UNIT-IV: SCHEMA REFINEMENT

1) State and explain Third normal form with an example. 5 M

Ans:

Third Normal Form (3NF)

Definition:
A relation (table) is said to be in **Third Normal Form (3NF)** if it satisfies the following conditions:

1. It is in **Second Normal Form (2NF)**.
2. **No transitive dependency** exists — that is, **no non-prime attribute** (attribute that is not part of any candidate key) should depend on another **non-prime attribute**.

In simple words, in 3NF, **every non-key attribute should depend only on the primary key** and not on any other non-key attribute.

Explanation:

A **transitive dependency** means:

If A → B and B → C, then A → C is a transitive dependency.

Here, if A is the primary key, and C is dependent on B (a non-key attribute), then this violates 3NF.

Example:

Table: Student

Student_ID	Student_Name	Department_ID	Department_Name
1	Ravi	D1	Computer Science
2	Priya	D2	Electronics
3	Arjun	D1	Computer Science

- **Primary Key:** Student_ID
- **Dependencies:**
    - Student_ID → Student_Name, Department_ID, Department_Name
    - Department_ID → Department_Name

Here, **Department_Name** depends on **Department_ID**, not directly on **Student_ID**.
This is a **transitive dependency**, because:
Student_ID → Department_ID → Department_Name

Hence, the table **is not in 3NF**.

Conversion to 3NF:

We remove the transitive dependency by splitting the table into two:

1. Student Table

Student_ID	Student_Name	Department_ID
1	Ravi	D1
2	Priya	D2
3	Arjun	D1

2. Department Table

Department_ID	Department_Name
D1	Computer Science
D2	Electronics

Now:

- In **Student**, all non-key attributes depend only on the primary key (Student_ID).
- In **Department**, all non-key attributes depend only on Department_ID.
  So, both tables are now in **Third Normal Form (3NF)**.

Summary:

Normal Form	Condition
1NF	No repeating groups (atomic values only)
2NF	No partial dependency (every non-key attribute depends on the whole primary key)
3NF	No transitive dependency (non-key attributes depend only on the key)

2) Explain dependency preserving decomposition with examples 5 M

Ans:

A decomposition of R into R_1, R_2, ..., Rn is **dependency preserving** if:

$$(F_1 \cup F_2 \cup ... \cup Fn)^+ = F^+$$

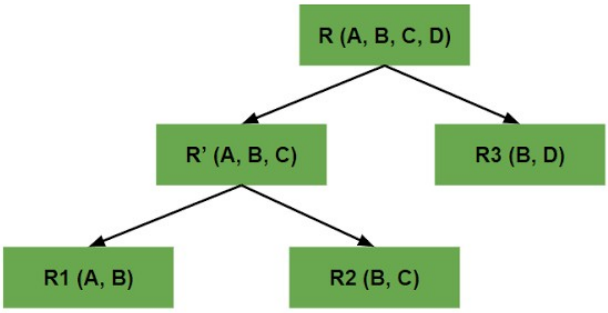
where

- F = original set of FDs on R
- Fi = projection of F onto Ri (functional dependencies that involve only attributes of Ri)
- F^+ = closure of F

A dependency preserving decomposition is a database decomposition where all the original functional dependencies (FDs) of a relation can be checked within the decomposed sub-relations without needing a join operation.

This is important for maintaining data integrity and efficiency, as it prevents anomalies and ensures that the original constraints can still be enforced on the new, smaller tables.

Diagram showing decomposing a table R(A, B, C, D)



Example 1: Dependency Preserved Decomposition

Let's consider a relation:

R(A, B, C)
and functional dependencies:

F = { A → B, B → C }

We decompose **R** into:

- **R_1(A, B)**
- **R_2(B, C)**

Now,

- In **R_1**, we can enforce **A → B**.
- In **R_2**, we can enforce **B → C**.

So all dependencies in F can be checked **locally** in their respective tables.
Therefore, this decomposition is **dependency preserving**.

