kam ho. Ye normalization ka ulta nahi hai, balki ek **optimization technique** hai jo tab use hoti hai jab performance zyada important ho integrity se

- **Integrity** ka matlab hota hai **correctness + consistency**.
- Matlab: Jo data ya information hai, wo **sahi (accurate)** ho aur **har jagah ek jaisa (consistent)** ho.

❌ Normalized Design Example

STUDENT Table

StudentID	StudentName	DeptID
S101	Ramesh	D01
S102	Priya	D02

DEPARTMENT Table

DeptID	DeptName	DeptLocation
D01	CSE	Jaipur
D02	ECE	Delhi

- Agar query karni ho: “Ramesh ka department location kya hai?” → STUDENT + DEPARTMENT join karna padega.

✅ Denormalized Design Example

StudentID	StudentName	DeptName	DeptLocation
S101	Ramesh	CSE	Jaipur
S102	Priya	ECE	Delhi

- Ab query fast ho gayi (sirf ek table se answer mil jaata hai).
- Lekin redundancy aa gayi: DeptLocation “Jaipur” ya “Delhi” different student ke liye repeat ho sakta hai.

⚠️ Trade-offs

- **Pros:**
  - Queries fast ho jaati hain (joins kam).
  - Reporting/analytics mein helpful.
- **Cons:**
  - Redundancy badh jaati hai.
  - Update anomalies ho sakti hain (DeptLocation change karna ho toh multiple rows update karni padengi).
  - Storage thoda zyada lagta hai.

📌 Summary

“Denormalization ka matlab hai ki performance ke liye hum intentionally redundancy allow karte hain. Normalization mein data split hota hai aur joins zyada hote hain, denormalization mein data combine hota hai aur queries fast ho jaati hain. Downside ye hai ki redundancy aur anomalies badh jaati hain.”

📘 Join in DBMS

- **Definition:** A JOIN clause is used to combine rows from two or more tables based on a related column.
- **Use case:** Normalization ke baad data multiple tables mein split ho jaata hai. JOIN ke through hum unko ek saath laa kar meaningful result nikalte hain

Join ka matlab hai: do ya zyada tables ke rows ko ek common column ke basis par combine karna. Ye DBMS ka powerful feature hai jo normalized tables ko ek saath query karne deta hai.

📘 Clause (matlab: ek condition ya statement jo SQL query mein use hoti hai)

- Example: `WHERE clause` (filter condition), `JOIN clause` (tables combine karne ka rule).
- Clause = ek **rule/condition** jo query ko guide karta hai.

🧾 Example Tables

STUDENT Table

StudentID	StudentName	DeptID
S101	Ramesh	D01
S102	Priya	D02
S103	Ankit	D01

DEPARTMENT Table

DeptID	DeptName
D01	CSE
D02	ECE
D03	ME

🔑 Types of Joins with Example

1️⃣ INNER JOIN (sirf matching rows)

```
SELECT StudentName, DeptName
FROM STUDENT
INNER JOIN DEPARTMENT
ON STUDENT.DeptID = DEPARTMENT.DeptID;
```

Result:

StudentName	DeptName
Ramesh	CSE
Priya	ECE
Ankit	CSE

👉 Sirf wo rows jahan DeptID dono tables mein match hua.

📘 Outer Join Definition

👉 **Outer Join** = Aisa join jo **matching rows + non-matching rows** dono ko result mein laata hai.

- Agar match na ho toh missing values ke liye **NULL fill** hota hai.

🔑 Types of Outer Joins

1. **LEFT OUTER JOIN**
2. **RIGHT OUTER JOIN**
3. **FULL OUTER JOIN**

2️⃣ LEFT JOIN (Left table ke saare rows + Right table match)

```
SELECT CustName, Product
FROM CUSTOMER
LEFT JOIN ORDER
ON CUSTOMER.CustID = ORDER.CustID;
```

🧾 Example Tables

CUSTOMER Table

CustID	CustName
C01	Ramesh
C02	Priya
C03	Ankit

ORDER Table

OrderID	CustID	Product
O101	C01	Laptop
O102	C02	Mobile
O103	C04	Tablet

Result:

CustName	Product
Ramesh	Laptop
Priya	Mobile
Ankit	NULL

👉 CUSTOMER ke saare rows aaye.

- Ramesh aur Priya ke orders match ho gaye.
- Ankit ka order nahi hai → Product = NULL.

3️⃣ RIGHT JOIN (Right table ke saare rows + Left table match)

```
SELECT CustName, Product
FROM CUSTOMER
RIGHT JOIN ORDER
ON CUSTOMER.CustID = ORDER.CustID;
```

🧾 Example Tables

CUSTOMER Table

CustID	CustName
C01	Ramesh
C02	Priya
C03	Ankit

ORDER Table

OrderID	CustID	Product
O101	C01	Laptop
O102	C02	Mobile
O103	C04	Tablet

Result:

CustName	Product
Ramesh	Laptop
Priya	Mobile
NULL	Tablet

👉 ORDER ke saare rows aaye.

- O101 aur O102 ke customers match ho gaye.
- O103 ka CustID = C04 hai jo CUSTOMER table mein nahi hai → CustName = NULL.

🎯 Difference Clear

- **LEFT JOIN** → Left table ke saare rows preserve hote hain.
- **RIGHT JOIN** → Right table ke saare rows preserve hote hain.

4️⃣ FULL JOIN (dono side ke saare rows)

```
SELECT EmpName, ProjName
FROM EMPLOYEE
FULL JOIN PROJECT
ON EMPLOYEE.EmpID = PROJECT.EmpID;
```

🧾 Example 1: FULL JOIN

EMPLOYEE Table

EmpID	EmpName
E01	Ramesh
E02	Priya
E03	Ankit

PROJECT Table

ProjID	EmpID	ProjName
P101	E01	Database
P102	E02	Networking
P103	E04	AI

Result

EmpName	ProjName
Ramesh	Database
Priya	Networking
Ankit	NULL
NULL	AI

👉 **Explanation:**

- Dono tables ke saare rows aaye.
- Jahan match hai wo combine ho gaye.
- Jahan match nahi hai wahan

5️⃣ CROSS JOIN (cartesian product)

```
SELECT ProdName, SuppName
FROM PRODUCT
CROSS JOIN SUPPLIER;
```

🧾 Example Table: CROSS JOIN

PRODUCT Table

ProdID	ProdName
P01	Laptop
P02	Mobile

SUPPLIER Table

SuppID	SuppName
S01	Dell
S02	Samsung
S03	Apple

Result (Cartesian Product)

ProdName	SuppName
Laptop	Dell
Laptop	Samsung
Laptop	Apple
Mobile	Dell
Mobile	Samsung
Mobile	Apple

👉 **Explanation:**

- Har product har supplier ke saath pair ho gaya.
- Agar PRODUCT = 2 rows aur SUPPLIER = 3 rows → result = 2 × 3 = 6 rows.