Example 2: Not Dependency Preserving

Consider a relation:

R(A, B, C)
and functional dependencies:

F = { A → B, B → C }

If we decompose **R** into:

- **R_1(A, B)**
- **R_2(A, C)**

Now check where each FD can be enforced:

- **A → B** can be enforced in **R_1**
- But **B → C** cannot be enforced in **R_1** or **R_2**, because:
    - **R_1** doesn't have C
    - **R_2** doesn't have B

To check **B → C**, we'd have to **join R_1 and R_2**.
Hence, this decomposition is **not dependency preserving**.

3) Discuss about "the concept of surrogate" key with example 5 M

Ans:

Definition:

A surrogate key is an artificial or system-generated key used to uniquely identify each record in a table. It has no business meaning or real-world significance — it simply acts as a unique identifier. It is usually created by the database system itself (for example, an auto-increment number).

Explanation:

- Sometimes, a table doesn't have a natural or simple primary key.
- In such cases, we create a surrogate key to uniquely identify each row.
- Surrogate keys are not derived from application data — they are just identifiers.

For example:

- Auto-increment integers (1, 2, 3, 4, …)
- System-generated unique IDs (like GUIDs or UUIDs)

Example:

Table: Employee

S.No.	Emp_Name	Email	Phone
1	Ravi	ravi@gmail.com	9876543210
2	Priya	priya@gmail.com	8765432109
3	Arjun	arjun@gmail.com	9988776655

Here:

- S.No. is a surrogate key, automatically generated by the system.
- It has no business meaning — it's just used to uniquely identify each employee.

- Even if two employees share the same name or phone number, the S.No remains unique**.**

Advantages:

1. Uniqueness — Guarantees that every record can be uniquely identified.
2. Stability — Surrogate keys don't change even if business data changes.
3. Simplifies relationships — Easier to use as foreign keys in other tables.
4. Independence — Not tied to any real-world attribute, avoiding future key changes.

Disadvantages:

1. No real-world meaning — It doesn't provide information about the record.
2. Extra column — Adds one more attribute to the table.
3. Possible confusion — Users may prefer meaningful (natural) identifiers.

4) Define BCNF. How does BCNF differ from 3NF? Explain with an example 5 M

Ans:

Definition:

A relation is said to be in Boyce–Codd Normal Form (BCNF) if it satisfies the following condition:

For every functional dependency (FD) X → Y in the relation, X must be a super key.

In BCNF, no non-prime attribute should depend on anything other than a super key.

BCNF is a stronger version of Third Normal Form (3NF).
It removes all anomalies that can still exist in 3NF relations.

Difference Between 3NF and BCNF

	3NF	BCNF
Condition	For every FD X → Y, either: 1 X is a super key, or 2 Y is a prime attribute (part of some candidate key).	For every FD X → Y, X must be a super key (no exceptions).
Strength	Weaker — allows some dependencies that may cause anomalies.	Stronger — stricter form of 3NF.
Data Anomalies	Some anomalies may still remain.	Removes all anomalies.

	3NF	BCNF
Relationship	Every BCNF relation is also in 3NF.	Every 3NF relation is not necessarily in BCNF.

Example:

Relation: R(Student_ID, Course, Instructor)

Functional Dependencies:

1. (Student_ID, Course) → Instructor
2. Instructor → Course

Step 1: Candidate Key

- From (1), we can see (Student_ID, Course) is a key.
- But since Instructor → Course, Instructor is not a key.

Step 2: Check 3NF Condition

For FD Instructor → Course:

- Instructor is not a super key, ❌
- Course is a prime attribute? No ❌

Hence, it violates 3NF, and therefore also BCNF.

Step 3: Decompose into BCNF

We decompose R into two relations:

1. R_1(Instructor, Course) — (because Instructor → Course)
2. R_2(Student_ID, Instructor)

Now:

- In R_1, Instructor → Course (Instructor is a key) ✅
- In R_2, (Student_ID, Instructor) is a key ✅

✅ Both are in BCNF

Summary Table:

Property	3NF	BCNF
Removes transitive dependencies	✔	✔

Property	3NF	BCNF
Removes overlapping candidate keys	✗	✓
Allows non-key determinants if dependent is prime	✓	✗
Strength	Moderate	Stronger, stricter

BCNF eliminates all redundancy by ensuring every determinant is a super key, while 3NF is slightly more relaxed to preserve dependency and practicality.

5) What is the purpose of Schema Refinement? 3 M

Ans:

Definition:

Schema Refinement is the process of **improving the design of database tables** (schemas) to eliminate **data redundancy**, **update anomalies**, and **inconsistencies** using **normalization techniques**.

Schema Refinement aims to produce a **well-structured, anomaly-free, and consistent database design** by systematically applying **normalization rules**

Purpose / Objectives:

1. **Eliminate Data Redundancy**
    - Avoid storing the same information in multiple places.
    - Reduces storage waste and inconsistency.
2. **Avoid Anomalies**
    - Prevents **insertion**, **deletion**, and **update anomalies** that occur due to poor schema design.
3. **Preserve Data Integrity**
    - Ensures that all data dependencies (functional dependencies) are maintained correctly.
4. **Improve Data Consistency and Efficiency**
    - Makes the database easier to maintain and update without errors.
5. **Achieve a Good Normal Form**
    - Converts relations into **1NF, 2NF, 3NF, BCNF**, etc., to ensure a well-structured design.

6) Explain about Lossless join and Lossy join decomposition with examples 7 M

Ans:

Definition:

When a relation (table) is **decomposed** into two or more smaller relations, it is important to ensure that when we **join them back**, we can **reconstruct the original relation without losing or gaining information**.

This property is known as the **lossless join property**.

Lossless Join Decomposition

Definition:

A decomposition of a relation **R** into R_1 and R_2 is said to be **lossless join** if:

Joining R_1 and R_2 using a **natural join** gives back exactly the original relation **R** — **no data is lost or added.**

Condition:

Decomposition of **R** into R_1 and R_2 is **lossless** if

$(R_1 \cap R_2) \rightarrow R_1$ or $(R_1 \cap R_2) \rightarrow R_2$

That means the common attributes between the decomposed tables **functionally determine** one of the relations.

Example:

Relation R(A, B, C)
Functional Dependency: **A → B**

Decompose R into:

- **R_1(A, B)**
- **R_2(A, C)**

Now,
Common attribute = **A**
and **A → B** holds true.

Hence, decomposition is **lossless join**.

Joining R_1 and R_2:

A	B	C
1	X	P
2	Y	Q

→ After join, we get back the **original relation R**, with no missing or extra data.

Lossy Join Decomposition

Definition:

A decomposition of **R** into R_1 and R_2 is said to be **lossy join** if:

Joining R_1 and R_2 produces **extra (spurious) tuples** or **loses original tuples** — meaning information is lost or distorted.

Example:

Relation R(A, B, C)
No functional dependency between attributes.

Decompose R into:

- **R_1(A, B)**
- **R_2(B, C)**

Common attribute = **B**

But **B → A** or **B → C** does **not** hold.

❌ Hence, it is a **lossy decomposition**.

Original R:

A	B	C
1	X	P
2	X	Q

After decomposition:

R_1(A, B):

A	B
1	X
2	X

R_2(B, C):

B	C
X	P
X	Q

After joining back:

A	B	C
1	X	P

A	B	C
1	X	Q
2	X	P
2	X	Q

→ These are **extra tuples** (spurious data), so information is lost or distorted.

❌ **Lossy Join**

✅ **Difference Between Lossless and Lossy Join**

Feature	Lossless Join	Lossy Join
Definition	Original relation can be perfectly reconstructed by join.	Join produces extra (spurious) tuples or loses data.
Condition	$(R_1 \cap R_2) \rightarrow R_1$ or $(R_1 \cap R_2) \rightarrow R_2$	No such dependency exists.
Data Integrity	Maintained ✅	Lost ❌
Result of Join	Exact original table	Incorrect (extra or missing tuples)

Hence:

Lossless Join = No data loss (good decomposition)
Lossy Join = Data loss or spurious data (bad decomposition)

7) What is Normal Form? What are the differences between 1NF, 2NF, 3NF? 5 M
Ans:

A **Normal Form** is a **set of rules** used in **database normalization** to design efficient, consistent, and redundancy-free database tables.