📌 Summary

- **FULL JOIN** → Dono tables ke saare rows combine hote hain, match na ho toh NULL fill hota hai.
- **CROSS JOIN** → Cartesian product deta hai, har row left table ka har row right table ke saath pair hota hai.

6️⃣ NATURAL JOIN

- Automatically common column ke basis par join karta hai (same name + same datatype).
- Explicit condition dene ki zarurat nahi hoti.

STUDENT Table

StudentID	StudentName
S101	Ramesh
S102	Priya
S103	Ankit

ENROLLMENT Table

StudentID	CourseName
S101	DBMS
S102	CN
S104	OS

```sql
SELECT StudentName, CourseName
FROM STUDENT
NATURAL JOIN ENROLLMENT;
```

Result

StudentName	CourseName
Ramesh	DBMS
Priya	CN

👉 Common column = `StudentID`.

- S101 aur S102 match ho gaye → result mein aaye.
- S103 ka enrollment nahi hai, aur S104 ka student nahi hai → result mein nahi aaye.

`INNER JOIN` aur `NATURAL JOIN` dono hi common column pe kaam karte hain, toh difference kya hai?

📘 INNER JOIN

- **Definition:** Explicitly condition likhni padti hai (`ON clause`) jisme batana hota hai ki kaunse column pe join karna hai.
- **Flexibility:** Tum khud decide karte ho ki kaunse column match karna hai, chahe naam same ho ya alag.
- **Safe:** Ambiguity nahi hoti kyunki tum condition clearly specify karte ho.

📘 NATURAL JOIN

- **Definition:** Automatically join karta hai **same name + same datatype** wale columns pe.
- **No ON clause:** Condition likhne ki zarurat nahi hoti.
- **Risk:** Agar tables mein multiple same-name columns hain, toh sab pe join ho jaata hai → kabhi kabhi unwanted results milte hain.

⭐ Important DBMS Interview Questions with Detailed Answers

1. What is DBMS?

- **Answer:** DBMS (Database Management System) ek software hai jo data ko store, retrieve aur manage karta hai. Ye user aur database ke beech ek bridge ka kaam karta hai. Example: MySQL, Oracle, MongoDB

2. Difference between DBMS and RDBMS

- **Answer:**
  - DBMS: Data ko files ya unstructured form mein store karta hai. Example: XML, IMS.
  - RDBMS: Data ko *tables* mein store karta hai aur relations maintain karta hai. Example: MySQL, PostgreSQL.
  - RDBMS mein *constraints* aur *normalization* hoti hai jo DBMS mein nahi

📘 Transaction ka Matlab (DBMS mein)

👉 **Transaction = ek logical unit of work in database.** Matlab ek set of operations jo ek saath perform hote hain aur database ko ek valid state se dusre valid state me le jaate hain.

- Example: Bank transfer ₹500 from A → B
  - Step 1: A ke account se ₹500 debit
  - Step 2: B ke account me ₹500 credit 👉 Ye dono steps **milke ek transaction** banate hain. Agar ek fail ho jaaye toh pura rollback ho jaata hai.

🔑 ACID Properties with Real-Life Examples

1️⃣ Atomicity (All or Nothing)

- **Meaning:** Transaction ya to poora hoga ya bilkul nahi hoga.
- **Scenario:**
  - User A ne Flipkart pe order place kiya.
  - Payment debit ho gaya par order confirm nahi hua. 👉 Atomicity ensure karega ki agar order confirm fail ho jaaye toh payment bhi rollback ho jaaye.

2️⃣ Consistency (Valid Rules Follow)

- **Meaning:** Transaction ke baad database hamesha valid rules follow kare.
- **Scenario:**
  - User A ke account me ₹1000 hai.
  - Usne ₹500 transfer kiya. 👉 Consistency ensure karega ki balance kabhi negative nahi hoga aur final balance ₹500 hi hoga.

3️⃣ Isolation (No Interference)

- **Meaning:** Parallel transactions ek-dusre ko disturb nahi karte.
- **Scenario:**
  - User A ₹500 transfer kar raha hai User B ko.
  - Us time User C, User B ka balance check kar raha hai. 👉 Isolation ensure karega ki User C ko ya to **old balance** dikhe ya **new balance**, par beech ka "half updated" balance kabhi nahi.

4️⃣ Durability (Permanent Changes)

- **Meaning:** Ek baar transaction commit ho gaya toh wo permanent hai, chahe system crash ho jaaye.
- **Scenario:**
  - User A successfully ₹500 transfer karta hai User B ko.
  - Immediately server crash ho jaata hai. 👉 Durability ensure karega ki transfer record database logs me safe rahe aur crash ke baad bhi balance update ho.

🧾 Easy Real-Life Scenarios

- **Railway Ticket Booking:**
  - Atomicity → Seat reserve tabhi hogi jab payment success ho.
  - Consistency → Ek seat do log ko allocate nahi hogi.
  - Isolation → Parallel bookings clash nahi karengi.
  - Durability → Confirmed ticket crash ke baad bhi safe rahega.
- **Online Shopping:**
  - Atomicity → Payment deduct tabhi hoga jab order confirm ho.
  - Consistency → Stock count sahi update hoga.
  - Isolation → Agar 10 log ek product kharid rahe hain, toh stock galat nahi dikhega.
  - Durability → Order record crash ke baad bhi safe rahega.