A relation is said to be in a *normal form* if it satisfies certain conditions that eliminate **data redundancy** and **anomalies** (insertion, deletion, and update).

Normalization is done step-by-step through **various normal forms** —
1NF → 2NF → 3NF → BCNF → 4NF …

Purpose of Normalization:

1. To remove duplicate data (redundancy).
2. To ensure data integrity and consistency.
3. To simplify maintenance and improve database performance.

Differences between 1NF, 2NF, and 3NF

Feature	First Normal Form (1NF)	Second Normal Form (2NF)	Third Normal Form (3NF)
Definition	A relation is in 1NF if it contains **only atomic (indivisible) values** — no repeating groups or arrays.	A relation is in 2NF if it is in **1NF** and **no partial dependency** exists (i.e., every non-key attribute depends on the *whole primary key*).	A relation is in 3NF if it is in **2NF** and has **no transitive dependency** (non-key attributes depend only on the key).
Main Focus	Remove repeating groups and ensure atomic values.	Remove **partial dependencies**.	Remove **transitive dependencies**.
Depends on	Structure of attributes (atomic values).	Functional dependencies on **part of a composite key**.	Functional dependencies between **non-key attributes**.
Type of Anomaly Removed	Repeating data anomalies.	Partial dependency anomalies.	Transitive dependency anomalies.
Example	**Not in 1NF:** Student(ID, Subjects) → {101, (Math, Science)} **In 1NF:** Student(ID, Subject) → {101, Math}, {101, Science}	**Not in 2NF:** Student_Course(Student_ID, Course_ID, Student_Name) Here, Student_Name depends only on Student_ID (partial). **In 2NF:** Split into: Student(Student_ID, Student_Name) Course_Enroll(Student_ID, Course_ID)	**Not in 3NF:** Student(Student_ID, Dept_ID, Dept_Name) Dept_Name depends on Dept_ID (transitive). **In 3NF:** Student(Student_ID, Dept_ID) Department(Dept_ID, Dept_Name)
Result	Atomic structure.	Eliminates partial dependency.	Eliminates transitive dependency.

Summary:

Normal Form	Removes	Based On
1NF	Repeating groups / non-atomic data	Atomic values
2NF	Partial dependency	Full functional dependency
3NF	Transitive dependency	Only key-based dependency

In short:

Normalization organizes data in multiple steps (1NF, 2NF, 3NF…) to make the database **efficient**, **consistent**, and **free from redundancy and anomalies**.

8) Define Functional Dependency? Explain trivial, non-trivial, multi valued dependency 5 M

Ans:

Definition:

A **Functional Dependency (FD)** is a **relationship between two sets of attributes** in a relation (table).

It is denoted as:

X → Y

which means:

Attribute (or set of attributes) **X** *functionally determines* attribute **Y**,
i.e., if two tuples have the same value for **X**, they must have the same value for **Y**.

Example:

In a relation **Student(Reg_No, Name, Dept, Phone):**

If Reg_No → Name, Dept, Phone

then knowing a student's **Reg_No** uniquely determines their **Name**, **Dept**, and **Phone**.
So, **Reg_No** is said to *functionally determine* the other attributes.

Types of Functional Dependencies

1. Trivial Functional Dependency

Definition:

A functional dependency **X → Y** is **trivial** if **Y** is a subset of **X**.
It is always true for any relation.

Example:

- {Reg_No, Name} → Name
  Here, **Name** is part of **{Reg_No, Name}**, so it's **trivial**.

Meaning: It gives no new information about the relation.

2. Non-Trivial Functional Dependency

Definition:

A functional dependency **X → Y** is **non-trivial** if **Y** is **not a subset** of **X**.

That means Y contains attributes that are not already in X.

Example:

- Reg_No → Name
  Here, **Name** is not a part of **Reg_No**, so it is **non-trivial**.

Meaning: It shows a real dependency between different attributes.

3. Multivalued Dependency (MVD)

Definition:

A **Multivalued Dependency (MVD)** occurs when an attribute in a table **determines multiple independent values** of another attribute.