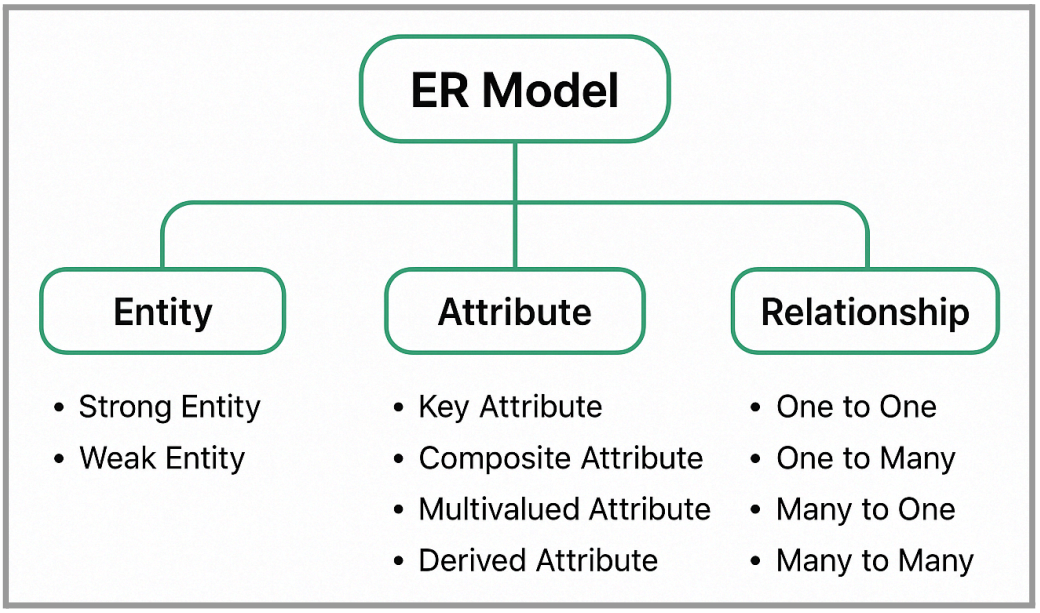
📌 Summary

"Transaction ek logical unit of work hai (jaise money transfer). ACID properties ensure karte hain ki transaction reliable ho: Atomicity = all or nothing, Consistency = rules follow, Isolation = parallel transactions disturb nahi karte, Durability = commit hone ke baad permanent save."

📘 Definition

👉 **E-R Model** ek conceptual model hai jo database design ke liye use hota hai.

- Ye **entities** (real-world objects), **attributes** (properties), aur **relationships** (connections) ko represent karta hai.
- Isse hum database ka logical structure easily visualize kar sakte hain.



📘 Entity Kya Hota Hai? Or Uske Types ?

👉 **Entity** = koi bhi real-world object ya concept jisko database mein store karna hai.

- Har entity ke paas kuch **attributes** hote hain (jaise Student ke paas Name, Age, ID).
- Entity ko **rectangle** se represent karte hain ER diagram mein.

🔹 1. Strong Entity

📌 Definition:

- Apne aap uniquely identify ho sakta hai.
- Iske paas **Primary Key** hoti hai.
- Kisi aur entity pe dependent nahi hota.

🗒️ Example:

STUDENT

StudentID (PK)	Name	Age
S101	Ramesh	20

- Yahan `StudentID` primary key hai → STUDENT apne aap identify ho sakta hai.
- STUDENT = strong entity.

🔹 2. Weak Entity — Simple Definition

👉 **Weak Entity** = Aisi entity jo apne aap uniquely identify nahi ho sakti.

- Iske paas **apni primary key nahi hoti**.
- Ye kisi **Strong Entity** pe dependent hoti hai.
- Isko identify karne ke liye **foreign key + partial key** ka combination use hota hai.
- ER diagram mein isko **double rectangle** se dikhate hain.

📘 Definition

👉 **Partial Key** = Wo attribute jo **Weak Entity** ko uniquely identify karta hai **sirf apne Strong Entity ke context mein**.

- Matlab: Apne aap se unique nahi hota.
- Lekin jab usko Strong Entity ke foreign key ke saath combine karte hain, tab unique ban jaata hai.

🗒️ Example: Employee & Dependent

✅ Strong Entity: EMPLOYEE

EmployeeID (PK)	Name
E101	Ramesh
E102	Priya

🔸 Weak Entity: DEPENDENT

DependentName (Partial Key)	Age	Relation	EmployeeID (FK)
Riya	12	Daughter	E101
Riya	10	Sister	E102

❓ Problem

- Yahan **DependentName = Riya** do baar aa raha hai.
- Agar sirf `DependentName` use kare toh unique nahi hai.

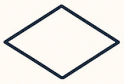
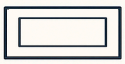
✅ Solution

- Combine karo: **EmployeeID + DependentName**
- E101 + Riya → unique (Ramesh ki daughter)
- E102 + Riya → unique (Priya ki sister)

👉 Yahan **DependentName** = **Partial Key** hai, kyunki wo apne aap unique nahi hai, par Strong Entity ke saath milke unique ho jaata hai.

📊 Summary

“Partial Key ek Weak Entity ka attribute hota hai jo apne aap unique nahi hota, par Strong Entity ke foreign key ke saath milke us Weak Entity ko uniquely identify karta hai. Example: DependentName apne aap unique nahi hai, par EmployeeID ke saath milke unique ban jaata hai.”

Figure	Symbol Name	Represents in ER Model
	**Entity**	Real-world object jisko database mein store karte hain (jaise STUDENT, COURSE)
	**Attribute**	Entity ki property ya feature (jaise Name, Age)
	**Relationship**	Entities ke beech ka connection (jaise teaches, enrolls)
	Connector	Shapes ko connect karta hai diagram mein
	**Multivalued Attribute**	Aise attribute jisme ek se zyada values ho sakti hain (jaise PhoneNumbers)
	**Weak Entity**	Aisi entity jo apne aap identify nahi ho sakti. kisi strong entity pe dependent hoti hai

📘 What is an Attribute?

👉 **Attribute** = Entity ki property ya feature hoti hai.

- Jaise agar entity hai **STUDENT**, toh uske attributes ho sakte hain: Name, Age, RollNo, Email.
- ER diagram mein attribute ko **ellipse (◯)** se dikhate hain.

🔹 Types of Attributes (with Examples)

1️⃣ Simple Attribute

- **Definition:** Aise attribute jo further divide nahi ho sakte.
- **Example:** Age, Gender, Salary
- **Hinglish:** “Seedha ek value hoti hai, aur aur parts mein nahi toot sakta.”

2️⃣ Composite Attribute

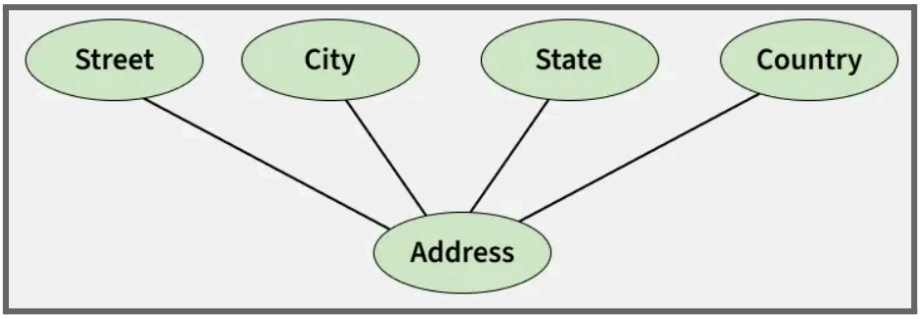
👉 Ye aisa attribute hota hai jo **multiple parts** mein toot sakta hai.

- **Example:** Address → Street, City, State, Country
- **ER Diagram Symbol:** Ek ellipse ke bahar **multiple ellipses** point karein — har sub-part ko dikhane ke liye.

🗣️ *"Address ek composite attribute hai kyunki uske andar chhoti-chhoti details hoti hain — jaise Street, City."*

🔹 Structure:

- Central ellipse → e.g., `Address`
- Connected ellipses → `Street`, `City`, `State`, `Country`
- Lines connect each sub-attribute to the main attribute

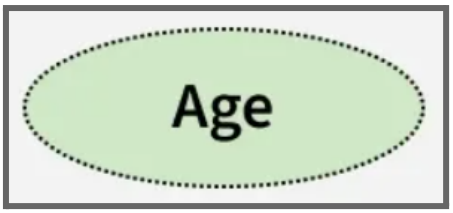


3️⃣ Derived Attribute

👉 Ye aisa attribute hota hai jo **kisi aur attribute se calculate** kiya jaata hai.

- **Example:** Age → calculated from DOB
- **ER Diagram Symbol: Dashed Ellipse** (◯ with dashed border)

🗣️ *"Age directly store nahi hota — DOB se nikalte hain, isliye ye derived attribute hai."*



4️⃣ Multivalued Attribute

👉 Ye aisa attribute hota hai jisme ek entity ke liye **ek se zyada values** ho sakti hain.

- **Example:** Phone_No (ek student ke paas 2 ya 3 numbers ho sakte hain)
- **ER Diagram Symbol: Double Ellipse**

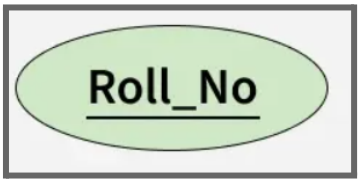
🗣️ *“Phone_No multivalued hai kyunki ek student ke paas ek se zyada numbers ho sakte hain.”*

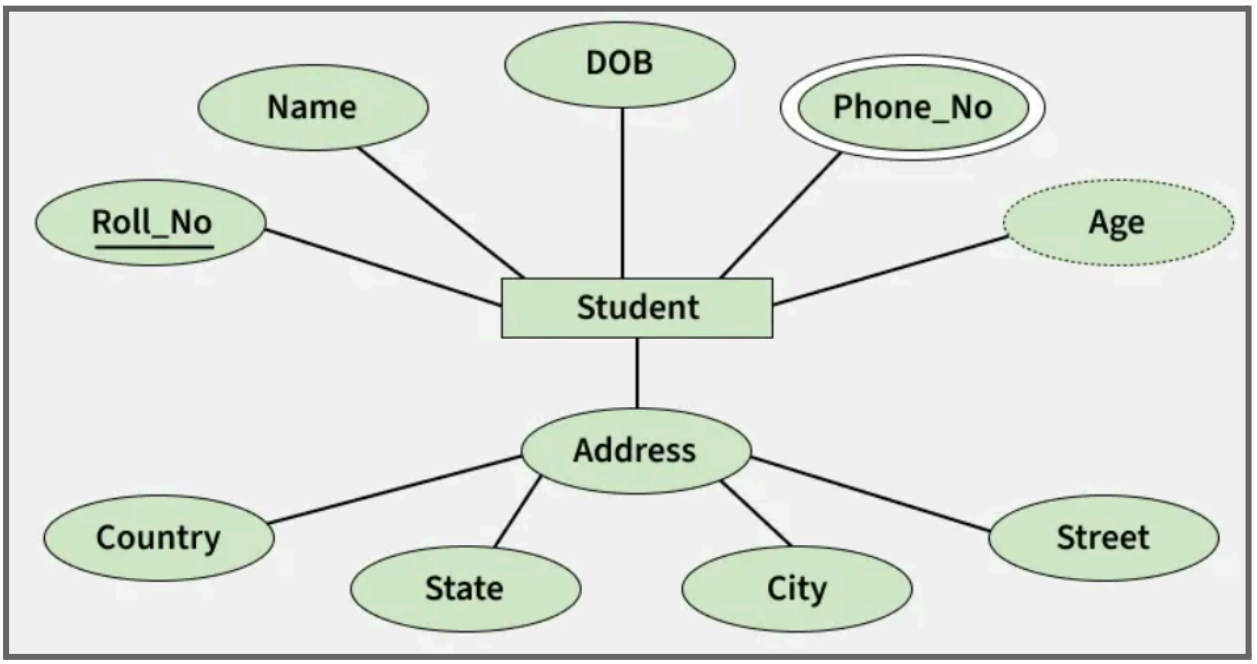
5️⃣ Key Attribute

👉 Ye wo attribute hota hai jo har entity ko **uniquely identify** karta hai.

- **Example:** Roll_No har student ke liye alag hoga.
- **ER Diagram Symbol:** Ellipse (◯) with **underline**.

🗣️ *“Roll_No jaise attribute se har student ki pehchaan hoti hai — ye unique hota hai.”*





📘 ER Diagram Explanation: STUDENT Entity

🔹 1. Entity: STUDENT

- Represented by a **rectangle**
- Real-world object jiska data store karna hai
- Example: Each student in a college database

🔹 2. Key Attribute: Roll_No

- Uniquely identifies each student
- Represented by an **underlined ellipse**
- Example: 101, 102, etc.

🔹 3. Simple Attributes

- **Name**, **DOB** → Represented by **normal ellipses**
- These are atomic values, not further divisible
- Example: Name = Mohit, DOB = 01/01/2004

🔹 4. Derived Attribute: Age

- Calculated from DOB
- Represented by a **dashed ellipse**
- Not stored directly, but derived when needed
- Example: Age = Current Year − Birth Year

🔹 5. Multivalued Attribute: Phone_No

- A student can have multiple phone numbers
- Represented by a **double ellipse**
- Example: 9876543210, 9123456789

🔹 6. Composite Attribute: Address

- Made up of sub-parts: Street, City, State, Country
- Represented by an **ellipse labeled 'Address'** connected to sub-ellipses
- Example:
    - Street = 12 MG Road
    - City = Jaipur
    - State = Rajasthan
    - Country = India

📘 What is an Entity Set?

🔹 Definition:

Entity Set is a collection of similar entities that share the same attributes.

🔹 Hinglish:

"Entity Set matlab ek jaise objects ka group — jinke attributes same hote hain. Jaise STUDENT entity set mein sab students honge, aur sabke paas Name, Roll_No, Age jaise attributes honge."