It is denoted as:

X →→ Y

It means: For a given value of X, there can be **multiple independent values of Y**, and Y is **independent of other attributes**.

Example:

Consider a relation **Student(Reg_No, Hobby, Skill)**

Reg_No	Hobby	Skill
101	Music	Coding
101	Painting	Coding
101	Music	Writing
101	Painting	Writing

Here,

- **Reg_No →→ Hobby**
- **Reg_No →→ Skill**

Both Hobby and Skill depend on Reg_No **independently**.

So this is a **multivalued dependency**.

Summary Table:

Type	Symbol	Condition	Example
Trivial FD	$X \rightarrow Y$	$Y \subseteq X$	$\{A, B\} \rightarrow A$
Non-Trivial FD	$X \rightarrow Y$	$Y \not\subset X$	Roll_No → Name
Multivalued Dependency (MVD)	$X \twoheadrightarrow Y$	One X value determines many Y values independently	Student_ID →→ Hobby

In short:

- **Functional Dependency (FD):** Relation between attributes.
- **Trivial FD:** Obvious dependency ($Y \subseteq X$).
- **Non-Trivial FD:** Meaningful dependency (Y not in X).
- **MVD:** One attribute determines multiple independent values of another.

9) Define MVD. Explain 4NF in detail. 5 M

Ans:

Definition of Multivalued Dependency (MVD)

A **Multivalued Dependency (MVD)** occurs in a relation when **one attribute in a table uniquely determines multiple independent values of another attribute**.

It is denoted as:

X →→ Y

and means:

For a given value of **X**, there can be **multiple independent values of Y**, and **Y** is independent of other attributes in the relation.

Example of MVD:

Relation: Student(Reg_No, Hobby, Skill)

Reg_No	Hobby	Skill
101	Music	Coding
101	Painting	Coding
101	Music	Writing
101	Painting	Writing

Here:

- One student (Reg_No = 101) can have **multiple hobbies** and **multiple skills**.
- The set of hobbies and skills are **independent**.

Hence,

Reg_No →→ Hobby and **Reg_No →→ Skill**

This is a **multivalued dependency**.

Fourth Normal Form (4NF)

Definition:

A relation is said to be in **Fourth Normal Form (4NF)** if:

1. It is in **Boyce–Codd Normal Form (BCNF)**.
2. It has **no non-trivial multivalued dependencies** other than a dependency on a **superkey**.

A table is in **4NF** if no attribute in it has multiple independent multi-valued relationships with the same key.

Why 4NF is Needed:

Even after achieving BCNF, **multivalued dependencies** can still cause **data redundancy** and **update anomalies**.
4NF removes these by ensuring that **each independent multivalued fact is stored separately**.

Example:

Relation: Student(Reg_No, Hobby, Skill)
As shown earlier,

- Reg_No →→ Hobby
- Reg_No →→ Skill

Problem:
Data redundancy — each Hobby-Skill combination is repeated unnecessarily.

Decomposition to Achieve 4NF:

Decompose the relation into two independent tables:

1. **R_1(Reg_No, Hobby)**
2. **R_2(Reg_No, Skill)**

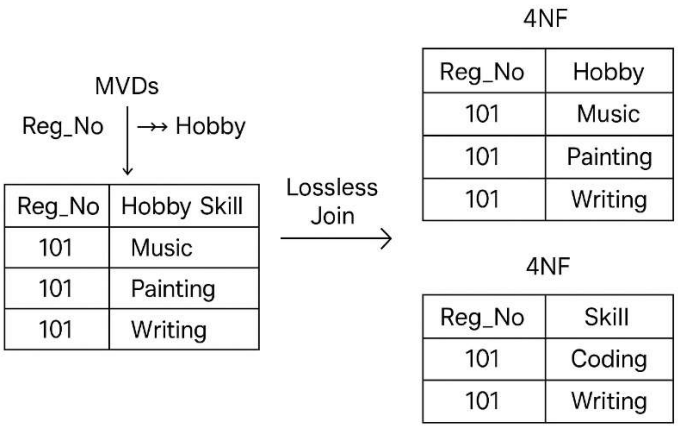
Now:

- Each relation represents **one independent multivalued fact**.
- Both are **free from redundancy**.
- Together they can be **joined losslessly** using Reg_No.