🎓 Example: STUDENT Entity Set

Roll_No	Name	Age	Phone_No	DOB	Address
101	Mohit	21	9876543210	01/01/2004	Jaipur, Rajasthan, India
102	Riya	20	9123456789	05/06/2005	Delhi, India

- Yahan **Mohit** aur **Riya** dono STUDENT entity set ke members hain.
- Dono ke paas same type ke attributes hain — bas values alag hain.

🎯 Summary

*"Entity Set ek group hota hai ek jaise entities ka — jaise STUDENT entity set mein sab students honge. Har student ke paas same type ke attributes hote hain. ER diagram mein rectangle se dikhate hain, aur agar strong ho toh **key attribute** underline hota hai."*

.

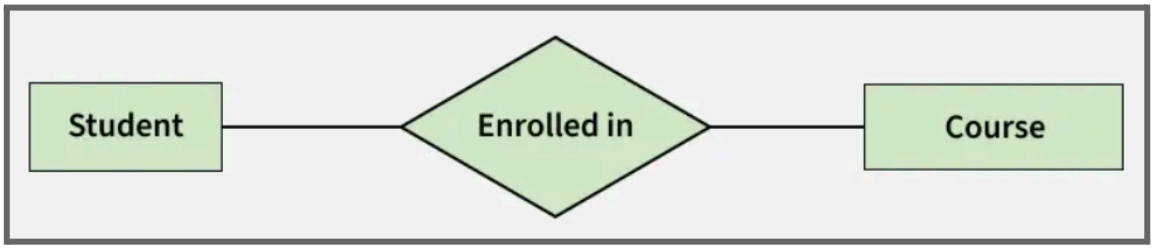
📘 1. Relationship Type vs Relationship Set

🔹 Relationship Type

"Ye batata hai ki kaun entity types aapas mein connected hain."

- Example: `Enrolled_In` is a relationship type between `Student` and `Course`
- ER Diagram Symbol: **Diamond (◇)**
- Connects entities using **lines**

🧠 *Think: 'Teaches', 'Works_On', 'Owns' — ye sab relationship types hain.*

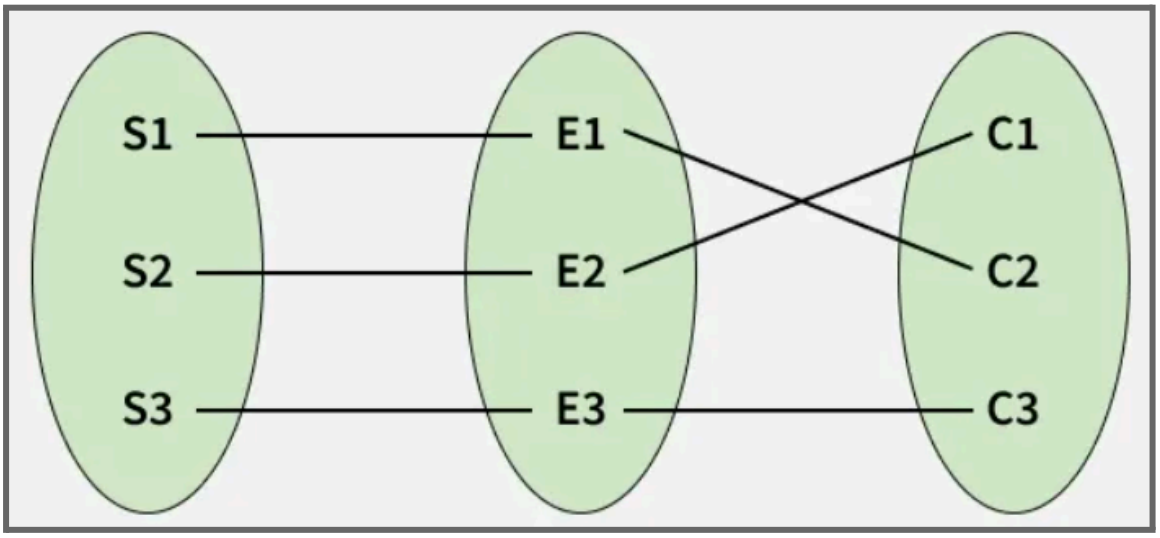


🔹 Relationship Set

"Ek hi type ke multiple relationships ka group."

- Example:
  - S1 enrolled in C2
  - S2 enrolled in C1
  - S3 enrolled in C3 → Ye sab `Enrolled_In` relationship set ke members hain

🧠 *Think: Table jisme Student-Course ke pairs likhe ho — wo relationship set hai.*



📘 Diagram Breakdown

🔹 Left Side: S1, S2, S3

- Ye sab **Student entities** hain
- Ye belong karte hain **STUDENT Entity Set** ko
- Har student ek unique instance hai

🔹 Middle: E1, E2, E3

- Ye represent karte hain **Enrollment relationships**
- Har E ek instance hai of the **Relationship Type: Enrolled_In**
- Diamond shape hota hai ER diagram mein — yahan abstracted as middle layer

🔹 Right Side: C1, C2, C3

- Ye sab **Course entities** hain
- Ye belong karte hain **COURSE Entity Set** ko
- Har course ek unique instance hai

🎯 Summary

> *"Diagram mein left side pe STUDENT entity set hai, right side pe COURSE entity set hai, aur beech mein ENROLLMENT relationship type hai. Har student ek course mein enrolled hai — in connections ka group banata hai Relationship Set."*

📊 Degree of Relationship Set — Table

Degree Type	No. of Entity Sets Involved	Description (Hinglish)
Unary / Recursive	1	Ek hi entity set apne aap se relate karta hai
Binary	2	Do alag entity sets ke beech relationship hoti hai
Ternary	3	3 entity sets ek hi relationship mein participate karte hain
N-ary	N (N > 3)	N entity sets ek relationship mein involved hote hain

📏 Cardinality in ER Model (Short Summary)

Cardinality batata hai ki ek entity kitni baar kisi relationship mein participate kar sakti hai.

🔹 Types of Cardinality:

Type	Meaning	Example
One-to-One	Ek entity sirf ek baar relate hoti hai	Person ↔ Passport
One-to-Many	Ek entity multiple entities se relate hoti hai	Department → Many Doctors
Many-to-One	Kai entities ek hi entity se relate hoti hain	Many Surgeries → One Surgeon
Many-to-Many	Dono sides pe multiple entities involved hote hain	Employee ↔ Project

📘 SQL Commands – Interview-Ready

🔹 What are SQL Commands?