✅ Both **R_1** and **R_2** are now in **Fourth Normal Form (4NF)**.

Key Points / Summary:

Concept	Explanation
MVD (X →→ Y)	X determines multiple independent values of Y
Condition for 4NF	Relation must be in BCNF and have no non-trivial MVD except on a super key
Goal of 4NF	Remove redundancy due to independent multivalued dependencies
Decomposition	Relation is split into smaller relations (lossless join, dependency preserved)



In short:

Fourth Normal Form (4NF) eliminates redundancy caused by **multivalued dependencies**, ensuring that each fact is represented only once and that all MVDs are based on a **super key**.

10) Write the properties of functional dependencies. 5 M

Ans:

Functional Dependencies (FDs) follow certain **mathematical rules** called **Armstrong's Axioms**. These are also called as Inference Rules(IR).
These properties help in **inferring all possible FDs** from a given set and are essential for **normalization** and **schema refinement**.

1 Reflexivity (Trivial FD)

If **Y** is a subset of **X**, then

X → Y

Example:
If we have attributes (A, B), then

(A, B) → A
Because A is part of (A, B).

2 Augmentation (Additivity)

If **X → Y**, then adding the same attributes **Z** to both sides gives:

XZ → YZ

Example:
If Roll_No → Name,
then (Roll_No, Dept) → (Name, Dept)

(We can add **Dept** to both sides.)

3 Transitivity

If **X → Y** and **Y → Z**, then

X → Z

Example:
If Roll_No → Dept and Dept → HOD,
then Roll_No → HOD

4 Union (Additive Rule)

If **X → Y** and **X → Z**, then

X → YZ

Example:
If Roll_No → Name and Roll_No → Dept,
then Roll_No → (Name, Dept)

5 Decomposition (Projective Rule)

If **X → YZ**, then

X → Y and **X → Z**

Example:
If Roll_No → (Name, Dept),
then Roll_No → Name
and Roll_No → Dept

6 Pseudo Transitivity

If **X → Y** and **WY → Z**, then

WX → Z

Example:
If Roll_No → Dept and (Dept, Course) → Faculty,
then (Roll_No, Course) → Faculty

🔹 **Summary Table**

Property	Rule	Explanation / Example
Reflexivity	If Y ⊆ X → X → Y	(A,B) → A
Augmentation	If X → Y → XZ → YZ	Roll_No → Name ⇒ (Roll_No, Dept) → (Name, Dept)
Transitivity	If X → Y and Y → Z ⇒ X → Z	Roll_No → Dept, Dept → HOD ⇒ Roll_No → HOD
Union	If X → Y and X → Z ⇒ X → YZ	Roll_No → Name, Roll_No → Dept ⇒ Roll_No → (Name, Dept)
Decomposition	If X → YZ ⇒ X → Y and X → Z	Roll_No → (Name, Dept) ⇒ Roll_No → Name
Pseudo Transitivity	If X → Y and WY → Z ⇒ WX → Z	Roll_No → Dept, (Dept, Course) → Faculty ⇒ (Roll_No, Course) → Faculty

These properties (Inference Rules) are used to **derive**, **verify**, and **simplify** functional dependencies — an essential step in **database normalization**.

11) Explain BCNF and 5NF with examples 5 M

Ans:

1️⃣ **Boyce–Codd Normal Form (BCNF)**

Definition:

A relation is in **BCNF** if:

For every **non-trivial functional dependency** (X → Y),
X is a **super key** of the relation.

🟢 **In short:**
Every determinant must be a **candidate key** or **super key**.

Why BCNF is needed:

Even if a relation is in **3NF**, **anomalies** may still exist when:

- A **non-prime attribute** determines part of a key, or

- There are **overlapping candidate keys**.