> *"SQL commands are like tools — they let you create tables, insert data, fetch records, control access, and manage transactions in a database."*

🧰 Types of SQL Commands

Type	Full Form	Purpose	Common Commands	Hinglish Explanation
DDL	Data Definition Language	Define or change structure of database	`CREATE`, `ALTER`, `DROP`, `TRUNCATE`	Table banana, column add/remove karna
DML	Data Manipulation Language	Modify data inside tables	`INSERT`, `UPDATE`, `DELETE`	Table mein data daalna, badalna, ya hataana
DQL	Data Query Language	Retrieve data from database	`SELECT`	Table se data nikalna (query karna)
DCL	Data Control Language	Control access and permissions	`GRANT`, `REVOKE`	Kisi user ko access dena ya wapas lena
TCL	Transaction Control Language	Manage transactions and data consistency	`COMMIT`, `ROLLBACK`, `SAVEPOINT`	Ek saath multiple changes ko save ya undo karna

🎯 Summary for Interview

> *"SQL commands 5 types ke hote hain — DDL se structure banta hai, DML se data change hota hai, DQL se data fetch hota hai, DCL se access control hota hai, aur TCL se transaction manage hota hai. Har command ka apna role hota hai database ke andar."*

📊 Difference Between `TRUNCATE` and `DELETE` in SQL

🔹 “Table remains intact” ka matlab

Jab hum `DELETE` ya `TRUNCATE` karte hain, **table structure** (columns, data types, constraints) **change nahi hota** — sirf data delete hota hai.

Feature	DELETE Command	TRUNCATE Command
Purpose	Deletes specific rows from a table	Removes all rows from a table
WHERE Clause	✅ Supported	❌ Not supported
Rollback (TCL)	✅ Possible	❌ Not possible
Triggers	✅ Fired	❌ Not fired
Speed	Slower	Faster
Usage	For selective deletion	For bulk deletion
Table Structure	Remains intact (columns, constraints stay)	Remains intact (but identity counters reset)

🔥 What is a Trigger in SQL?

Trigger ek automatic SQL block hota hai jo tab chalta hai jab table pe koi specific event hota hai — jaise `INSERT`, `UPDATE`, ya `DELETE`.

🧠 *Think: Jaise hi koi row delete kare, ek hidden SQL command auto-run ho jaaye — bina manually likhe.*

🧰 Trigger ka Use Case

- Audit logs maintain karna
- Automatically update related tables
- Business rules enforce karna (e.g., salary > 0)
- Notifications ya backups trigger karna

📘 Audit Log ka Matlab

- **Audit Log** ek special record hota hai jo database ya system ke **sabhi important actions/events** ko track karta hai.
- Ye ek **history table** ki tarah hota hai jisme likha jaata hai:
    - Kisne action kiya (user/employee)
    - Kya action hua (INSERT, UPDATE, DELETE)
    - Kab action hua (timestamp)
    - Kis data pe action hua (row details)

🧪 Trigger Fire Kab Hota Hai?

Event	Trigger Type	Example Use
`INSERT`	AFTER INSERT	New employee added → log entry banao
`UPDATE`	BEFORE UPDATE	Salary update hone se pehle check karo
`DELETE`	AFTER DELETE	Row delete hone ke baad audit mein likho

📘 DBMS mein Data Storage Models

1. **File-based storage (basic DBMS)**
    - Data simple files mein store hota hai (text, CSV, etc.)
    - Jaise tum Windows folder mein ek-ek file rakhte ho.
    - Problem: redundancy, searching slow, aur relationships manage karna mushkil.
2. **Hierarchical Model**
    - Data ek **tree structure** mein hota hai.
    - Ek parent → multiple children.
    - Example: Company → Departments → Employees.
    - Jaise Windows Explorer mein ek folder ke andar sub-folders aur files.
3. **Navigational Model**
    - Data ek **network structure** mein hota hai (graph-like).
    - Ek record directly dusre record se linked hota hai pointers ke through.
    - Tum “navigate” karte ho ek record se dusre record tak, jaise linked nodes.
    - Example: Student → Course → Instructor → Department (multiple links).
    - Ye **CODASYL DBMS** style tha (old systems).

🧠 *Think: Hierarchical = tree (folders), Navigational = graph (nodes + links).*

📊 Difference Between DBMS and RDBMS

Feature	DBMS	RDBMS
Full Form	Database Management System	Relational Database Management System
Data Storage	Data stored as files, hierarchical or navigational models	Data stored in tables (relations: rows & columns)
Relationships	No relationships between data files	Relationships defined using Primary Key and Foreign Key
Redundancy	High redundancy (duplicate data)	Reduced redundancy via Normalization
Data Integrity	Limited enforcement	Strong enforcement with constraints (NOT NULL, UNIQUE, CHECK)
Transactions	Basic support, not fully ACID compliant	Full ACID properties (Atomicity, Consistency, Isolation, Durability)
Examples	File System DB, XML DB, IMS	MySQL, Oracle, SQL Server, PostgreSQL
Use Case	Small-scale applications, single-user systems	Large-scale, multi-user enterprise systems

📘 What is a Database System?

- A **database** is an organized collection of structured information stored electronically in a computer system.
- A **Database System = Database + DBMS + Applications**.
- **DBMS (Database Management System)** controls how data is stored, retrieved, and updated.
- It ensures:
    - **Efficient storage & retrieval**
    - **Data integrity** (rules, constraints)
    - **Security & access control**
    - **Multi-user support**

📘 1. Index kya hota hai?

- **Index** ek data structure hai jo database table ke rows ko fast search karne ke liye use hota hai.
- Jaise tum book ke **index page** se directly topic dhoondh lete ho, bina poori book padhe.
- Database mein index ek shortcut banata hai rows tak pahunchne ka.

⚡ 2. Search Speed kaise improve hoti hai?

- Without index → DB ko poori table scan karni padti hai (Full Table Scan).
- With index → DB directly pointer se row tak pahunch jaata hai.
- Result: **Query execution fast** ho jaata hai, specially on large tables.

🌳 Balanced Tree ka Matlab (Database Context)

- **Balanced Tree** ek aisa tree hota hai jisme har level par nodes evenly distributed hote hain.
- Matlab: Root se leaf tak path length **lagbhag equal** hota hai → searching fast hoti hai (O(log n)).
- Database indexes (Clustered/Non-Clustered) mostly **B-Tree / B+ Tree** pe based hote hain.

🧠 *Think: Agar tree unbalanced ho jaaye toh ek side lamba ho jaata hai aur search slow ho jaata hai. Balanced tree ensure karta hai ki har search path short aur equal ho.*

🔑 Key aur Data ka Matlab (Database Example)

- **Key** = Column value jisko tum search karte ho (e.g., `EmployeeID`, `RollNo`).
- **Data** = Us row ka actual record (e.g., Employee ka naam, salary, department).
- Index tree mein **keys sorted order mein store hote hain** aur unse data rows tak shortcut milta hai.

📘 B-Tree (Database Example)

- Keys + Data dono internal nodes mein ho sakte hain.
- Example: `EmployeeID` index banaya hai.
- Tree nodes mein EmployeeID + pointer to row stored hote hain.

```
          [100]
         /     \
     [50]       [150]
    /   \      /     \
 [20]  [70] [120]   [180]
```

👉 Yahan har node mein **key + pointer to data row** hai.

📘 Example: Employee Table with `EmployeeID` Index

Maan lo ek table hai:

EmployeeID	Name	Dept
20	Riya	HR
50	Aman	IT
70	Neha	Sales
100	Mohit	Admin
120	Rahul	Finance
150	Priya	Ops
180	Arjun	Lega

🌳 B-Tree Search for EmployeeID = 120

🔹 Search Path:

1. Root = 100
    - 120 > 100 → go **right child**
2. Node = 150
    - 120 < 150 → go **left child**
3. Node = 120 → ✅ Found

👉 Yes, **120 yahan EmployeeID hai**.

🎯 Result Milega:

Database query engine index se shortcut leke **EmployeeID = 120** wali row return karega:

```
EmployeeID: 120
Name: Rahul
Dept: Finance
Salary: 70,000
```

Summary

> *"Index tree se search karne par tumhe poora row milta hai jisme EmployeeID = 120 hai. Matlab Rahul ka record Finance department aur uski salary ke saath return hoga."*

🌳 B+ Tree Structure (Simplified)

🔹 Internal Nodes (keys only, no data)

```
              [100 | 150]   ← This is ONE root node with two keys
             /     |     \
       [50|70]  [120]   [180]   ← Internal child nodes
       /
    [20]   ← leaf node
```

👉 Yeh **ek node** hai jisme `100` aur `150` dono keys store hain.
👉 Iska matlab hai:

- Left child → values < 100
- Middle child → values between 100 and 150

- Right child → values > 150

🔹 Leaf Nodes (actual data + linked list)

```
[20] → [50] → [70] → [100] → [120] → [150] → [180]
```

👉 Leaf nodes mein **actual data pointers** hote hain (EmployeeID → row in table).
👉 Ye sequentially linked hote hain → range queries fast.

🔎 Example Search

- **Search EmployeeID = 120**
  Root [100 | 150] → 120 > 100 → go middle child → leaf [120] → ✅ Found Rahul (Finance).
- **Range Query EmployeeID BETWEEN 70 AND 150**
  Leaf linked list scan: [70] → [100] → [120] → [150] → ✅ Found Neha, Mohit, Rahul, Priya.

✅ Agar internal node mein key milti hai, toh uske leaf node pe jaate hain?

Bilkul.

- Internal node mein agar key milti hai (jaise 50), toh wo **actual data nahi deta**
- Us key ke corresponding **leaf node** pe jaana padta hai
- Internal node ke pointer se hum us leaf node tak pahunchte hain

👉 Example:

- Internal node: `[50 | 70]`
- Search: `EmployeeID = 50`
- Internal node se pointer follow karke leaf node `[50]` pe jaate hain
- Leaf node se actual row milti hai: Aman, IT Department
-

📌 Important Points

1. **Single balanced tree** hota hai (root → child → leaf).
2. **Internal nodes** sirf keys rakhte hain (guide karte hain).
3. **Leaf nodes** actual data pointers + linked list hote hain.
4. **Range queries** super fast because of linked list.
5. **Clustered index** usually B+ Tree pe implement hota hai.

🧠 Automatic Selection kaise hota hai?

- Jab B+ Tree banaya jaata hai (during indexing), toh database engine **automatically decide karta hai**:
  - Kitne keys ek node mein fit ho sakte hain (based on block size)
  - Kab node ko split karna hai
  - Kaun root banega, kaun child banega

👉 Yeh sab **self-balancing** hota hai — isliye B+ Tree fast aur efficient hota hai.

> *"Tumhare diagram mein root node ek hi hai jisme 2 keys hain — 100 aur 150. Yeh keys automatically select hote hain indexing ke time pe, aur tree self-balance karta hai taaki search fast ho."*

🎯 Summary

> *"Diagram sahi hai — root node mein multiple keys ho sakti hain, internal nodes sirf guide karte hain, aur leaf nodes actual data rakhte hain. Leaf nodes linked list se connected hote hain, isliye range queries fast hoti hain. Agar internal node mein key milti hai, toh uske corresponding leaf node pe jaake data fetch hota hai."*

🚫 When to Avoid Indexes?

- On **small tables** (index overhead > benefit).
- On tables with **frequent INSERT/UPDATE/DELETE** (index maintenance cost high).
- On columns with **low selectivity** (e.g., Gender = Male/Female → index useless).
- When memory/disk overhead is critical.

📘 Easy Explanation: Clustered vs Non-Clustered Index

🔹 1. Clustered Index — "Table hi sorted hoti hai"

- **Data rows physically sorted** hote hain index key ke according.
- Table ke andar hi data ka order change ho jaata hai.
- **Ek hi clustered index** ho sakta hai per table.
- **Leaf nodes** of B+ Tree contain **actual data rows**.

👉 *Example:* Agar `EmployeeID` pe clustered index hai, toh table rows `20, 50, 70, 100, 120...` ke order mein store honge.

🔹 2. Non-Clustered Index — "Pointer se data milta hai"

- Data rows table mein original order mein hote hain.
- Index ek **separate structure** hota hai jisme key + pointer hota hai.
- **Multiple non-clustered indexes** allowed.
- **Leaf nodes** contain **row pointers**, not actual data.

👉 *Example:* Agar `Name` pe non-clustered index hai, toh index mein `Aman → pointer to row`, `Mohit → pointer to row` hoga — table ke andar rows scattered ho sakte hain.

🔍 Visual Analogy

Feature	Clustered Index	Non-Clustered Index
Data Order	Table rows sorted	Table rows unsorted
Index Location	Inside table	Separate structure
Leaf Node Content	Actual data rows	Row pointers
Count per Table	Only one	Many allowed
Speed for Range	Fast (linked leaf nodes)	Slower (extra lookup needed)
Use Case	Primary key, range queries	Searching by name, email, etc.

🔒 Deadlock in Database (DBMS) Context

🔹 Definition:

Deadlock tab hota hai jab **do ya zyada transactions ek dusre ke resources ka intezaar karte hain**, aur koi bhi aage nahi badh pata.