Example:

Relation: Course_Faculty(Course, Faculty, Room)

Course	Faculty	Room
DBMS	A	101
DBMS	A	102
CN	B	103

Functional Dependencies:

1. Course → Faculty
2. Faculty → Room

Here,

- **Course** is not a super key, but determines Faculty.
- **Faculty** is not a super key, but determines Room.
  → Violates BCNF.

BCNF Decomposition:

Decompose into two relations:

1. **R_1(Course, Faculty)**
2. **R_2(Faculty, Room)**

Now:

- In R_1 → Course is a key.
- In R_2 → Faculty is a key.

✅ Both relations are now in **BCNF**.
✅ Anomalies removed.

Advantages of BCNF:

- Eliminates redundancy due to functional dependencies.
- Prevents update, insertion, and deletion anomalies.

2️⃣ **Fifth Normal Form (5NF) — *Also known as Project-Join Normal Form (PJNF)***

Definition:

A relation is in **5NF** if:

It is in **4NF**, and every **join dependency** in the relation is implied by the **candidate keys** of the relation.

🟢 **In simple words:**
A table is in **5NF** if it cannot be further decomposed into smaller tables **without losing data** (lossless join).

Purpose of 5NF:

To eliminate **redundancy** that arises due to **join dependencies** (complex relationships involving 3 or more tables).

Example:

Relation: Course_Text_Author(Course, Textbook, Author)

Course	Textbook	Author
DBMS	T1	A1
DBMS	T1	A2
DBMS	T2	A1
DBMS	T2	A2

- Each course can have multiple textbooks.
- Each textbook can have multiple authors.
- Each author can write for multiple textbooks of the same course.

No single attribute functionally determines the others — this is a **join dependency**.

Decomposition to Achieve 5NF:

Decompose into three tables:

1. **R_1(Course, Textbook)**
2. **R_2(Textbook, Author)**
3. **R_3(Course, Author)**

Now, the original relation can be **reconstructed** by joining R_1, R_2, and R_3 —
✅ **without redundancy**
✅ **without losing any information**

Hence, it is in **5NF**.

✅ **Summary Table:**

Normal Form	Removes	Condition	Example Issue Solved
BCNF	Redundancy due to overlapping FDs	Every determinant must be a super key	Course → Faculty, Faculty → Room
5NF	Redundancy due to complex join dependencies	No non-trivial join dependency exists except those implied by candidate keys	Course–Textbook–Author relation

In short:

- **BCNF** ensures all dependencies are on super keys (removes FD anomalies).
- **5NF** ensures no redundancy remains due to join dependencies (most refined form).

12) What are the advantages of normalized relations over the un-normalized relations? 5 M

Ans:

Normalization is the process of organizing data in a database to reduce redundancy and improve data integrity. Normalized relations offer several benefits compared to unnormalized (raw or redundant) tables.

1️⃣ **Eliminates Data Redundancy**

- In normalized tables, each piece of information is stored **only once**.
- Prevents repetition of data across multiple rows.

✅ **Example:**
In an unnormalized table, a student's department name might be repeated for every course enrolled. Normalization separates it into two tables — Student and Department, reducing duplication.

2️⃣ **Prevents Update, Insertion, and Deletion Anomalies**

- **Update anomaly:** Changing data in one place automatically reflects everywhere.
- **Insertion anomaly:** New data can be added without requiring unnecessary information.
- **Deletion anomaly:** Removing a record doesn't lead to loss of useful information.

✅ Normalized design ensures **data consistency** across all operations.

3️⃣ **Improves Data Integrity and Consistency**

- Data dependencies are clearly defined through **keys** and **foreign keys**.
- Ensures that all references between tables remain valid.

✅ Example: Every student's department must exist in the Department table — enforcing **referential integrity**.

4️⃣ Easier Maintenance and Flexibility

- Changes to the structure or data are easier because each fact is stored in only one place.
- The database is easier to scale and adapt to new requirements.

✅ Example: Adding new attributes like Faculty_Email affects only one table, not multiple redundant copies.

✅ In short:

Normalized relations lead to a **cleaner, consistent, and efficient** database design,
whereas **unnormalized relations** cause **redundancy, anomalies, and maintenance difficulties.**

*****