📘 Real-Life Example (Database)

Scenario:

- **T1** locks **Row A**, wants **Row B**
- **T2** locks **Row B**, wants **Row A**

Result:

- T1 → wait kar raha hai Row B ke liye
- T2 → wait kar raha hai Row A ke liye
- Dono ek dusre ka intezaar kar rahe hain → **Deadlock**

🔍 Deadlock Detection in DBMS

- DBMS **periodically check karta hai** ki koi cycle to nahi ban gayi (Resource Allocation Graph)
- Agar cycle milti hai → deadlock detected
- DBMS fir:
    - **Rollback** karta hai ek transaction ko
    - Ya **kill** karta hai least priority transaction

🛡️ Deadlock Prevention Techniques

1. **Lock ordering**:
    - Sab transactions ko resources ek fixed order mein lock karna hota hai
    - Example: Always lock Row A before Row B
2. **Timeouts**:
    - Agar transaction zyada time tak wait karta hai → rollback kar diya jaata hai
3. **Avoid Hold and Wait**:
    - Transaction ko sab locks ek saath lene padte hain — partial locking allowed nahi

🚦 Deadlock Avoidance (Advanced)

- **Wait-Die** and **Wound-Wait** protocols:
    - Based on transaction timestamps
    - Older transaction ya to wait karega ya younger ko abort karega

🎯 Summary

> *“Database mein deadlock tab hota hai jab transactions ek dusre ke locked rows ka intezaar karte hain. Isse system freeze ho jaata hai. DBMS deadlock detect karta hai graph se, prevent karta hai lock ordering ya timeouts se, aur avoid karta hai smart protocols se.”*

👁️ View

- **View** = Ek virtual table jo ek SQL query ke result ko represent karta hai.
- Data **physically store nahi hota**, bas query ke through dikhaya jaata hai.

🔹 Materialized View

- **Materialized View** = Query ka result **physically store hota hai** (cache ki tarah).

- Fast read ke liye use hota hai, par refresh/update karna padta hai.

🔄 Cursor

- **Cursor** = Ek pointer jo query ke result set ko row-by-row traverse karta hai.
- Useful jab tumhe **row-wise processing** karni ho (loops ke andar).
- Downside: slow hota hai compared to set-based operations.

📘 Transaction kya hota hai?

- **Transaction** = ek logical unit of work in DB.
- Matlab ek group of SQL operations jo **ACID properties** follow karta hai (Atomicity, Consistency, Isolation, Durability).
- Example: Agar tum bank transfer karte ho:
    1. Account A se ₹500 debit
    2. Account B me ₹500 credit 👉 Ye dono ek **transaction** hai. Agar ek fail ho jaaye toh pura rollback ho jaata hai.

🔐 Isolation Levels (with examples)

Isolation level decide karta hai ki ek transaction doosre transaction ke data ko kab dekh sakta hai. Lower isolation → zyada concurrency, par zyada problems.
Higher isolation → safe, par slow.

1️⃣ Read Uncommitted

- **Allowed:** Transaction doosre ka **uncommitted data** read kar sakta hai.
- **Problem: Dirty Read** possible.
- 🔍 *Example*:
    - T1: Balance update karta hai 5000 → 7000 (not committed yet)
    - T2: Balance read karta hai → 7000
    - T1 rollback kar deta hai → actual balance 5000 hai, par T2 ne galat data read kar liya.

2️⃣ Read Committed

- **Allowed:** Sirf **committed data** read hoga.
- **Problem: Non-repeatable Read** possible.
- 🔍 *Example*:
    - T1: Balance read karta hai → 5000
    - T2: Balance update karke commit karta hai → 7000
    - T1 dobara read kare → ab 7000 milta hai (same query, different result).

3️⃣ Repeatable Read

- **Allowed:** Same row baar-baar read karne pe **same result** milega.

- **Problem: Phantom Read** possible.
- 🔍 *Example*:
  - T1: `SELECT * FROM Accounts WHERE type='Savings'` → 5 rows
  - T2: Ek naya savings account insert karta hai aur commit karta hai
  - T1 dobara same query kare → ab 6 rows milti hain (extra row = Phantom Read).

4️⃣ Serializable

- **Allowed:** Sabse strict level. Transactions ekdum serial order mein execute hote hain.
- **Problem:** Koi anomaly nahi (Dirty, Non-repeatable, Phantom sab avoid).
- 🔍 *Example*:
  - Agar T1 aur T2 dono savings accounts pe kaam kar rahe hain, DB unko serially run karega → ek ke baad ek, no conflict.

⚠️ Problems Recap

Problem	Cause	Avoided by Isolation Level
Dirty Read	Uncommitted data read	Read Committed ↑
Non-repeatable Read	Same row read twice, value changed	Repeatable Read ↑
Phantom Read	Same query returns different row count	Serializable only

🎯 Summary

> *"Transaction ek logical unit of work hai jo ACID follow karta hai. Isolation levels decide karte hain ki ek transaction doosre ka data kab dekh sakta hai. Read Uncommitted sabse risky hai (Dirty Read possible), Read Committed thoda safe hai (Non-repeatable Read possible), Repeatable Read aur safe hai (Phantom Read possible), aur Serializable sabse strict hai jisme koi anomaly nahi hoti."*

🔄 Concurrency ka Matlab

- **Concurrency** = ek hi time pe multiple transactions chal rahe hain.
- Matlab ek database system ek saath **parallel users/transactions** ko handle kar raha hai.

👉 Example:

- T1: Bank transfer kar raha hai (A → B ₹500)
- T2: Balance check kar raha hai (Account A)

- T3: Naya account open kar raha hai

Ye sab ek hi waqt chal rahe hain → **concurrent transactions**.

⚡ Zyada Concurrency ka Matlab

- Jab isolation level **low** hota hai (jaise Read Uncommitted, Read Committed), toh DB ek saath **zyada transactions allow karta hai** → concurrency high hoti hai.
- Par iska side effect: **problems/anomalies** ho sakti hain (Dirty Read, Non-repeatable Read, Phantom Read).

👉 Simple analogy:

- **Low isolation = zyada concurrency, kam safety**
- **High isolation = kam concurrency, zyada safety**

🎯 Summary

> *"Zyada concurrency ka matlab hai ek hi time pe zyada transactions parallel chal rahe hain. Low isolation levels concurrency badhate hain par anomalies ka risk hota hai. High isolation levels concurrency kam karte hain par data consistency safe ho jaati hai."*

Other Questions :) No End

📘 Serializable kya hota hai?

🔒 Conflict Serializability

⚙️ CCP (Concurrency Control Protocols)

📘 Relational Algebra ka Matlab

🔑 Important Operations with Proper Examples